SISTEM PENCEGAH PLAGIARISME DALAM TUGAS MAHASISWA DENGAN *SNAPSHOTTING* DAN *USER ACTIVITY LOGGING*

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
AZZAM SYAWQI AZIZ
NIM: 155150207111132

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2019

PENGESAHAN

SISTEM PENCEGAH PLAGIARISME DALAM TUGAS MAHASISWA DENGAN *SNAPSHOTTING* DAN *USER ACTIVITY LOGGING*

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh:
Azzam Syawqi Aziz
NIM : 155150207111132

Skripsi ini telah diuji dan dinyatakan lulus pada:
2 Juli 2019
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Fajar Pradana, S.ST, M.Eng	Dr.Eng. Fitra Abdurrachman Bachtiar, S.T, M.Eng
NIP: 19871121 201504 1 004	NIP: 198406282019031006

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T, M.T, Ph.D
NIP: 19710518 200312 1 001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar referensi.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akàdemik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 2 Juli 2019

Azzam Syawqi Aziz

NIM : 155150207111132

KATA PENGANTAR

Puji syukur kehadirat Allah SWT yang telah melimpahkan rahmat, taufik dan hidayah-Nya sehingga skirpsi yang berjudul "SISTEM PENCEGAH PLAGIARISME DALAM TUGAS MAHASISWA DENGAN *SNAPSHOTTING* DAN *USER ACTIVITY LOGGING*" ini dapat terselesaikan. Dengan selesainya penulisan skirpsi ini, penulis ingin menyampaikan terima kasih kepada:

1. Allah SWT yang telah memberikan nikmat kesehatan, kekuatan dan kemudahan sehingga penulis dapat menyelesaikan skripsi ini.
2. Ustaz, orang tua dan seluruh keluarga besar penulis yang senantiasa mendoakan dan mendukung demi terselesaikannya skripsi ini.
3. Bapak Fajar Pradana, S.ST, M.Eng, selaku Dosen Pembimbing I yang telah meluangkan waktu untuk membimbing dengan semangat dan sabar sehingga penulis dapat menyelesaikan skripsi ini.
4. Bapak Dr.Eng. Fitra Abdurrachman Bachtiar, S.T, M.Eng, selaku Dosen Pembimbing II yang telah mengarahkan dan membimbing penulis dengan dengan semangat dan sabar sehingga penulis dapat menyelesaikan skripsi ini.
5. Bapak Tri Astoto Kurniawan, S.T, M.T, Ph.D dan Bapak Agus Wahyu Widodo, S.T, M.Cs selaku Ketua Jurusan Teknik Informatika dan Kepala Program Studi Teknik Informatika Fakultas Ilmu Komputer, Universitas Brawijaya.
6. Seluruh pengembang kakas bantu perangkat lunak bebas yang digunakan dalam skripsi ini.
7. Seluruh civitas akademika Teknik Informatika Universitas Brawijaya yang telah banyak memberi bantuan dan dukungan selama penyelesaian skripsi ini.

Penulis menyadari bahwa dalam penelitian ini masih banyak kekurangan. Oleh karena itu, saran dan kritik yang membangun sangat penulis harapkan. Akhir kata penulis berharap penelitian ini dapat membawa manfaat bagi semua pihak yang menggunakannya.

Malang, 2 Juli 2019

Penulis
azzamsa@student.ub.ac.id

ABSTRAK

Azzam Syawqi Aziz, Sistem Pencegah Plagiarisme Dalam Tugas Mahasiswa dengan *snapshotting* dan *User Activity Logging*

Pembimbing: Fajar Pradana, S.ST, M.Eng dan Dr.Eng. Fitra Abdurrachman Bachtiar, S.T, M.Eng

