**Prediksi Kecacatan Perangkat Lunak Berdasarkan Penambangan Data di Repository Rekayasa Perangkat Lunak**

**DRAFT TESIS**

**BAB I**

**Oleh**

**Fernando Maruli Tua Simangunsong**

**23516301**

**(Program Studi Magister Informatika)**

****

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**2018**

# DAFTAR ISI

[DAFTAR ISI ii](#_Toc516055064)

[DAFTAR TABEL iii](#_Toc516055065)

[DAFTAR GAMBAR iv](#_Toc516055066)

[BAB I PENDAHULUAN 5](#_Toc516055067)

[I.1 Latar Belakang 5](#_Toc516055068)

[I.2 Rumusan Masalah 9](#_Toc516055069)

[I.3 Hipotesis Awal 9](#_Toc516055070)

[I.3 Tujuan Penelitian 10](#_Toc516055071)

[I.4 Batasan Masalah 10](#_Toc516055072)

[I.5 Metodologi Penelitian 10](#_Toc516055073)

[BAB II STUDI LITERATUR 12](#_Toc516055074)

[II.1 Peta Literatur 12](#_Toc516055075)

[II.X Kecacatan Perangkat lunak 12](#_Toc516055076)

[II.X.1 Prediksi menggunakan ukuran dan kompleksitas metrik 13](#_Toc516055077)

[II.X.2 Prediksi menggunakan metrik pengujian 14](#_Toc516055078)

[II.X.3 Prediksi menggunakan data kualitas proses pengembangan 15](#_Toc516055079)

[II.X Sistem Manajemen Konfigurasi 16](#_Toc516055080)

[II.X.1 Repositori Perangkat Lunak 18](#_Toc516055081)

[II.X.2 Pelaporan Bug Pada SCM 20](#_Toc516055082)

[II.4 Penambangan Repositori 20](#_Toc516055083)

[DAFTAR PUSTAKA 22](#_Toc516055084)

# DAFTAR TABEL

# DAFTAR GAMBAR

[Gambar 1. Siklus Proses Evolusi Perangkat Lunak 6](#_Toc510471605)

[Gambar 2. *Process Mining Meta Model.* 9](#_Toc510471606)

# BAB I PENDAHULUAN

## I.1 Latar Belakang

Karateristik Perangkat lunak yang berkualitas adalah perangkat lunak yang bebas atau memiliki sedikit mungkin kesalahan *(fault)* dan kegagalan (failure) dalam periode waktu yang telah di tentukan. Kegagalan pada perangkat lunak adalah suatu kondisi dimana perangkat lunak tidak memenuhi persyaratan yang telah ditentukan atau tidak sesuai dengan harapan pengguna karena mengandung cacat (*defect*).

Cacat perangkat lunak dapat menyebabkan *fault* atau *failure.* Hal ini akan membuat kerugian besar karena produk yg dirilis tidak dapat diterima oleh konsumen. Kepuasan konsumen adalah hal penting dalam dunia industri perangkat lunak. Oleh karena itu, perangkat lunak berkualitas tinggi harus dipenuhi melalui pencapaian persyaratan konsumen, pengembangan produk sesuai dengan anggaran, waktu, dan sumber daya yang tersedia[1].

Terminologi *defect* menurut IEEE *Standart Glosaary of Software Engineering* adalah kesalahan langkah, proses dan pendefinisian data pada program komputer atau perangkat lunak. Dalam istilah sederhana, *defect* adalah penyimpangan dari persyaratan yang sesuai dengan spesifikasi kebutuhan perangkat lunak (SKPL) yang menyebabkan tidak berfungsinya perangkat lunak dan kadang kala yang berakibat perangkat lunak berprilaku anomali. *Defect* tersebut dapat berupa *error*, *bug*, *failure* atau *fault*  yang dapat menghasilkan *output* yang tidak akurat atau tidak sesuai dengan SKPL[1]. *Defect* tersebut dapat menyebabkan masalah besar baik pada tahap pengembangan (*development)* atau pada tahap perawatan (*maintenance)*.

Tim pengembang perangkat lunak pasti selalu menginginkan untuk menghasilkan perangkat lunak yang berkualitas dengan nol atau sedikit cacat. Usaha yang banyak dilakukan ialah dengan mengidentifikasi *bug* pada tahap pengujian, dan tim pengembang lebih berfokus pada dimensi fungsionalitas pada level *method* atau *class* yang membangun perangkat lunak tersebut, hal ini dikarenakan umumnya kegagalan ditimbulkan oleh kesalahan dalam kode program yang menimbulkan kegagalan ketika kode program yang salah tersebut dijalankan.

Namun faktanya kualitas perangkat lunak tidak dapat dicapai hanya dengan mengidentifikasi *bug* atau kesalahan pada tahap pengujian. *Defect* secara otomatis menginfeksi *source code* pada setiap fase siklus hidup pengembangan perangkat lunak sehingga pendeteksian *defect* harus dilakukan pada fase yang tepat. Hal ini menyebabkan banyak perangkat lunak yang dikirim ke pengguna yang mengandung *defect* karena penentu kualitas perangkat lunak hanya sampai pada tahap mengindentifikasi  *bug* atau *error* pada tahap pengujian (*testing*) sedangkan faktanya jenis *defect* pada perangkat lunak terdiri dari *requirment* *defect, design defect* dan *code* *defe*ct [2]*.*

Menemukan *defect* atau mengindentifikasi bagian – bagian rawan kesalahan (komponen yang beresiko tinggi) dari perangkat lunak sedini mungkin sangat bermanfaat dalam proyek perangkat lunak untuk meningkatkan kualitas produk perangkat lunak yang akan dihasilkan karena *defect* selalu meningkatkan biaya dan waktu dalam menyelesaikan produk perangkat lunak dengan kualitas yang diharapkan. Selain itu, mengidentifikasi dan memperbaiki *defect* adalah salah satu proses yang paling memakan waktu dan mahal. Faktanya tidak mudah untuk menghilangkan setiap defect tetapi mengurangi besarnya *defect* dan efek buruknya pada proyek dapat dicapai. Oleh sebab itu proses memprediksi *defect* perangkat lunak sangat memiliki peranan yang besar untuk membangun perangkat lunak yang berkualitas baik dengan cara yang tepat dan efisien [3].

Dengan prediksi yang efektif dalam memprediksi perangkat lunak yang rentan terhadap perubahan dapat menghasilkan penuruan biaya (berkurang secara eksponensial) perawatankarena pengembang perangkat lunak dapat melakukan berbagai tindakan perbaikan yang akan mengurangi kemungkinan timbulnya *defect* jika adanya perubahan pada fase pengembangan perangkat lunak selanjutnya.

