Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer e-ISSN: 2548-964X
Vol. 3, No. 3, Maret 2019, hlm. 2167-2172 http://j-ptiik.ub.ac.id

Pembangunan Kakas Bantu Untuk Mengukur Maintainability Index Pada Perangkat Lunak Berdasarkan Nilai Halstead Metrics dan McCabe's Cyclomatic Complexity

Rasio Ganang Atmaja[1], Bayu Priyambadha[2], Fajar Pradana[3]

Program Studi Teknik Informatika, Fakultas Ilmu Komputer, Universitas Brawijaya
Email: [1]rasioganang29@student.ub.ac.id, [2]bayu_priyambadha@ub.ac.id, [3]fajar.p@ub.ac.id

Abstrak

Dalam siklus pengembangan perangkat lunak terdapat fase *maintenance*. Pada fase ini, kesalahan atau kecacatan perangkat lunak yang sebelumnya belum ditemukan pada face pengembangan atau pengujian akan diperbaiki. Pada tahap ini, perangkat lunak juga mengalami perubahan untuk menyesuaikan dengan lingkungan sistem yang baru dan kebutuhan *stakeholder*. Dalam pengembangan perangkat lunak ada beberapa alasan mengapa diperlukan perhitungan nilai maintainability suatu perangkat lunak diantaranya, nilai dari *maintainability* bisa membantu dalam memutuskan apakah suatu perangkat lunak mudah dirawat atau perlu dilakukan perancangan ulang. Ada beberapa cara yang bisa digunakan untuk mengukur nilai *Maintainability* perangkat lunak, salah satunya adalah *Maintainability Index* (MI). *Maintainability Index* dihitung berdasarkan nilai dari *Halstead's Volume, McCabe's Cyclomatic Complexity*, dan jumlah baris kode sumber. Sistem kalkulasi *Maintainability Index* ini menyediakan fitur untuk mengalkulasi nilai *Maintainability Index* dari kode sumber Java dan menampilkan visualisasi *graph* dengan menggunakan teknologi Java. Sistem ini telah diuji dengan menggunakan pengujian unit dan pengujian integrasi yang menggunakan metode *Whitebox* serta pengujian validasi yang menggunakan metode *Blackbox*. Sistem ini mempunyai tingkat akurasi sebesar 100% dan waktu untuk kalkulasi satu method hanya membutuhkan waktu kurang dari 1000ms.

Kata kunci: *perangkat lunak, maintenance, Maintainability Index, java, Halstead Metric, Cyclomatic Complexity*

Abstract

In the software development cycle there is a maintenance phase. In this phase, errors or defects of the software that have not been found on development or testing phase will be corrected. In this phase, software is also changing to fit the new system environment and stakeholder needs. In software development there are several reasons why it is necessary to calculate maintainability value of the software, such as the value of maintainability can help in deciding whether a software is easy to maintain or needs to be redesigned. There are several methods that can be used to measure maintainability value of the program, one is the Maintainability Index (MI). Maintainability Index is calculated based on the value of Halstead's Volume, McCabe's Cyclomatic Complexity, and line of codes. The Maintainability Index calculations system provide features for calculate Maintainability Index values of the Java source code and display graph visualizations using Java technologies. This system has been tested using unit testing, integration testing that uses Whitebox methods and validations testing that use Blackbox methods. This system has an accuracy of 98% and the time for calculating one method only takes less than 1000ms.

Keywords: *software, maintenance, Maintainability Index, java, Halstead Metric, Cyclomatic Complexity*

1. PENDAHULUAN

Dalam siklus pengembangan perangkat lunak terdapat fase *maintenance*. Pada fase ini sistem perangkat lunak mengalami perbaikan kesalahan yang sebelumnya belum ditemukan di tahapan pengembangan dan pengujian. Selain itu pada tahap ini juga meliputi penambahan fitur dan perubahan sistem untuk menyesuaikan

dengan lingkungan sistem yang baru dan kebutuhan *stakeholder* (Sommerville, 2011). Selama fase maintenance terdapat dua isu yaitu yang pertama adanya permasalahan yang muncul dan harus diselesaikan dan yang kedua adalah perlunya peningkatan fungsionalitas sesuai dengan permintaan *stakeholder*. (Visser 2016). Kemampuan sistem perangkat lunak dalam menerima perubahan pada fase maintenance disebut dengan *maintainability*.