Plagiarisme merupakan hal yang tak terpisahkan dari lingkungan akademisi. Beragam motivasi termasuk bekerja paruh waktu dan dilema sosial menyumbang tingginya persentase plagiarisme di tingkat universitas. Plagiarisme juga telah menumbangkan karier tokoh-tokoh besar. Penanganan plagiarisme dengan metode *plagiarism detection* memiliki banyak kekurangan seperti timbulnya paradigma polisi-kriminal antara murid dan guru, pola pikir nilai *oriented*, dan beberapa kelemahan teknis lain seperti lemah dalam menganalisis dokumen yang telah diterjemahkan. Hellas, Leinonen, dan Ihantola telah melakukan penelitian plagiarisme dengan metode *plagiarism prevention*, tetapi metode yang dilakukan memiliki beberapa kekurangan, yaitu mengabaikan proses pengerjaan tugas sehingga dapat dicurangi siswa dengan membeli tugas, analisis hanya dilakukan di akhir sehingga siswa dapat berkolaborasi melakukan kecurangan selama tugas dikerjakan, tidak memiliki *log* sehingga guru tidak dapat menilai usaha siswa, dan perangkat lunak yang digunakan terikat dengan *platform* tertentu. Penelitian ini bertujuan untuk menyempurnakan penelitian yang telah dilakukan oleh Hellas, Leinonen, dan Ihantola dengan menambahkan proses *snapshotting* yang merekam seluruh perubahan berkas tugas, dan *user activity logging* yang merekam seluruh aktivitas siswa selama mengerjakan tugas. Sistem yang dibangun juga bersifat *agnostic* sehingga guru memiliki kebebasan memilih bentuk tugas dan siswa bebas menggunakan perangkat lunak disukai untuk mengerjakan tugas. Pengujian sistem dilakukan dengan pengujian unit, integrasi, validasi, *automated testing*, dan pengujian *compatibility*. Pengujian unit dan integrasi bernilai valid dan mencapai nilai di atas 70% *code coverage*, pengujian validasi pada 34 kasus uji menghasilkan hasil valid, dan pengujian *compatibility* menunjukkan sistem dapat berjalan pada enam *desktop environment* umum. Hasil *automated testing* yang valid dan tidak memiliki galat dapat mempercepat proses pengujian.

Kata kunci: *plagiarism prevention*, *snapshotting*, *user activity logging*

ABSTRACT

Azzam Syawqi Aziz, Sistem Pencegah Plagiarisme Dalam Tugas Mahasiswa dengan *snapshotting* dan *User Activity Logging*

Supervisors: Fajar Pradana, S.ST, M.Eng and Dr.Eng. Fitra Abdurrachman Bachtiar, S.T, M.Eng

Plagiarism is an inseparable thing from an academic environment. Various motivations including working part-time and social dilemmas contribute to the high percentage of plagiarism at the university level. Plagiarism has also toppled the careers of great figures. Handling plagiarism by plagiarism detection the method has many disadvantages such as the emergence of the police-criminal paradigm between students and teachers, value-oriented mindset, and some other technical weaknesses such as inability in analyzing documents that have been translated. Hellas, Leinonen, and Ihantola have done plagiarism research with the plagiarism prevention method, but the method that is carried out has several disadvantages, namely ignoring the work process so that students can be cheated by buying assignments, the analysis is only done at the end so students can collaborate cheating during assignments, no any log produced so the teacher can't assess student effort, and the software used depends on a certain platform. This research aims to perfect the research done by Hellas, Leinonen, and Ihantola by adding the snapshotting process which records all document changes, and user activity logging which records all student activities while working on a task. The system is also agnostic so that the teacher has the freedom to choose the form of assignments and students are free to use the preferred software to work on the task. The system is tested using unit testing, integration, validation, automated testing, and compatibility testing. Unit testing and integration are valid and scored above 70% code coverage, validation testing in 34 test cases produced valid results, and the compatibility test shows that the system able to run on six general desktop environment. The automated testing result is valid and does not contain errors can speed up the testing process.

Keywords: *plagiarism prevention*, *snapshotting*, *user activity logging*

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Plagiarisme dalam lingkungan akademisi adalah penggunaan kata atau ide dari sebuah sumber tanpa menyertakan pengakuan sebagaimana ditentukan oleh prinsip-prinsip akademik (Meuschke & Gipp, 2013). Meuschke & Gipp (2013) berpendapat bahwa plagiarisme dalam lingkungan akademisi tidak selalu dapat dikaitkan dengan pencurian. Plagiarisme tersebut dapat terjadi karena unsur tidak sengaja, seperti penulis yang lupa memasukkan sitiran, melakukan sitiran pada sumber yang salah, atau tidak sengaja melakukan sitiran pada dirinya sendiri. Terdapat beberapa tipe plagiarisme, diantaranya menyalin kata, parafrase, dan menerjemahkan kata (Kiss, 2013).