Prediksi defect perangkat lunak adalah proses untuk menemukan sejumlah pola defect perangkat lunak yang dapat digunakan untuk memprediksi kemunculan defect perangkat lunak di masa mendatang. Pola defect perangkat lunak ini adalah sekumpulan attribut yang dapat digunakan untuk mendeskripsikan dan memprediksikan kemunculan dari defect perangkat lunak. Pola defect perangkat lunak dapat diturunkan dengan menerapkan model statistik pada perangkat lunak yang mengandung defect. Attribut – attribut dari perangkat lunak seperti ukuran *Line Of Code (*LOC), kompleksitas, dan sejarah perubahandibandingkan dengan pola defect perangkat lunak untuk mengukur produk perangkat lunak [3].

Untuk menghasilkan perangkat lunak berkualitas tinggi, produk akhir harus memiliki *defect* sedikit mungkin sedangkan proses penemuan *defect* pada perangkat lunak sering kali memakan waktu yang sangat panjang sehingga berpengaruh terhadap biaya yang dikeluarkan pihak pengembang [3]. Jadi, studi tentang prediksi *defect* penting untuk meningkatkan kualitas perangkat lunak. *Defect*  yang tidak biasa sangat sulit ditemukan pada saat proses pengujian, ada 3 jenis *defect* yang tidak biasa yang membatasi suatu perangkat lunak untuk bekerja sesuai dengan spesifikasinya.

1. Cacat Laten (*Laten Defect*)

Cacat tersembunyi dalam perangkat lunak yang tidak diidentifikasi oleh pengguna (meskipun pengembang / pemilik menyadarinya) sampai serangkaian operasi tidak dilakukan. Cacat ini diidentifikasi hanya ketika perangkat lunak diharapkan untuk melakukan tugas tertentu tanpa adanya skenario biasa atau terpapar pada keadaan yang tidak biasa. Ini merusak perangkat lunak biasa selama proses produksi, dan juga lolos ke pengujian pra-produksi.

2. Cacat Bertopeng ( *Masked Defect )*

Cacat Bertopeng adalah cacat yang sudah ada dalam eksekusi perangkat lunak terutama karena tertutup atau ditutupi oleh cacat lain. *Masked defect* sebenarnya ditemukan atau operasi spesifik dilakukan.

3. Cacat Cascading

Dalam pengujian perangkat lunak, cacat cascading berarti memicu cacat lain dalam aplikasi. Ketika cacat tidak teridentifikasi atau tidak terdeteksi saat pengujian, itu akan menimbulkan cacat lainnya. Akibatnya, beberapa cacat muncul di tahap selanjutnya. Dengan kata lain, itu adalah cacat utama yang memperkenalkan banyak cacat terkait dengannya pada tahap lebih lanjut dari siklus hidup aplikasi.

Hal ini memungkinkan cascading cacat terus mempengaruhi fungsi-fungsi lain dalam perangkat lunak dan mengidentifikasi fungsi yang terkena dampaknya bukan hal yang mudah untuk dilakukan jika tidak ada sebuah analisis awal yang digunakan sebagai acuan untuk mengidentifikasi fungsi fungsi yang tergolong rawan cacat dan mengandung cacat.

Banyak model yang dapat digunakan untuk memprediksi *defect* perangkat lunak salah satunya adalah model dengan penambangan data kegagalan perangkat lunak di masa lalu melalui repositori yang digunakan selama pengembangan nya. Dengan seiringnya bertumbuhnya komunitas dan popularitas pengguna *open source software (OSS)* telah membuat sejumlah besar data yang dimanfaatkan untuk tujuan tersebut.

Dalam beberapa tahun terakhir, para peneliti telah menggunakan sejumlah besar data yang terkandung oleh repositori perangkat lunak untuk memprediksi lokasi cacat dalam kode yang menyebabkan masalah. Fakta – fakta empiris yang dikumpulkan melalui analisis data yang dikumpulkan dari repositori perangkat lunak dapat membantu periset perangkat lunak untuk membangun teori yang terbentuk dengan baik dan umum. Data yang diperoleh dari repositori perangkat lunak dapat digunakan untuk menjawab sejumlah pertanyaan seperti :

1. Apakah desain A lebih baik dari desain B ?
2. Apakah proses atau metode A lebih baik dari metode B ?
3. Berapakah probabilitas terjadinya cacat pada modul ?
4. Berapakah waktu yang dibutuhkan untuk memperbaiki sebuah Bug ?

Pada penelitian ini berfokus untuk memprediksi besarnya kemungkinan cacat pada modul perangkat lunak dengan harapan pengembang dapat memperbaiki cacat yang ada pada perangkat lunak pada phase *maintenance* dengan tepat karena lokasi cacat pada kode sumber dapat diprediksi berdasarkan pengalaman dari masa lalu dengan memanfaatkan informasi *bug* yang digunakan selama proyek pengembangan perangkat lunak.

Karena dengan mengetahui modul mana yang kemungkinan mengandung cacat terlebih dahulu dapat membantu dalam mengarahkan langkah-langkah jaminan kualitas perangkat lunak seperti inspeksi dan pengujian ke modul-modul yang dimungkinkan memiliki cacat. Dengan prediksi cacat, modul rawan-rawan dalam versi sistem perangkat lunak yang akan datang dapat diprediksi dari data tentang versi sebelumnya. Jadi, prediksi cacat adalah bantuan yang menjanjikan untuk meningkatkan keefektifan dan efisiensi dari pengukuran jaminan kualitas yang mahal ini [1].

Metrik produk statis adalah salah satu kategori metrik yang telah sering digunakan sebagai prediktor cacat (mis., [2], [3], [4]). Contohnya adalah metrik yang terkait dengan ukuran, kompleksitas, desain berorientasi objek, dan struktur sistem. Kategori lain mencakup metrik proses yang mencirikan berbagai aspek dan kegiatan pengembangan perangkat lunak (misalnya, [5] [6], [7]). Contohnya adalah metrik yang terkait dengan riwayat perubahan, churn kode, atau resolusi cacat. Akhirnya, sumber daya metrik seperti metrik tentang pengembangan personil baru-baru ini telah terlibat dalam studi empiris (misalnya, [8], [9], [10]). Metrik tersebut telah diturunkan dari jaringan sosial pengembang atau kontribusinya ke sistem perangkat lunak.