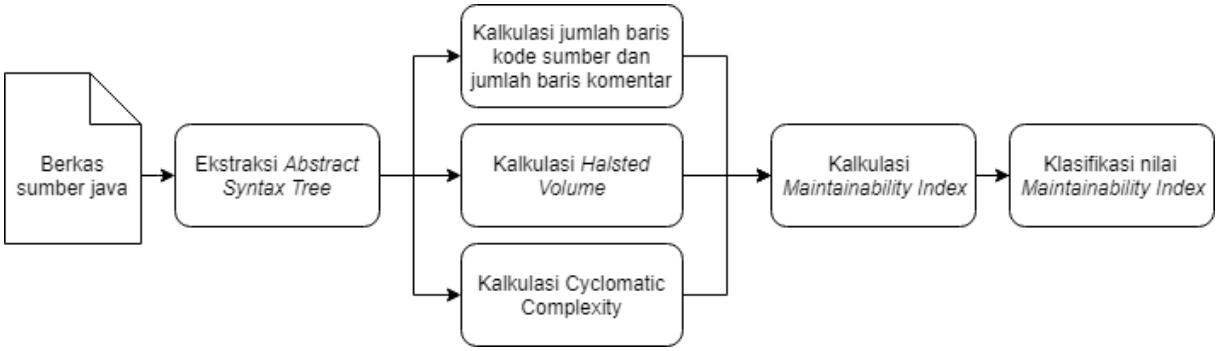
Menurut Chen et al. (2017) pada siklus hidup pengembangan perangkat lunak terdapat beberapa alasan mengapa diperlukan perhitungan nilai *maintainability* suatu perangkat lunak diantaranya, nilai dari maintainability bisa membantu dalam memutuskan apakah komponen suatu perangkat lunak bisa digunakan kembali (*reuse*). Nilai dari maintainability suatu perangkat lunak bisa membantu dalam memutuskan apakah suatu perangkat lunak mudah dirawat atau perlu dilakukan perancangan ulang. Nilai *maintainability* bisa membantu dalam mengestimasi usaha yang diperlukan untuk merawat suatu komponen perangkat lunak. Selain itu, nilai *maintainability* bisa membantu dalam menentukan biaya dan kriteria penerimaan perangkat lunak kepada pengguna.

Banyak software metric atau metode yang bisa digunakan untuk mengukur nilai *Maintainability* perangkat lunak, salah satunya adalah *Maintainability Index* (MI). *Maintainability Index* dihitung berdasarkan nilai dari *Halstead's Volume*, *McCabe's Cyclomatic Complexity*, dan jumlah baris kode sumber (*source code*). Semakin tinggi nilai dari kalkulasi *Maintainability Index* mengindikasikan bahwa kode program memiliki tingkat maintainability yang baik dimana ini akan membuat kode sumber lebih mudah dipahami dan dirawat sehingga akan lebih mudah dalam pencarian dan perbaikan kecacatan (*bugs*), membuat perubahan atau penambahan fungsionalitas baru. Menurut (Laird & Brennan 2006) dalam bukunya mengatakan bahwa *Maintainability Index* bisa digunakan untuk memantau sistem apakah sistem itu mudah di rawat atau tidak, dan *Maintainability Index* bisa memberikan indikasi apakah suatu perangkat lunak perlu dilakukan perancangan ulang. *Maintainability Index* juga berguna ketika *developer* mengubah *code* sistem, *developer* juga akan mengetahui bagaimana dampak perubahan kode sistem terhadap tingkat perawatannya. *Maintainability Index* adalah salah satu metrik yang paling banyak digunakan dalam industri dan telah sepenuhnya sukses diterapkan di sistem perangkat lunak seperti Visual Studio (Chen et al. 2017). Namun untuk saat ini visual studio baru sepenuhnya mendukung bahasa pemrograman C# dan Visual Basic .NET.