Plagiarisme menjadi hal yang seakan tak terpisahkan di lingkungan akademisi. Mahasiswa seringkali melakukan plagiarisme dalam tugas maupun ujian. Ketersediaan sumber daya *internet* dan kemudahan mencari informasi berkontribusi pada munculnya plagiarisme oleh siswa (Born, 2003). Plagiarisme di Indonesia tidak hanya dilakukan oleh mahasiswa saja, melainkan juga peneliti dan dosen dari universitas-universitas ternama (Agustina & Raharjo, 2017). Kiss (2013) juga menyampaikan bahwa banyak insiden plagiarisme yang menyebabkan karier tokoh-tokoh besar hancur. Banyaknya motivasi untuk melakukan plagiarisme di tingkat universitas, seperti mahasiswa yang bekerja paruh waktu dan dilema sosial, membuat tingkat plagiarisme di universitas semakin tinggi (Park, 2004). Oleh karena alasan-alasan tersebut diperlukan penanganan plagiarisme.

Metode dalam plagiarisme dapat dikategorikan menjadi dua. Pertama adalah metode pendeteksian atau *plagiarism detection*. Metode ini menitikberatkan pada proses pendeteksian apakah seorang siswa melakukan plagiarisme setelah siswa tersebut menyelesaikan aktivitas yang memungkinkan terjadinya plagiarisme, seperti kuis tugas rumah (*take home quiz*), ujian harian dan ujian akhir. Hasil dari tugas atau ujian tersebut akan melewati proses pendeteksian plagiarisme, seperti menggunakan alat bantu perangkat lunak *plagiarism detection software* (Neill & Shanmuganthan, 2004) atau dilakukan secara manual. Metode kedua adalah metode pencegahan atau *plagiarism prevention*. Penanganan plagiarisme pada metode pencegahan ditekankan pada saat sebelum siswa memulai aktivitas atau selama mengerjakan aktivitas yang memungkinkan terjadinya plagiarisme. Proses pencegahannya, seperti mengimplementasikan *formative assessment* pada siswa (Leung & Cheng, 2017), mengedukasi siswa tentang plagiarisme, mengajarkan cara sitir-

an dengan benar, membangun atmosfer yang sehat di kelas (McLafferty & Foust, 2004), dan melakukan pencegahan dengan bantuan perangkat lunak (Hellas, Leinonen, & Ihantola, 2017).

Penggunaan *Plagiarism Detection Software* atau *PDS* memudahkan para guru untuk menemukan siswa tersangka plagiarisme. Tetapi penggunaan *PDS* memiliki kekurangan, seperti dapat menyebabkan paradigma siswa kepada guru bagaikan hubungan polisi-kriminal daripada guru-murid (Howard, 2002), membentuk pola pikir siswa menjadi nilai *oriented* dari pada *goal oriented* (Dweck & Leggett, 1988), membuat siswa mencari cara-cara baru agar tidak terdeteksi *PDS* (Kiss, 2013); mengubah teks menjadi gambar, membubuhkan *invinsible character* dan mengubah *character map*, yaitu mengubah huruf "a" menjadi karakter "c" dan mengubah huruf lain ke karakter lain; lemah dalam menganalisis dokumen yang sudah diterjemahkan (Meuschke & Gipp, 2013), memiliki batasan dalam proses menganalisis berbagai macam gaya sitiran (Modiba, Pieterse, & Haskins, 2016) dan berbagai macam bahasa (Martins, et al., 2014), tidak dapat melacak bantuan eksternal yang datang dari luar kelas (Hellas, Leinonen, & Ihantola, 2017), harga yang ditawarkan tidak terjangkau sehingga negara berkembang tidak dapat mengadopsinya (Agustina & Raharjo, 2017; Modiba, Pieterse, & Haskins, 2016) dan pelanggaran hak-hak intelektual dan hak milik siswa seperti yang dilakukan *Turnitin*. *Turnitin* menyalin dan menyimpan seluruh tugas siswa ke dalam basis datanya, terlepas siswa tersebut melakukan plagiarisme atau tidak. Oleh karena itu beberapa universitas meninggalkannya (Park, 2004).

PDS tidak dapat mendeteksi semua kasus yang ada pada permasalahan plagiarisme (Martins, et al., 2014). Oleh karena itu, pencegahan plagiarisme sangat diutamakan (Garner, et al., 2012). Metode pencegahan plagiarisme (*plagiarism prevention*) dengan bantuan perangkat lunak memberikan kemampuan pada guru untuk menganalisis kecurangan selama proses pengerjaan tugas, seperti mendeteksi plagiarisme dari sumber non-digital dan bantuan eksternal. Selain itu, guru dapat menganalisis pola proses pengerjaan dari *log* yang dihasilkan sehingga dapat menganalisis adanya kejanggalan maupun menemukan siswa yang lemah pada pelajaran tertentu. Kemampuan-kemampuan tersebut tidak dapat dilakukan oleh *PDS* (Hellas, Leinonen, & Ihantola, 2017).