Dengan mengumpulkan informasi mengenai *bug* dari sistem pelacakan *bug* dan menganalisis metrik perangkat lunak dengan memanfaatkan repositori pengelolaan kode sumber dengan tepat, diharapakan dapat menghasilkan model prediksi cacat perangkat lunak yang lebih baik secara fungsional dan memiliki kinerja yang lebih baik dari penelitian – penelitian terdahulu.

*The Mining Software Repositories (MSR)* adalah bidang penelitian dalam menganalisis data yang tersedia direpositori perangkat lunak menggunakan teknik penambangan data. Data pada repositori pengembangan perangkat lunak bersifat kaya dan menarik untuk digali informasi yang dapat ditindaklanjuti untuk berbagai hal dalam siklus hidup perangkat lunak.

Dalam menganalisa data tersebut banyak manfaat pada area penelitian pengembangan rekayasa perangkat lunak untuk memberikan solusi alternatif atau baru untuk mengingkatkan keefektifan dan keakurasian dari sebuah penelitian seperti *defect prediction, pattern mining, change impact analysist, design correction, change prediction, effort estimation, trend analysis* dan lainnya.

Proses penambangan informasi bertujuan untuk mengekstraksi informasi dari *event log* yang dikoleksi oleh *Software Configuration Management* (SCM), hal ini bertujuan untuk menangkap proses siklus hidup perangkat lunak selama proses evolusiperangkat lunak tersebut.

Seiring dengan berevolusinya perangkat lunak yang dikembangakan, perubahan – perubahan yang dilakukan pengembang terekam dalam repositori yang digunakan selama pengembangan. Informasi yang yang terkait tersebut antara lain sebagai berikut :

1. Nama berkas yang mengalami perubahan
2. Tanggal dan jam perubahan dilakukan
3. Identitas Pengembang yang melakukan perubahan
4. Penjelasan Terkait perubahan yang dilakukan pengembang.

Selain itu repositori juga menyimpan informasi seperti kode sumber, arsip email, dan database bug , berisi teks tidak terstruktur dan berlabel. Berikut ini adalah parameter umum dalam mendefinisikan *bugs* di repository.

1. Kategori : parameter yang melabeli jenis kecacatan. Contohnya *software engineering process (SEP*), *hardware, code, product system, operation.*
2. Priority : parameter yang melabeli tingkat kepentingan dalam menangani atau mengatasi cacat. Pritoritas dikategorikan kedalam angka 1 – 3. Contohnya 1 – *Mission Critical, 2-Mission Essential (ok with work-around), 3-Cosmetic).*
3. *Severity,* besaran atau tingkatan kecacatan yang mempengaruhi penggunaan perangkat lunak. Besaran keras kecacatan ini akan mempengaruhi prioritas pekerjaan perbaikan.
4. *Status,*  status mengenai cacat yang ditemukan, status dapat berupa open, closed, reopened, started, assigned, uncofirmed, confirmed, dll).

## I.2 Rumusan Masalah

Berdasarkan latar belakang yang telah diuraikan pada bagian sebelumnya, dapat di identifikasi sebuah permasalahan utama yaitu :

1. Bagaimana mengklasifikasikan kecacatan perangkat lunak menjadi rawan cacat dan tidak rawan cacat.
2. Bagaimana menemukan asosiasi kecacatan yang ada pada perangkat lunak.

Dari masalah utama tersebut diuraikan 4 masalah turunan yaitu :

1. Seberapa besar ketepatan metode Cost Sensitive Neural Network (NN) untuk mengklasifikasi kecacatan perangkat lunak menjadi rawan cacat dan tidak rawan cacat dibandingkan dengan metode konvensional.
2. Bagaimana Metric Branch Count (*MBC)* memprediksi asoiasi kecacatan yang ada pada perangkat lunak?

## I.3 Hipotesis Awal

Dari latar belakang yang telah diuraikan sebelumnya dan rumusan masalah yang telah dijabarkan, penulis membuat sebuah hiptesis awal yaitu :

Model prediksi kecacatan memanfaatkan penambangan repositori dengan menggali informasi yang berasal dari log atau perubahan kode program, *bug/issue* akan menghasilkan model pengujian pada titik yang optimum.

## I.3 Tujuan Penelitian

Dari hipotesis yang telah dijabarkan maka didefinisikan tujuan dilakukannya penelitian ini yaitu sebagai berikut :

1. Membangun model prediksi kecacatan berbasis penambangan perangkat lunak repositori, hal ini memanfaatkan kualitas proses pengembangan perangkat lunak itu sendiri.
2. Mengembangkan model yang berakurasi tinggi dalam mengklasifikasi kecacatan perangkat lunak menjadi rawan cacat dan tidak rawan cacat menggunakan metode Cost Sensitive Neural Network (NN).
3. Menguji metode Cost Sensitive Neural Network (NN) dan Metric Branch Count (*MBC)* dalam prediksi kecacatan perangkat lunak.

## I.4 Batasan Masalah

Batasan dari penelitian ini adalah :

1. Penelitian terbatas kepada proyek *open source* yang menggunakan bahasa pemograman java yang dibangun mengunakan standar konfigurasi management proyek .
2. Repositori yang menjadi bahan pengujian ialah repositori *software open source*  seperti BugZilla dan Eclipse.

## I.5 Metodologi Penelitian

Berdasarkan yang digunakan untuk mengerjakan penelitian ini adalah sebagai berikut :

1. Studi Literatur, mempelajari literatur – literatur yang terkait dengan konsep model kecacatan pada perangkat lunak, metrik perangkat lunak,data mining, dan *meachine learning.*
2. Analisis Masalah, melakukan analisis metrik proses yang diperlukan untuk membangun model prediksi kecatatan perangkat lunak.
3. Pengembangan model sistem prediksi kecacatan perangkat lunak. Pada tahap ini, dilakukan implementasi pembentukan metrik, serta implementasi algortima *machine learning* untuk memprediksi kecacatan pada model klasifikasi dan prediksi jumlah kecacatan perangkat lunak.
4. Pengujian model dengan mengembangkan perangkat lunak predikisi kecacatan perangkat lunak.

# BAB II STUDI LITERATUR

### II.1 Cacat Perangkat Lunak (*Software Defect*)

Cacat perangkat lunak (software defect) didefinisikan sebagai cacat pada perangkat lunak seperti cacat pada dokumentasi, pada kode program, pada desain dan hal – hal lain yang menyebabkan kegagalan perangkat lunak. Cacat perangkat lunak dapat muncul pada berbagai tahap proses pengembangan perangkat lunak (Pressman, 2001). Cacat perangkat lunak merupakan faktor penting yang mempengaruhi kualitas perangkat lunak.