Kalkulasi nilai Maintainability Index bila dilakukan dengan manual akan membutuhkan waktu yang lama dan usaha yang besar. Kalkulasi secara manual dilakukan dengan menganalisis setiap *operand* dan *operator* yang ada di kode sumber, kemudian dari *operator* an *operand* yang ditemukan akan di kalkulasi nilai *Halstead Volume* dan *Cyclomatic complexity*. Dari nilai *Halstead Volume* dan *Cyclomatic complexity* yang didapat akan dimasukkan kedalam formula *Maintainability Index*. Kalkulasi secara manual ini akan membutuhkan waktu yang lama apabila kode sumber yang di kalkulasi mempunyai banyak *class* dan banyak *method*. Oleh karena itu, pada penelitian ini akan dikembangkan sebuah kakas bantu yang dapat mengukur nilai *Maintainability Index* secara otomatis, terutama pada kode sumber dengan bahasa pemrograman *java*.

2. METODE KALKULASI

Dalam mengalkulasi nilai *Maintainability Index* pada kode sumber java ada beberapa tahapan yang dilakukan. Tahapan mengalkulasi dimulai dari mengekstraksi *abstract syntax tree* kode sumber *java* untuk menemukan *operand*, *operator*, jumlah baris kode sumber dan jumlah baris komentar. *Operand* dan *operator* yang ditemukan akan dijadikan masukan untuk mengalkulasi nilai *Halstead Volume* dan *Cyclomatic complexity*. Setelah mengalkulasi nilai *Halstead Volume* dan *Cyclomatic complexity* maka hasilnya akan digunakan sebagai masukan pada formula *Maintainability Index*. Setelah nilai *Maintainability Index* diketahui maka akan diklasifikasikan berdasarkan nilainya. Dalam pengklasifikasian ada tiga jenis klasifikasi yaitu *Highly maintainable*, *moderately maintainable*, dan *Difficult to maintain* Alur tahapan kalkulasi akan dipaparkan dalam Gambar 1.



Gambar 1. Alur kalkulasi dan klasifikasi nilai *Maintainability Index*

2.1. *Maintainability Index*

Maintainability Index adalah *software metric* yang mengukur sebuah perangkat lunak mudah atau sulit untuk mengalami perawatan atau perubahan di masa mendatang. *Maintainability Index* mengalkulasi formula berdasarkan *Lines of Code* (LOC), *Cyclomatic Complexity* (CC) dan *Halstead Volume* (HV) (Terzimehić, Schneegass, and Hussmann 2002). Persamaan *Maintainability Index* ditunjukkan pada Persamaan (1), dan klasifikasi *Maintainability Index* ditunjukkan pada Tabel 1.

$$MI = 171 - 5.2 \times \ln(HV) - 0.23 \times CC - 16.2 \times \ln(LOC) + (50 \times \sin(\sqrt{2.46 \times perCM})) \quad (1)$$

Keterangan:
HV = Halstead Metrics Volumes
CC = Cyclomatic Complexity
LOC = Lines of Code
perCM = Percent line of comment

Tabel 1. Klasifikasi Maintainability Index

Nilai *Maintainability Index*	Klasifikasi
MI > 85	*Highly maintainable*
65 < MI ≤ 85	*Moderately maintainable*
MI ≤ 65	*Difficult to maintain*

2.2. *Halstead Metric*

Halstead's metric adalah pengukuran yang dikembangkan untuk mengukur kompleksitas modul suatu program langsung dari kode sumber. Pengukuran dilakukan dengan menentukan ukuran kuantitatif kompleksitas dari *operator* dan *operand* dalam modul sistem (Laird and Brennan 2006). Pada *Halstead's metric* terdapat beberapa enam jenis komponen yaitu:

1. *Length of program*
   Length of program adalah kalkulasi Jumlah total *operator* dan *operan* yang muncul. Persamaan *Length of the program* dirumuskan pada Persamaan (2)

   $$N = N1 + N2 \quad (2)$$

   Keterangan:
   N1 = total semua operator yang muncul
   N2 = total semua operan yang muncul

2. *Vocabulary of the program*
   Vocabulary of the program adalah kalkulasi Jumlah operator dan operand unik yang muncul dalam program. Persamaan *Vocabulary of the program* dirumuskan pada Persamaan (3).

   $$n = n1 + n2 \quad (3)$$

   Keterangan:
   n = *Vocabulary of the program*
   n1 = jumlah *operator* unik
   n2 = jumlah *operand* unik

3. *Volume of the program*
   Volume dalam *Halstead metric* di gunakan untuk mengetahui volume program. Persamaan *Volume of the program* dirumuskan pada Persamaan (4).