Hellas, Leinonen, & Ihantola (2017) menggunakan metode pencegahan plagiarisme (*plagiarism prevention*) dengan bantuan perangkat lunak. Metode tersebut merekam waktu mulai, waktu selesai dan alamat *IP* siswa dengan bantuan perangkat lunak *JPlag*. Pada akhir prosesnya dilakukan analisis hasil tugas menggunakan algoritme *edit-distance*. Rekaman waktu mulai dan waktu selesai digu-

nakan untuk menemukan siswa yang mengerjakan pada waktu yang bersamaan, rekaman alamat *IP* digunakan untuk menenemukan siswa yang mengerjakan pada tempat yang sama, dan analisis hasil akhir tugas menggunakan *edit-distance* digunakan untuk menemukan siswa yang mendapatkan bantuan dari sumber eksternal atau tidak dikerjakan dengan kemampuannya sendiri. Tetapi metode ini memiliki beberapa kelemahan, seperti tidak merekam segala aktivitas yang dilakukan selama mengerjakan tugas sehingga guru hanya menganalisis hasil akhir tugas dan mengabaikan proses pengerjaan tugas. Hal ini dapat dicurangi siswa dengan menyelesaikan tugas dengan membeli (Leung & Cheng, 2017). Analisis bantuan eksternal hanya dilakukan pada hasil akhir tugas sehingga kecurangan selama mengerjakan tugas dapat dicurangi dengan bantuan mencari jawaban melalui *browser* maupun berkolaborasi menggunakan sosial media. Metode ini juga tidak memiliki *log* pengerjaan sehingga guru tidak dapat menilai usaha yang telah dilakukan siswa (Hellas, Leinonen, & Ihantola, 2017). Selain itu, proses rekaman dilakukan dengan bantuan perangkat lunak yang terikat dengan *platform* tertentu sehingga memiliki batasan bentuk tugas yang akan diberikan dan batasan pilihan kakas bantu yang akan digunakan siswa.

Kelemahan yang ada pada metode yang dilakukan Hellas, Leinonen, & Ihantola (2017) akan disempurnakan dengan menyelesaikan masalah yang telah disebutkan sebelumnya dengan cara menambahkan proses *snapshotting* dan *user activity logging* selama pengerjaan tugas. Proses *snapshotting* akan merekam seluruh perubahan dokumen. Oleh karena itu, *snapshotting* dapat menghindari adanya siswa yang melakukan kecurangan melalui bantuan eksternal seperti membeli tugas dari orang lain. *User activity logging* merekam seluruh aktivitas siswa selama pengerjaan tugas, seperti nama *login*, nama peranti yang digunakan, *window* yang sedang aktif, dan seluruh *window* yang terbuka. Maka *user activity logging* dapat menghindari adanya bantuan eksternal melalui *browser* maupun perangkat lunak sosial media ketika proses mengerjakan tugas. Proses perekaman data menggunakan perangkat lunak yang bersifat *agnostic* sehingga guru tidak memiliki batasan dalam memilih bentuk tugas yang akan diberikan dan siswa memiliki kebebasan menggunakan kakas bantu yang disukai. Selain itu, seluruh proses pengerjaan tugas direkam sehingga guru dapat menganalisis seberapa besar kesugguhan dan seberapa banyak keterlibatan siswa dalam tugas yang diberikan.

Penelitian ini akan mengimplementasikan metode *snapshotting* dan *user activity logging*, dengan tambahan algoritme *edit-distance* yang didapatkan dari penelitian sebelumnya. Metode *snapshotting* akan dibangun di atas *distributed version control* untuk merekam seluruh perubahan dokumen tugas. Metode *acti-*

vity logging digunakan untuk merekam segala aktivitas siswa selama pengerjaan tugas. Algoritme *edit-distance* digunakan untuk menghitung nilai *edit-distance* pada berkas tugas. Maka seluruh usaha, keputusan, keterlibatan dan aktivitas siswa dapat dianalisis oleh guru.