Kualitas perangkat lunak dapat ditingkatkan dengan mencegah munculnya cacat perangkat lunak melalui perbaikan aksi yang mungkin menghasilkan cacat perangkat lunak pada proses pengembangan perangkat lunak (Boehm dan Basili, 2001).

Teknik pencegahan cacat perangkat lunak pertama kali diusulkan oleh IBM corporation dan dapat digunakan untuk mencegah munculnya cacat perangkat lunak di tahap lanjut pada proses pengembangan perangkat lunak (Chang,2007). Setelah itu, muncul beberapa pendekatan untuk mencegah munculnya cacat perangkat lunak seperti causal analysis dan prediction model (Chang,2007).

### II.X Metrik Perangkat Lunak

Metrik perangkat lunak adalah ukuran kuantitatif sederhana yang dapat dipindahkan dari semua atribut siklus hidup perangkat lunak. Metrik perangkat lunak memungkinkan pengembang perangkat lunak untuk mengukur dan memprediksi proses perangkat lunak, sumber daya yang diperlukan untuk proyek dan produk kerja yang relevan untuk upaya pengembangan perangkat lunak.

Pada penelitian ini mengunakan 2 kategori metrik kualitas perangkat lunak. Yang pertama ialah metrik proses, terkait dengan proses pengembangan perangkat lunak dan metrik produk, terkait dengan pemeliharaan perangkat lunak.

*Lines of Code* (LOC), Kompleks Cyclomatic McCabe, dan Ilmu Pengetahuan Perangkat Halstead adalah atribut statis yang umum digunakan para pengembang untuk mengukur kompleksitas kode [3].

### II.X.X Metrik KLOC

### II.X.X Metrik Cyclomatic McCabe

## II.X Kecacatan Perangkat lunak

Kecacatan Perangkat Lunak adalah suatu kondisi perangkat lunak yang tidak memenuhi persyaratan perangkat lunak (sesuai spesifikasi kebutuhan), atau harapan pengguna. Pada umumnya kecacataan perangkat lunak merupakan kesalahan dalam pengkodean, logika, atau komputai yang menyebabkan suatu program mengalami kerusakan atau menghasilkan hasil yang salah atau tidak terduga.

Klasifikasi dari penyebab kecacatan perangkat lunak :

1. Kesalahan definisi dari kebutuhan perangkat lunak.
2. Kegagalan komunikasi antara client dan pengembang.
3. Penyimpangan yang disengaja dari kebutuhan perangkat lunak.
4. Kesalahan analisis dan design.
5. Kesalahan penulisan code program.
6. Ketidaksesuaian antara dokumentasi dengan implementasi penulisan kode.
7. Kekurangan dari proses pengujian.

Kecacatan perangkat lunak dilaporkan dalam dokumen pelaporan kecacatan atau *bug/issue* *tracking system*. Potensi cacat tertinggi terjadi pada tahap pengkodean sebesar 1,75% cacat per fungsi, kedua pada desain sebesar 1,25 % cacat per fungsi, ketiga pada persyaratan sebesar 1% cacat, ke-empat pada dokumentasi sebesar 0,6 % cacat per fungsi, dan pada kesalahan perbaikan sebesar 0,4 cacat per fungsi[1].

Berikut ini adalah Klasifikasi kecacatan perangkat lunak yang umum digunakan pada pelaporan kecacatan perangkat lunak:

1. *Defect Severity*, menunjukan tingkat dampak kecacatan terhadap kualitas perangkat lunak. Biasanya dampak kecacatan perangkat lunak di klasifikasikan kedalam tingkatan S1 (*Critical*) , S2 (*Major*), S3 (*Minor*), dan S4 (*Trivial*).
2. *Defect Probabilty*, menunjukan kemungkinan pengguna menghadapi kemungkinan cacat pada fitur yang ada pada perangkat lunak, kemungkinan cacat di klasifikasikan ke *high, medium ,low.*
3. *Defect Priority,* menunjukan pentingnya atau urgensinya sebuah kondisi kecacatan harus diperbaiki. Tingkat proritas dapat dikategorikan kedalam tingkat *urgent, medium, low dan low.*
4. Klasfikasi berdasarkan fase terdeteksi sebuah cacat, menunjukan fase dalam siklus perangkat lunak dimana cacat terindentifikasi, mungki pada fase *unit testing, integration testing, system testing* atau *acceptance testing.*

Berikut penelitian – penelitian mengenai model prediksi cacat perangkat lunak :

### II.X.1 Prediksi menggunakan ukuran dan kompleksitas metrik

Sebagian besar penelitian prediksi cacat perangkat lunak didasarkan ukuran dan kompleksitas metrik. Studi seperti ini telah lama dilakukan oleh Akiyama[2] yang didasarkan pada sistem yang dikembangkan di Fujitsu, Jepang. Ini adalah tipikal dari banyak model 'data fitting' berbasis regresi. Studi ini menunjukkan bahwa model linier dari beberapa metrik sederhana memberikan perkiraan yang masuk akal untuk jumlah cacat D (variabel dependen) yang sebenarnya didefinisikan sebagai jumlah cacat yang ditemukan selama pengujian dan cacat yang ditemukan selama dua bulan setelah pelepasan. Akiyama menghitung empat persamaan regresi. Contoh : Persamaan (1) yang melibatkan baris kode L (LOC) adalah sedemikian rupa sehingga, misalnya, modul 1000 LOC (mis. 1 KLOC) diperkirakan memiliki sekitar 23 cacat:

*D =* 4*.*86 *+* 0*.*018 *L*

Studi lainnya adalah Ferdinand[3], mengemukakan bahwa jumlah cacat yang diharapkan meningkat dengan jumlah n dari segmen kode; Segmen kode adalah urutan pernyataan eksekusi yang sekali masuk, semua harus dieksekusi. Secara khusus, teori tersebut menegaskan bahwa untuk segmen yang lebih kecil, jumlah cacat sebanding dengan kekuatan n; untuk jumlah segmen yang lebih banyak, jumlah kerusakan meningkat sebagai konstan pada daya n.