   $$V = N \times \log_2 n \quad (4)$$

   Keterangan:
   V = *Volume of the program*
   N = nilai kalkulasi *length of the program*
   n = nilai kalkulasi *vocabulary of the program*

4. *Difficulty*,
   Difficulty dalam *Halstead metric* di gunakan untuk mengetahui kesulitan dan pengembangan program. Persamaan *Difficulty* dirumuskan pada Persamaan (5).

$$D = \frac{n1}{2} \times \frac{N2}{n2} \quad (5)$$

   Keterangan:
   D = *Difficulty*
   N2 = total semua operan yang muncul
   n1 = jumlah *operator* unik
   n2 = jumlah *operan* unik

5. *Effort*
   Effort dalam *Halstead metric* di gunakan untuk mengetahui sumber daya yang digunakan untuk pengembangan program. Persamaan *Effort* dirumuskan pada Persamaan (6).

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

   Keterangan:
   E = *Effort*
   D = nilai dari kalkulasi *Difficulty*
   V = nilai kalkulasi *Volume of the program*

6. *Number of bugs expected in the program*.
   Number of bugs expected in the program dalam *Halstead metric* di gunakan untuk mengetahui prediksi *bug* pada program. Persamaan *Number of bugs expected in the program* dirumuskan pada Persamaan (7).

$$B = \frac{V}{3000} \quad (7)$$

   Keterangan:
   B = *Number of bugs expected in the program*
   V = dari kalkulasi *Volume of the program*

2.3. *Cyclomatic Complexity*

McCabe's Cyclomatic Complexity adalah salah satu metric yang cukup terkenal yang mana metric ini menghitung *control flow* dari suatu modul. Apabila kompleksitas semakin tinggi maka akan modul tersebut akan semakin sulit untuk diuji dan dirawat (Laird and Brennan 2006). Untuk menghitung cyclomatic complexity bisa dilakukan dengan dua cara yaitu yang pertama menghitung berdasarkan banyaknya *nodes* dan *edge* yang dirumuskan pada Persamaan (8). Kedua dengan cara menghitung node percabangan (predicate node) yang dirumuskan pada Persamaan (9).

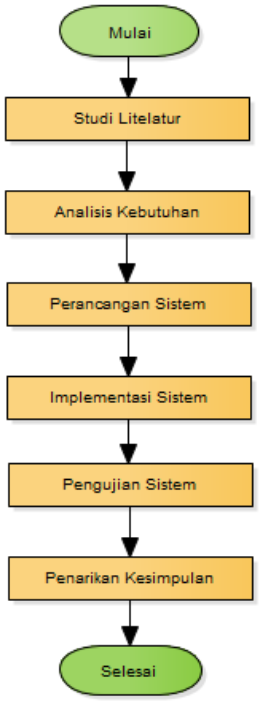
$$V(g) = e - n + 2 \quad (8)$$

$$V(g) = p + 1 \quad (9)$$

Keterangan:
e = jumlah *edge*
n = jumlah *node*
p = jumlah *predicate node*

3. METODOLOGI PENELITIAN

Pada penelitian ini ada enam tahapan metodologi yang dilakukan. Tahapan ini digunakan dalam pengembangan kakas bantu untuk mengukur maintainability index pada perangkat lunak berdasarkan nilai *Halstead Metrics* dan *McCabe's Cyclomatic Complexity*. Tahapan metodologi penelitian ini dipaparkan dalam Gambar 2. Tahapan metodologi yang dilakukan adalah sebagai berikut:



Gambar 2. Diagram alir metodologi penelitian

1. Studi Pustaka
   Pada tahap studi pustaka akan dilakukan penulisan dasar teori pengembangan sistem pada penelitian ini. Dasar teri yang digunakan didapatkan dari referensi jurnal, buku, dan konferensi.
2. Rekayasa Kebutuhan

Pada tahap rekayasa kebutuhan akan ditentukan kebutuhan sistem yang merepresentasikan kemampuan sistem. Hasil dari kebutuhan sistem akan dimodelkan dalam bentuk *sequence diagram* dan *use case scenario*. Tahap pertama yang dilakukan dalam analisis kebutuhan adalah identifikasi aktor. Dalam mengidentifikasi aktor akan menentukan aktor apa saja yang terlibat dalam interaksi sistem. Pada penelitian ini hanya ada satu aktor yaitu *developer*. Setelah identifikasi aktor selesai akan menentukan kebutuhan sistem baik itu kebutuhan fungsional maupun kebutuhan non-fungsional. Pada penelitian ini terdapat sembilan kebutuhan fungsional dan dua kebutuhan non-fungsional. Setelah kebutuhan sistem terdefinisi akan dimodelkan dalam bentuk *use case diagram* dan *use case scenario*. *Use case* diagram dari sistem yang dikembangkan ditunjukkan dalam Gambar 3.