1.2 Rumusan Masalah

Berdasarkan permasalahan yang diangkat, maka dapat dirumuskan masalah sebagai berikut:

1. Bagaimana hasil analisis kebutuhan dalam pengembangan aplikasi pencegah plagiarisme dengan menggunakan metode *snapshotting* dan *user activity logging*?
2. Bagaimana hasil perancangan dan implementasi aplikasi pencegah plagiarisme dengan menggunakan metode *snapshotting* dan *user activity logging*?
3. Bagaimana hasil pengujian dari pengembangan aplikasi pencegah plagiarisme dengan menggunakan metode *snapshotting* dan *user activity logging*?

1.3 Tujuan

Tujuan penelitian ini adalah sebagai berikut:

1. Menganalisis kebutuhan yang diperlukan untuk membangun aplikasi pencegah plagiarisme dengan menggunakan metode *snapshotting* dan *user activity logging*.
2. Merancang dan mengimplementasikan aplikasi pencegah plagiarisme dengan menggunakan metode *snapshotting* dan *user activity logging* sesuai dengan hasil analisis kebutuhan.
3. Menguji aplikasi pencegah plagiarisme dengan menggunakan metode *snapshotting* dan *user activity logging* untuk memastikan bahwa aplikasi yang dibangun sesuai dengan hasil rancangan dan kebutuhan sebelumnya.

1.4 Manfaat

Manfaat yang didapatkan dari penelitian ini adalah:

1. Menyediakan aplikasi yang dapat mencegah siswa melakukan plagiarisme.

2. Menyediakan aplikasi yang dapat memberikan petunjuk siswa yang melakukan kecurangan selama mengerjakan tugas.
3. Menyediakan aplikasi yang dapat memberikan petunjuk siswa yang lemah pada suatu pelajaran tertentu.

1.5 Batasan Masalah

Penelitian ini memiliki batasan-batasan sebagai berikut:

1. Sistem yang dikembangkan hanya dapat mengolah tugas dengan bentuk *plain text format*, sistem tidak dapat mengolah *binary format*.
2. Sistem yang dikembangkan hanya dapat berjalan pada sistem operasi *GNU-/Linux*.
3. Sistem yang dikembangkan membutuhkan perangkat lunak pengolah kata yang memiliki fitur *autosave*.

1.6 Sistematika Pembahasan

Penjelasan singkat mengenai pembahasan dan langkah-langkah yang dilakukan dalam setiap bab pada penelitian ini tersusun sebagai berikut:

BAB I: PENDAHULUAN

Bab ini menjelaskan tentang latar belakang pengembangan aplikasi pencegah plagiarisme dengan menggunakan metode *snapshotting* dan *user activity logging*, rumusan masalah, tujuan penelitian, manfaat penelitian, batasan masalah, dan sistematika pembahasan.

BAB II: LANDASAN KEPUSTAKAAN

Bab ini menjelaskan tentang penelitian-penelitian sebelumnya yang terkait dengan pencegahan plagiarisme, teori-teori yang menjadi landasan dalam penelitian, dan penjelasan kakas bantu yang digunakan dalam penelitian.

BAB III: METODOLOGI PENELITIAN

Bab ini menjelaskan tahapan-tahapan penelitian yang dilakukan untuk mengembangkan aplikasi pencegah plagiarisme dengan menggunakan metode *snapshotting* dan *user activity logging*. Tahapan-tahapan tersebut meliputi studi literatur, rekayasa kebutuhan, perancangan, implementasi, pengujian, dan penarikan kesimpulan dan saran.

BAB IV: REKAYASA KEBUTUHAN

Bab ini menjelaskan proses rekayasa kebutuhan untuk membangun sistem pencegah plagiarisme dengan menggunakan metode *snapshotting* dan *user activity logging*. Tahapan rekayasa kebutuhan meliputi analisis kebutuhan, identifikasi aktor, spesifikasi dan manajemen kebutuhan, dan pemodelan kebutuhan. Kebutuhan tersebut meliputi kebutuhan fungsional dan non-fungsional. Pada tahapan ini juga dijelaskan deskripsi umum sistem.

BAB V: PERANCANGAN & IMPLEMENTASI

Bab ini menjelaskan proses perancangan dan implementasi dari aplikasi yang dibangun. Tahapan perancangan meliputi perancangan arsitektur, komponen, dan antarmuka. Tahapan implementasi meliputi implementasi kode, dan antarmuka.

BAB VI: PENGUJIAN

Bab ini menjelaskan proses pengujian perangkat lunak yang dibangun. Pengujian yang dilakukan berupa pengujian unit, pengujian integrasi, pengujian validasi, *automated testing*, dan pengujian *compatibility*.