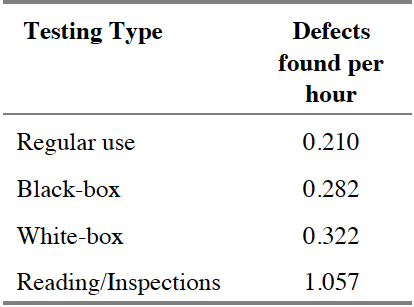
Halstead[10] juga mengajukan sejumlah metrik ukuran, yang telah ditafsirkan sebagai metrik 'kompleksitas', dan menggunakan ini sebagai prediktor defek program. Yang paling menonjol, Halstead menegaskan bahwa jumlah cacat D dalam program P diperkirakan oleh (2):

dimana V adalah metrik volume (bahasa dependent) (yang seperti semua metrik Halstead didefinisikan dalam jumlah operator unik dan operan unik di P. Pembagi 3000 mewakili jumlah rata-rata diskriminasi mental antara keputusan yang dibuat oleh pemrogram. Setiap keputusan tersebut mungkin menghasilkan kesalahan dan dengan demikian merupakan cacat residual.

### II.X.2 Prediksi menggunakan metrik pengujian

Metrik pengujian memprediksi cacat dengan ukuran cakupan testing pada setiap tahapan. Strategi pengujian struktural menentukan bahwa pengembang harus memilih uji kasus yang cukup sehingga masing-masing seperangkat "objek" dalam sebuah program terletak pada beberapa kasus uji. Idenya adalah dengan memiliki n fase yang telah ditentukan di mana pengembang mengumpulkan data dn tingkat cacat.

Prediksi menggunakan metrik pengujian adalah prediksi kecacatan perangkat lunak melibatkan pengumpulan data yang sangat hati-hati tentang cacat yang ditemukan selama tahap awal pengujian hingga akhir pengujian. Misalkan fasa n mewakili periode 6 bulan pertama produk di lapangan, sehingga dn adalah tingkat cacat yang ditemukan dalam periode tersebut. Untuk memprediksi dn pada fase n-1 (yang mungkin merupakan pengujian integrasi), makan akan terlihat urutan sebenarnya d1, d2, d3, ..., dn-1 dan bandingkan ini dengan profil produk serupa sebelumnya, dan gunakan teknik ekstrapolasi statistik. Dengan data yang cukup, mungkin untuk mendapatkan prediksi yang akurat tentang dn berdasarkan pengamatan d1, ..., dm dimana m kurang dari n-1. Ini diklaim sebagai prediksi yang sangat akurat dengan batas kepercayaan 95% dengan syarat kestabilitasan pembangunan menguji setiap phase dan tingkat pengumulan data yang tinggi, berikut padata tabel xcontoh data pengujian disetiap tahapan pengembangan yang dipublikasikan oleh Fenton tahun 1994.



Tabel . Pendeteksian Cacat Menggunakan Metrik Pengujian

### II.X.3 Prediksi menggunakan data kualitas proses pengembangan

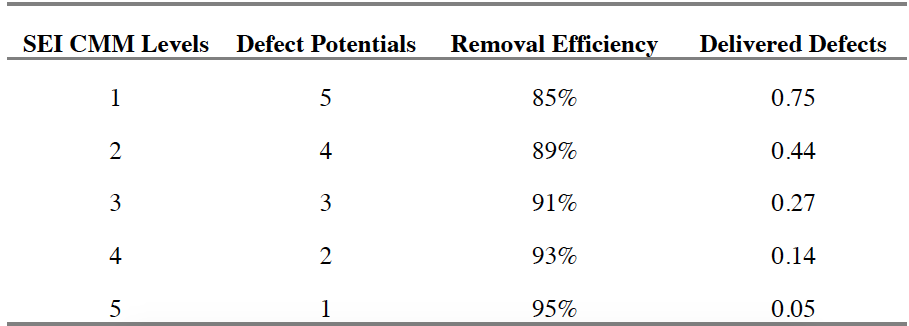
Banyak ahli berpendapat bahwa kualitas proses pengembangan adalah prediktor terbaik kecacatan terbaik karena secara jika kualitas proses pengembangan berjalan dengan baik secara otomatis menyisakan sedikit kecacatan perangkat lunak[4]. Namun menghubungkan kualitas proses dengan kualitas produk bukanlah hal yang gampang, perlu pembuktian empiris.

Metrik kualitas proses yang paling sederhana adalah *SEI* *Capability Maturity Model* (CMM) hal ini diusulkan oleh Diaz dan sligo pada tahun 1997[5]. Meskipun popularitasnya meluas, sampai saat ini tidak ada bukti yang menunjukkan bahwa perusahaan dengan tingkat (n + 1) umumnya menghasilkan produk dengan sisa cacat yang lebih rendah daripada pada tingkat (n).

Jelas peringkat 1-5 yang ketat, seperti yang ditentukan oleh SEI-CMM, terlalu kasar untuk digunakan secara langsung untuk prediksi cacat karena tidak semua proses yang tercakup dalam CMM akan berhubungan dengan kualitas perangkat lunak.

Bukti terbaik yang ada yang berkaitan dengan metode proses adalah mengurangi kepadatan menyangkut metode Cleanroom diusukan oleh Dyer 1992[6]. Ada validasi independen bahwa, untuk proyek yang relatif kecil (kurang dari 30 KLOC), penggunaan Cleanroom menghasilkan sekitar 3 kesalahan per KLOC selama pengujian statistik, dibandingkan dengan kepadatan cacat pasca-*­­release* antara 5 sampai 10 cacat per KLOC.

Jones, C.,[7] menghipotesiskan target kualitas yang dinyatakan dalam 'potensi cacat' dan 'cacat yang disampaikan' untuk tingkat CMM yang berbeda, seperti ditunjukkan pada Tabel X.