3. Perancangan Sistem

   Setelah tahap rekayasa selesai maka tahap selanjutnya adalah perancangan sistem. Pada tahap ini akan menghasilkan perancangan arsitektur, perancangan komponen, dan perancangan antarmuka. Pada perancangan arsitektur akan menghasilkan *sequence diagram* dan *class diagram*. Pada perancangan komponen akan menghasilkan rancangan algoritme dari method sistem yang akan digunakan. Pada perancangan antarmuka akan dipaparkan mengenai *layout* yang akan digunakan sebagai sarana interaksi antara pengguna dengan sistem.

4. Implementasi Sistem

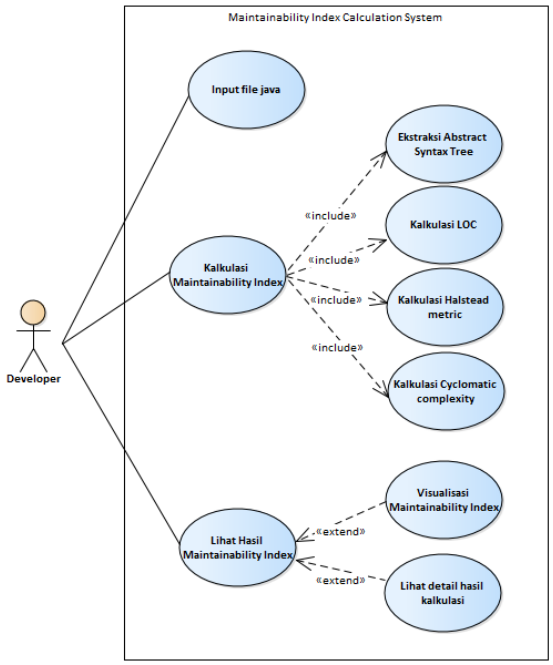
   Pada tahap implementasi sistem akan dilakukan pengimplementasian sistem berdasarkan perancangan yang telah didefinisikan sebelumnya. Pada pengimplementasian kode program akan dilakukan berdasarkan algoritme yang telah didefinisikan pada perancangan komponen. Algoritme yang telah didefinisikan akan diimplementasikan dengan menggunakan bahasa pemrograman *java*.

5. Pengujian dan Analisis

   Pengujian pada penelitian ini menggunakan tiga jenis pengujian yaitu pengujian unit, pengujian integrasi dan pengujian validasi.

6. Penarikan Kesimpulan

   Kesimpulan dilakukan setelah semua tahapan mulai dari studi literatur, analisis kebutuhan, perancangan, implementasi, dan pengujian yang diterapkan sudah selesai dilakukan. Kesimpulan dapat diperoleh secara valid berdasarkan hasil yang diperoleh pada tahap pengujian perangkat lunak.



Gambar 3. *Use Case Diagram Maintainability Index Calculation System*

4. PENGUJIAN DAN ANALISIS

Pada tahap pengujian sistem diuji untuk memastikan bahwa sistem yang kembangkan sesuai dengan kebutuhan dan hasil perancangan yang telah didefinisikan sebelumnya. Pada penelitian ini dilakukan tiga jenis pengujian yaitu pengujian unit, pengujian integrasi dan pengujian validasi. Pada pengujian unit dilakukan dengan menggunakan metode *Whitebox*. Pengujian unit ini dilakukan terhadap tiga method *visit, call* dan *insertIntoHashMaps.* Pada pengujian unit ini didapatkan menghasilkan 15 kasus uji. Pada pengujian integrasi yang dilakukan untuk mengintegrasikan *method visit* dan *method set yang* ada pada *class ClassProperty* dan method set pada *class MethodProperty*. Pada pengujian integrasi ini didapatkan enam kasus uji. Pada pengujian validasi menghasilkan 13 kasus uji untuk kebutuhan fungsional dan dua kasus uji untuk kebutuhan non-fungsional. Pada pengujian validasi dilakukan dengan

menggunakan method *Blackbox*. Hasil pengujian unit, pengujian integrasi dan pengujian validasi yang telah dilakukan menghasilkan status valid untuk keseluruhan kasus uji yang didefinisikan.