BAB VII: PENUTUP

Bab penutup menjelaskan kesimpulan dari hasil penelitian yang telah dikerjakan, dan saran-saran untuk perbaikan penelitian selanjutnya.

BAB 2 LANDASAN KEPUSTAKAAN

2.1 Kajian Pustaka

Penelitian-penelitian sebelumnya pada bidang plagiarisme banyak berfokus pada *plagiarism detection*, baik menggunakan cara non-teknis seperti yang dilakukan Leung & Cheng (2017) dan McLafferty & Foust (2004), maupun menggunakan cara teknis dengan bantuan *plagiarism detection software* atau biasa disebut *plagiarism checker*. Berbeda dengan yang lain, Hellas, Leinonen, & Ihantola (2017) memfokuskan penelitiannya pada *plagiarism prevention* dengan studi kasus *take-home exams*.

Hellas, Leinonen, & Ihantola (2017) menggunakan metode pencegahan plagiarisme (*plagiarism prevention*) dengan bantuan perangkat lunak. Penelitian tersebut menggunakan kakas bantu *JPlag* dan algoritme *edit-distance*. *JPlag* digunakan untuk merekam waktu mulai tugas, waktu selesai tugas, dan alamat *IP* siswa. Algoritme *edit-distance* digunakan untuk menghitung nilai *edit-distance* pada berkas tugas. Rekaman waktu mulai dan waktu selesai digunakan untuk menemukan siswa yang mengerjakan pada waktu yang bersamaan, rekaman alamat *IP* digunakan untuk menenemukan siswa yang mengerjakan pada tempat yang sama, dan analisis nilai *edit-distance* pada berkas tugas digunakan untuk menemukan siswa yang mendapatkan bantuan dari sumber eksternal atau tidak mengerjakan dengan kemampuannya sendiri.

Metode tersebut memiliki beberapa kelemahan, seperti tidak merekam segala aktivitas yang dilakukan selama mengerjakan tugas sehingga guru hanya menganalisis hasil akhir tugas dan mengabaikan proses pengerjaan tugas. Hal ini dapat dicurangi siswa dengan menyelesaikan tugas dengan membeli (Leung & Cheng, 2017). Analisis bantuan eksternal hanya dilakukan pada hasil akhir tugas sehingga kecurangan selama mengerjakan tugas dapat dicurangi dengan bantuan melalui *browser* maupun perangkat lunak sosial media. Proses rekaman dilakukan dengan bantuan perangkat lunak yang terikat dengan *platform* tertentu sehingga memiliki batasan bentuk tugas yang akan diberikan dan batasan plihan kakas bantu yang akan digunakan siswa. Metode ini juga tidak memiliki *log* pengerjaan sehingga guru tidak dapat menilai usaha yang telah dilakukan siswa.

Penelitian yang diajukan akan menerapkan metode *snapshotting*, *user activity logging*, dan algoritme *edit-distance*. Metode *snapshotting* ditujukan untuk merekam semua perubahan berkas tugas sehingga guru dapat menganalisis keputusan, usaha, keterlibatan siswa dalam proses pengerjaan tugas, dan menemukan

personal siswa yang lemah. Metode *user activity logging* digunakan untuk merekam seluruh aktivitas siswa selama proses pengerjaan tugas. Algoritme *edit-distance* digunakan untuk menghitung nilai *edit-distance* pada berkas tugas. Perangkat lunak yang akan dibangun bersifat *platform agnostic* sehingga siswa mendapat kebebasan untuk menggunakan kakas bantu yang disukai dalam mengerjakan tugas, dan guru tidak memiliki batasan dalam memilih bentuk tugas yang akan diberikan.

2.2 Rekayasa Perangkat Lunak

Rekayasa perangkat lunak atau *software engineering* memiliki definisi standar dari IEEE (1990) yang berbunyi "The systematic approach to the development, operation, maintenance and retirement of software". Hal ini senada dengan pendapat Sommerville (2014) yang mendefinisikan rekayasa perangkat lunak sebagai suatu teknik disiplin yang fokus pada seluruh aspek dalam produksi perangkat lunak dari konsep awal hingga operasi dan pemeliharaan. Dalam hal ini Bell (2000) berpendapat bahwa tidak ada satu metode terbaik untuk melakukan rekayasa perangkat lunak, akan tetapi terdapat beberapa macam pendekatan-pendekatan tertentu.