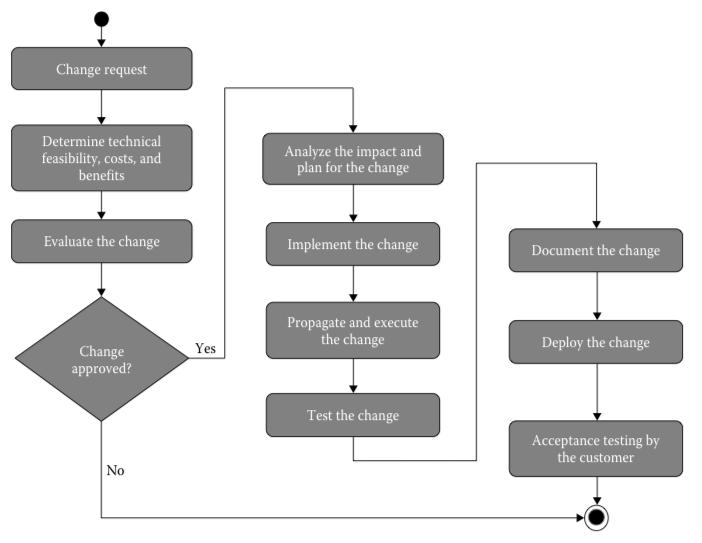
Tabel . Hubungan Level CMM dengan *Delivered Defect* [14]

## II.X Sistem Manajemen Konfigurasi

Sistem manajemen konfigurasi (SCM) sangat penting bagi hampir semua proyek perangkat lunak yang dikembangkan oleh organisasi. Tujuan dari sistem manajemen konfigurasi adalah untuk mengendalikan dan mengelola perubahan yang terjadi pada semua artefak yang dihasilkan selama siklus pengembangan perangkat lunak. Artifak (juga dikenal sebagai kiriman) yang diproduksi selama siklus pengembangan perangkat lunak mencakup spesifikasi persyaratan perangkat lunak, dokumen perancangan perangkat lunak, daftar kode sumber, manual pengguna, dan sebagainya (Bersoff et al 1980; Babich 1986).

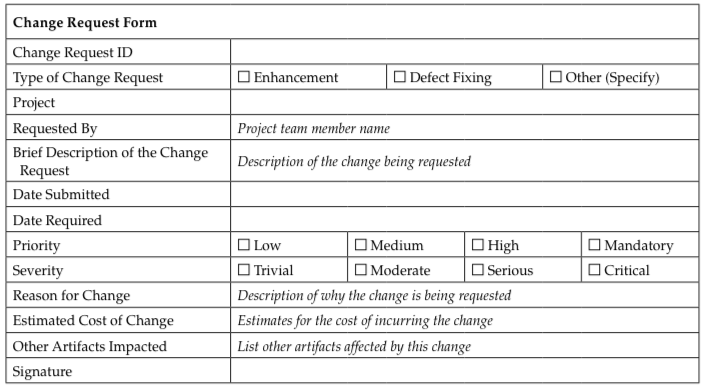
Setiap artefak yang berkaitan dengan proyek perangkat lunak yang diproduksi selama siklus pengembangan perangkat tersimpan di version kontrol sistem (VCS). Setiap aktivitas dapat dilacak dalam repositori perangkat lunak yang berada pada version sistem kontrol (VCS).

Hal ini yang dimanfatkan oleh CCB selaku personal yang memiliki otoritas dan bertanggung jawab atas persetujuan dan pelacakan perubahan karena setiap adanya permintaan sebuah perubahan (*change request*) maka CCB akan dengan hati – hati dan cermat meninjau setiap perubahan sebelum di setujui. Ini adalah sebuah proses yang kritis dan biasanya disebut sebagai kontrol konfigurasi. Proses ini menggabungkan persetujuan, kontrol, dan implementasi perubahan pada artefak proyek perangkat lunak, atau ke proyek perangkat lunak itu sendiri. Tujuan utamanya adalah untuk memastikan bahwa setiap perubahan yang terjadi pada artefak perangkat lunak dilakukan dengan pengetahuan dan persetujuan tim manajemen proyek perangkat lunak. Tahapan proses perubahan ini dapat dilihat pada Gambar X berikut :



**Gambar 2**. Siklus Perubahan (Change Cyle )[8]

Untuk mengajukan sebuah change request biasanya stakeholder atau orang yang terkait mengenai permintaan perubahan tersebut mengisi sebuah formulir sebuah perubahan yang biasa di sebut dengan *change request form. Gambar X.* berikut adalah contoh formulir perubahan yang banyak digunakan pada proses pengembangan perangkat lunak.



## II.X Repositori Perangkat Lunak

Repositori perangkat lunak adalah lokasi penyimpanan yang dikelola secara online atau offline oleh beberapa organisasi pengembangan perangkat lunak. Repositori ini adalah tempat segala artefak yang berhubungan dengan perangkat lunak yang sedang dikembangkan seperti kode sumber, bug, dan banyak informasi lainnya. Banyak terdapat repositori yang sifat nya open source.

Repositori perangkat lunak menyimpan rekaman perubahan dari suatu proyek perangkat lunak. Repositori tersebut dapat berupa sistem kontrol versi (VCS) kode, sistem penelusuran bug seperti BugZilla, atau arsip komunikasi antar pengembang seperti mailing list atau slack. Sistem kontrol kode sumber menyimpan rekaman perubahan kode sumber program dari suatu proyek perangkat lunak. Beberapa contoh sistem kontrol kode sumber adalah Git, Atlassian Bitbucket dan SourceForge. Berikut ini beberapa informasi yang dapat diidentifikasi pada setiap rekaman perubahan pada sistem kontrol kode sumber:

1. Nomor kode perubahan,
2. Pengembang yang melakukan perubahan,
3. Tanggal perubahan,
4. Deskripsi perubahan, dan
5. Berkas-berkas yang mengalami perubahan.

Pada peneltian yang dilakukan Michael Fischer dan rekan-rekannya, dijelaskan sebuah metode untuk membentuk basis data riwayat rilis perangkat lunak. Basis data tersebut dibentuk dengan mengkombinasikan data riwayat pada repositori kode sumber dengan data riwayat pada sistem penelusuran bug [1].

Pada penelitian ini, repositori perangkat lunak yang digunakan sebagai obyek penelitian adalah sistem kontrol kode sumber. Sisi ketersediaan dari data riwayat perubahan perangkat lunak pada repositori online, seperti Git, Attlassian Bitbucket dan SourceForge menjadi pertimbangan utama dipilihnya sistem kontrol kode sumber sebagai obyek penelitian. Dengan menggunakan web crawler , data riwayat tersebut akan disimpan ke dalam basis data relasional lokal.

VCS adalah salah satu bagian dari SCM. Fokus utama pada VCS adalah menelusuri perubahan yang tejadi ketika kode sumber diubah dari satu versi ke versi lainya (Brown, 2002). VCS digunakan untuk pengembangan perangkat lunak secara kolaboratif (dikerjakan oleh beberapa orang dalam satu tim atau dikerjakan oleh beberapa tim) karena setiap kode sumber yang baru dan diubah akan terekam perubahanya sehingga orang lain dapat mempelajari kodes sumber melalui perbuahan yang terjadi.