Pengujian akurasi sistem pada penelitian ini akan dilakukan dengan membandingkan hasil kalkulasi manual denga hasil kalkulasi sistem. Dalam pengujian ini akan menggunakan proyek aplikasi *AsciidocFX*. *AsciidocFX* didapat dari Github alamat https://github.com/asciidocfx/AsciidocFX. Pada penelitian ini akan diambil 50 *method* sebagai data uji dari proyek aplikasi *AsciidocFX*. Hasil pengujian akurasi didapatkan nilai akurasi 100% dari 50 data uji yang digunakan. Pada pengujian performa sistem akan dilakukan dengan mengamati waktu yang dibutuhkan sistem dalam melakukan kalkulasi *Maintainability Index.* Waktu kalkulasi *Maintainability Index* yang diambil adalah waktu ketika sistem mulai mengekstraksi *abstract syntax tree* sampai sistem selesai dalam mengalkulasi nilai *Halstead metric*, *Cyclomatic complexity* dan *Maintainability Index.* Kalkulasi berkas java yang di pakai adalah berkas java yang mengandung hanya satu *class* dan satu *method.* Pengujian ini akan mengambil sepuluh berkas java yang memiliki jumlah *line of code, operator, operands* dan hasil kalkulasi *Maintainability Index* yang berbeda-beda. Pada pengujian performa sistem didapatkan waktu kalkulasi kurang dari 1000ms untuk setiap *methodnya*.

5. KESIMPULAN

Kesimpulan yang didapat dari penelitian ini:

1. Dari hasil analisis kebutuhan didapatkan sembilan kebutuhan fungsional dan dua kebutuhan non-fungsional. Kebutuhan yang ada dimodelkan dalam bentuk *use case diagram* dan *use case scenario*.
2. Dari hasil perancangan didapatkan hasil perancangan arsitektur berupa rancangan *sequence diagram* dan *class diagram*. Hasil perancangan komponen berupa algoritme yang digunakan di dalam sistem dalam bentuk *pseudocode.* Hasil perancangan antarmuka berupa rancangan layout antar muka dari sistem.
3. Dari pengujian yang dilakukan yaitu pengujian unit, integrasi dan validasi didapatkan status valid pada semua kasus uji yang didefinisikan. Pada pengujian performa sistem didapatkan waktu kalkulasi kurang dari 1000ms untuk setiap *methodnya*. Untuk pengujian akurasi didapatkan nilai akurasi 100% dari 50 data uji yang digunakan.

6. DAFTAR PUSTAKA

Chen, Celia, Reem Alfayez, Kamonphop Srisopha, Barry Boehm, & Lin Shi. 2017. Why Is It Important to Measure Maintainability and What Are the Best Ways to Do It? *Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering Companion, ICSE-C 2017*, [e-journal] pp. 377–78. Tersedia melaui: IEEE Xplore Digital Library <https://ieeexplore.ieee.org/document/7965364/> [Diakses 3 Agustus 2018]

Laird, Linda M., and M. Carol Brennan. 2006. *Software Measurement and Estimation*.[e-book] Hoboken, NJ, USA: John Wiley & Sons, Inc. Tersedia melalui: Google books <https://books.google.co.id/books/about/Software_Measurement_and_Estimation.html?id=3g8wtcpFHZcC&redir_esc=y> [Diakses 19 Juli 2018]

Sommerville, Ian., 2011. Software engineering. 9th ed. London:Addison-Wesley.

Terzimehić, Nađa, Christina Schneegass, and Heinrich Hussmann. 2002. *Reliable Software Technologies — Ada-Europe 2002*. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 2361. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg.

Visser, Joost. 2016. *Building Maintainable Software*. [e-book] Sebastopol, CA: O'Reilly Media, Inc. Tersedia melalui: google books <https://books.google.co.id/books?isbn=1491953497 > [Diakses 19 Juli 2018]