Rekayasa perangkat lunak menjadi penting karena dua hal. Pertama karena semakin banyaknya aktivitas manusia yang bergantung pada perangkat lunak sehingga kita membutuhkan suatu cara untuk menghasilkan perangkat lunak yang handal dan terpercaya. Kedua karena penggunaan metode rekayasa perangkat lunak dapat membuat pengembangan perangkat lunak semakin murah, daripada menulisnya secara langsung tanpa menggunakan metode. Mengabaikan penggunaan metode rekayasa perangkat lunak dalam pengembangan perangkat lunak memiliki probabilitas terjadinya harga yang melambung untuk proses pengujian dan kesulitan dalam pemeliharaan jangka panjang (Sommerville, 2014).

Aplikasi metode rekayasa perangkat lunak yang benar dapat menghasilkan perangkat lunak yang baik. Perangkat lunak yang baik memiliki empat kriteria atau attribut, yaitu *acceptability, dependability* dan *security, efficiency,* dan *maintainability*. *Acceptability* merupakan karakteristik di mana perangkat lunak harus dapat diterima oleh penggunanya. *Dependability* merupakan karakteristik perangkat lunak yang memiliki sifat keamanan dan keandalan. Perangkat lunak tidak boleh menyebabkan kerugian fatal ketika terjadi kegagalan sistem. Atribut *efficiency* memungkinkan perangkat lunak berjalan tanpa menggunakan sumber daya yang ber-

lebihan atau boros. *Maintability* memungkinan perangkat lunak mudah menerima perubahan (Sommerville, 2014).

2.3 Pengembangan Perangkat Lunak

Pengembangan perangkat lunak memiliki beberapa model pengembangan, arsitektur, pendekatan, dan pemodelan. Salah satu model pengembangan perangkat lunak adalah model *waterfall*. Model *waterfall* memiliki batasan di mana suatu tahapan harus diselesaikan terlebih dahulu sebelum melanjutkan ke tahapan selanjutnya. Salah satu arsitektur yang umum digunakan dalam pengembangan perangkat lunak berbasis *desktop* adalah *Model-View-controller*, arsitektur ini memisahkan bagian-bagian perangkat lunak sesuai dengan tanggung jawabnya. Dari sisi pendekatan terdapat beberapa pendekatan, salah satunya adalah pendekatan berorientasi objek yang di dalamnya terdapat aktivitas *object oriented analysis (OOA)*, dan aktivitas *object oriented design (OOD)*. Pengembangan perangkat lunak diakhiri dengan pengujian untuk memastikan perangkat lunak sudah sesuai dengan kebutuhan yang didefinisikan. Tahapan pengujian diantaranya adalah pengujian unit, integrasi dan validasi. Metode pengujian diantaranya adalah metode pengujian *black-box* dan *white-box*. Sedangkan teknik pengujian diantaranya adalah teknik pengujian *basis path*, *boundary value analisys* dan *equivalence partitioning*. Pengujian *compatibility* merupakan pengujian yang dilakukan untuk memastikan *compatibility* sistem, dan *automated testing* merupakan proses yang dilakukan untuk mempercepat pengujian.

2.3.1 Model *Waterfall*

Terdapat berbagai macam model dalam proses pengembangan perangkat lunak. Tidak ada model *universal* yang dapat digunakan untuk semua situasi. Model *waterfall* cocok digunakan untuk sistem yang permasalahan kebutuhannya telah dipahami dengan baik (Pressman, 2010). Model *waterfall* memiliki beberapa tahapan seperti pada Gambar 2.1, yaitu *requirement definition, system* dan *software design, implementation* dan *unit testing, integration* dan *system testing, operation* dan *maintenance*. Sesuai dengan rumusan masalah penelitian yang telah dipaparkan, tahapan yang dilakukan terbatas pada tahapan pengujian.

1. *Requirements analysis* dan *definition*

Pada tahapan ini kebutuhan, batasan, dan tujuan sistem ditentukan dengan hasil konsultasi bersama pengguna sistem. Ketiga data tersebut kemudian didefi-

nisikan lebih mendetail dan dijadikan sebagai spesifikasi sistem. Terdapat beberapa tahapan dalam proses rekayasa kebutuhan, diantaranya adalah elisitasi atau analisis, spesifikasi, manajemen, dan pemodelan. Tahapan analisis kebutuhan dilakukan untuk menggali kebutuhan, seperti bertanya kepada pengguna, dan menentukan objek dari sistem yang ingin dibangun. Spesifikasi kebutuhan dilakukan untuk menjelaskan kebutuhan secara lebih mendetail dengan jelas, tidak ambigu, mudah dipahami, dan konsisten. Manajemen kebutuhan dilakukan untuk mengidentifikasi, mengontrol, dan melacak kebutuhan, dan pemodelan kebutuhan dilakukan untuk menggambarkan kebutuhan secara visual maupun tekstual (Pressman, 2010).