Aktifitas yang dilakukan oleh pengembang dalam kegiatan versioning adalah aktivitas dasar, pegelolaan fitur, dan kerja kolaborasi. Aktifitas dasar di awali dengan pembuatan repositori, pendaftaran perubahan, commit perubahan, dan status perubahan. Alur kerja dimulai dari membentuk repositori lokal, baik dengan membuat proyek baru atau meneruskan proyek lama. Ketika repositori sudah terbentuk, pengembang memeriksa status repositori seperti status *workspace,* status perubahan terakhir, dan status perubahan yang dilakukan. Selanjutnya pengembang mendaftarkan berkas – berkas yang akan direkam perubahannya yang nantinya akan menjadi *log versioning.*

Berkas – berkas yang didaftarkan di-*commit* agar sejarah perubahan yang dihasilkan dari setiap *commit* yang dilakukan oleh pengembang. Data yang direkan seperti waktu *commit*, *author*, *commit message* dan status *merge.*

Tabel X adalah daftar repositori rekayasa perangkat lunak yang dapat digunakan para peneliti untuk mengembangkan model prediksi perangkat lunak. kumpulan data pada repositori ini bersifat terbuka dan dapat digunakan peneliti – peneliti untuk berbagai tujuan.

Tabel 3. Daftar Repositori Perangkat Lunak

| **Repository** | **Web Link** | **Source** | **Data Format** | **Sources** | **Public** |
| --- | --- | --- | --- | --- | --- |
| FLOSSmole | http:// ossmole.org | OSS | DB dumps, text, DB access | Multiple | Yes |
| FLOSSMetrics | http:// ossmetrics.org | OSS | DB dumps, web service, web | Multiple | Yes |
| PROMISE | http://promisedata.org | Mostly  proprietary | Mostly ARFF | Multiple | Yes |
| Qualitas Corpus | http://qualitascorpus. com | OSS | CSV, source code, JAR | Multiple | Yes |
| Sourcerer project | http://sourcerer.ics.uci. | OSS | DB dumps | Multiple | Yes |
| Bug prediction data set | http://bug.inf.usi.ch | OSS | CSV | Multiple | Yes |
| Eclipse bug data | http://www.st.cs. | OSS | ARFF, CSV | Single (Eclipse) | Yes |
| Helix data set | http://www.ict.swin. edu.au/research/projects/helix | OSS | CSV | Multiple | Yes |

Pada penelitian ini, repositori rekayasa perangkat lunak yang digunakan untuk pengujian model prediksi yang di usulkan ada 3 buah repositori, yaitu promise, bug data prediction data set dan eclipse bug data.

## II.X Pelaporan Cacat Pada SCM

Berikut ini adalah jenis – jenis kesalahan yang umumnya dilaporkan pada laporan Bug :

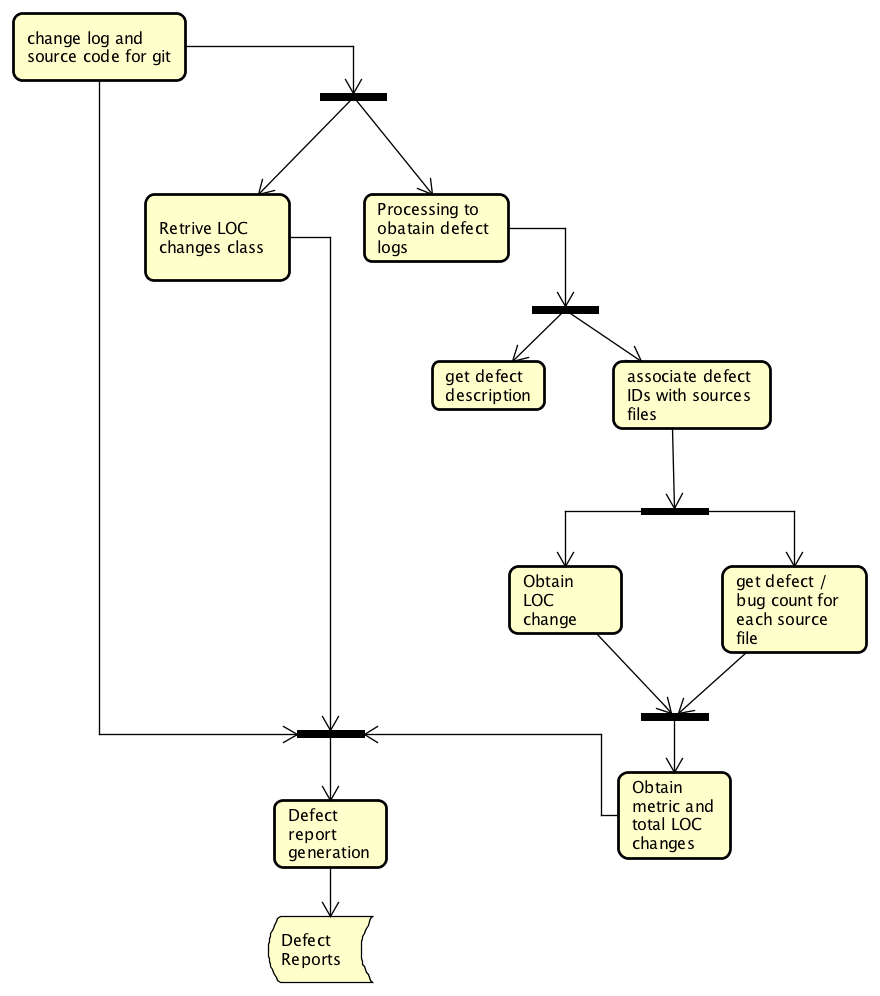
1. Coding Error : Program tidak melakukan apa yang dimaksud oleh penulis program
2. Isu Desain : Prilaku dari perangkat lunak membingungkan programmer dalam mengimplemtasikan kedalam bentuk code program.
3. Isu *Requirment* : Prilaku yang diharapkan mudah dimengerti namun gagal memenuhi persyaratan yang ditentukan.
4. Dokumentasi dan Kode Program tidak sesuai.
5. Spesifikasi dan kode program tidak sesuai.

Pelaporan *defect* memberikan informasi yang berguna tentang setiap *defect* seperti ID *defect*, deskripsi, dan kode sumber yang telah diubah untuk menghilangkan cacat tersebut.Laporan *defect* juga menyediakan file sumber perubahan LOC, sesuai dengan cacat yang diperbaiki dengan memodifikasi file tertentu.

infromasi yang ada dalam laporan *defect*, sesuai dengan masing-masing cacat, adalah sebagai berikut:

* kode sumber X(dapat beris nama file atau nama kelas)
* ID Defect X
* Deskripsi cacat (jika ada) X
* LOC sewaktu defect X
* LOC setelah perbaikan defect X
* Perubahan LOC total terhadap X cacat

Gambar X adalah flowchart yang digunakan untuk menyajikan laporan defect pada DCRS System.



Gambar 3. Flowchart untuk *Defect Reports*

## II.X Mining Software Repositories (MSR)

Penambangan repositori perangkat lunak (*MSR*) adalah bidang penelitian yang berfokus pada penggalian informasi – informasi yang sangat berharga dari repositori yang digunakan selama pengembangan perangkat lunak . Para peneliti telah mulai menambang repositori untuk mendapatkan pemahaman yang lebih baik dari artefak yang terus berubah yang terkait dengan proyek-proyek pengembangan perangkat lunak jangka panjang. Proses *MSR* mengekstraksi data dari berbagai repositori seperti sistem kontrol *source code*, sistem pelacakan *bug,* dan arsip komunikasi selama phase pengembangan untuk mengambil fakta tersembunyi dengan tujuan yang berbeda.

Thomas Zimmermann mengatakan bahwa *“Learning from past suc- cesses and failures helps us create better software”*. Belajar dari keberhasilan dan kegagalan di masa lalu membantu pengembang membuat perangkat lunak yang lebih baik. Hal ini dapat menggambarkan salah satu tujuan akhir dari *MSR* [5]. Penerapan *MSR* meliputi area penelitian seperti *predicting defect* *, Co-evolution of production and test code*, *impact analisis* , *effort prediction*, *similitry prediction* dan banyak lagi[d]. Semuanya itu bertujuan untuk meningkatkan kualitas proses perangkat lunak dan tujuan akhirnya untuk menghasilkan perangkat lunak yang berkualitas sesuai dengan yang diharapkan.

Alat dan teknik penambangan *MSR* semakin popularitas di lingkungan rekayasa perangkat lunak sejak 2013 karena hasilnya yang baik. Tabel X adalah daftar beberapa perangkat lunak yang digunakan untuk menganlisis *source code* untuk mengetahui kualitas dari sejarah pengembangan perangkat lunak di masa lalu.

Tabel X. Perangkat Lunak Analisis Kode Sumber

| **Tool** | **Avaibility** | **Source Languange** | **Languange(s) Supported** | **Metrics Provided** |
| --- | --- | --- | --- | --- |
| Understand | Propriety (Berbayar) | IDE | C, C++, Java , Ada, C# | Complexity, size, volume, and a few other OO metrics |
| CKJM calculator Source monitor | Open source Proprietary (freeware) | Java  C++ | Java byte code C, C++, C#,  .NET, Java, etc. .NET languages | OO design metrics Code metrics. |
| Code analyzer | Open source | Java | C, C++, Java,  Assembly,  HTML | Code metrics and allows user de ned metrics |
| SonarQube | Open source | Java | Java, C, C++,  PHP, Cobol, etc. | OO design, documentation size and test metrics |
| phpUnderControl | Open source | PHP, Java | PHP | Code, coverage, and test metrics |

Pada penelitian ini *tool* yang digunakan untuk menganalisis kode adalah Sonarqube. SonarQube adalah aplikasi *open source* dan sangat populer digunakan para pengembang perangkat lunak untuk mengalisis code dari perangkat lunak yang dikembangkan. SonarQube adalah aplikasi open source yang ditulis dalam bahasa pemrograman Java (www. Sonarqube.org). Sebagian besar digunakan untuk aplikasi berbasis Java, namun juga mendukung berbagai bahasa lain seperti C, C ++, PHP, COBOL, dan banyak lagi.

SonarQube menawarkan kemampuan untuk menambahkan peraturan kita sendiri ke bahasa-bahasa ini. SonarQube menyediakan berbagai metrik OO, termasuk metrik kompleksitas (kompleksitas kelas, kompleksitas metode, dll.), Metrik desain (RFC, siklus paket, dll.), Metrik dokumentasi, (baris komentar, komentar dalam divisi prosedur, dll. ), metrik duplikasi (duplikat baris), isu metrik (masalah total, masalah terbuka, dll.), metrik ukuran (LOC, jumlah kelas, dll.), dan metrik uji (*branch coverage*, *total coverage* , dll.).

# BAB III ANALISIS DAN PERANCANGAN

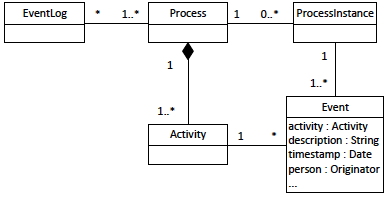
Bab ini gambaran sistem yang akan dibangun secara keseluruhan, terkait dengan kebutuhan dan rancangan arsitektur sistem defect prediction, dan skenario adaptasi.

## III.X Analisis Sistem

Dataset SDP berisi dua kategori informasi penting, pertama adalah metrik perangkat lunak seperti kompleksitas, line of code (loc) dan yang kedua adalah jumlah bug untuk setiap modul perangkat lunak (paling sering untuk setiap file) di masa lalu.

## III.X

Dari data yang didapat kemudian dihasilkan aturan pendeteksian berbentuk kombinasi dari matriks kualitas perangkat lunak.



Gambar 3. *Process Mining Meta Model.*

# 

# DAFTAR PUSTAKA

|  |  |
| --- | --- |
| [1] | Verma, Deepak and S. H. Shukla, "A Review on Software Defect Prediction.," *International Journal of Advanced Research in Computer Engineering & Technology.,* vol. 4., pp. 4387-4394, 2015. |
| [2] | P. Runeson, C. Andersson, T. Thelin, A. Andrews and T. Berling, "What Do We Know About Defect Detection Methods? [Software Testing].," *IEEE Software,* vol. 23, pp. 82-90, May-June 2006. |
| [3] | Chang and Chingpao-Pao, "Software process improvement using multivariate statistical process control and action based defect prediction,," *Departement of computer science and information engineering,,* 2008. |
| [4] | H. Zhang, X. Zhang and M. Gu, "Predicting Defective Software Components from Code Complexity Measures," pp. 93-96, 03 March 2007. |
| [5] | T. Zimmermann. [Online]. Available: http://thomas-zimmermann.com/research/ . [Accessed 12 Oct 2017]. |
| [6] | H. S. Shukla, " A Review on Software Defect Prediction," *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET),* vol. 4, no. 12, December 2015. |