2. *System* dan *software design*

Tahapan ini mengalokasikan kebutuhan sistem terhadap perangkat lunak maupun perangkat keras dan membentuk arsitektur sistem secara keseluruhan. Pada tahapan ini dilakukan identifikasi dan penjelasan pokok sistem perangkat lunak dan hubungannya satu sama lain. Di dalam *design* atau perancangan sistem terdapat beberapa tahapan, diantaranya adalah perancangan arsitektur, komponen, dan antarmuka. Perancangan arsitektur dilakukan untuk merancang desain arsitektur yang merepresentasikan struktur data dan komponen program yang dibutuhkan untuk membangun sistem. Perancangan komponen dilakukan untuk merancang struktur data, algoritme, dan mekanisme komunikasi antar komponen dalam sistem. Perancangan antarmuka dilakukan untuk merancang media yang efektif untuk komunikasi antar manusia dan komputer (Pressman, 2010).

3. *Implementation* dan *unit testing*

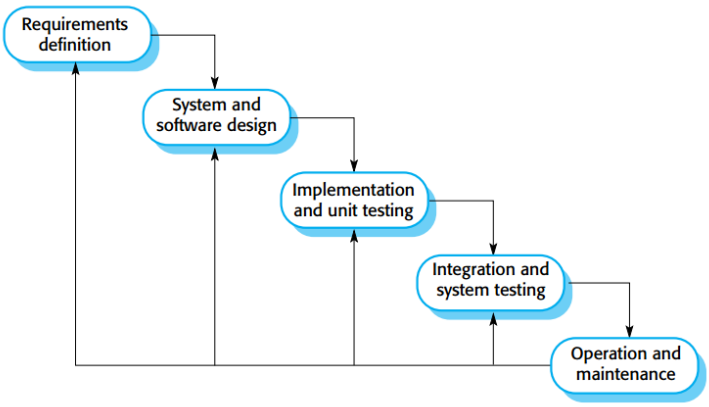
Tahapan ini merupakan aktivitas implementasi sistem. Sistem dibangun sebagai *unit-unit* yang kemudian diverifikasi agar memenuhi kriteria spesifikasi yang telah ditentukan. Tahapan implementasi dilakukan untuk mengimplementasikan tahapan-tahapan pada proses perancangan sebelumnya. Sementara itu, terdapat beberapa tahapan dalam proses pengujian, diantaranya pengujian unit, integrasi, validasi, dan pengujian sistem (Pressman, 2010).

4. *Integration* dan *system testing*

Pada tahapan ini *unit-unit* yang telah dibangun diintegrasikan dan diuji sebagai sistem yang lengkap, kemudian diserahkan kepada pengguna.

5. *Operation* dan *maintenance*

Tahap ini merupakan tahap pemeliharan perangkan lunak, di mana di dalamnya terdapat aktivitas berupa perbaikan galat yang terdapat pada perangkat lunak, peningkatan, dan penyempuraan perangkat lunak.



Gambar 2.1 Model *waterfall* (Sommerville, 2014)

2.3.2 Arsitektur *Model View Controller*

Model-View-Controller (MVC) merupakan sebuah model infrastruktur yang memisahkan antara *user interface* dari fungsionalitas inti sistem. *Model* memuat segala *content* dan fungsionalitas yang spesifik pada sistem yang dibangun, *processing logic* dan akses ke sumber informasi atau data eksternal. *View* memuat segala fungsi spesifik *interface*, dan segala proses fungsionalitas yang dibutuhkan *end-user*. *Controller* mengatur akses ke *model* dan *view* serta mengatur aliran data antara keduanya (Pressman, 2010). Penggunaan arsitektur *MVC* akan menghasilkan tiga *class* utama, yaitu *class view, model*, dan *class controller*. *Class model* akan memuat seluruh fungsionalitas inti yang spesifik pada sistem. Hal ini memberikan banyak keuntungan tatkala sistem akan dikembangkan lebih lanjut, karena pada pengembangan selanjutnya pengembang cukup fokus pada bagian *controller* dan *view*.

2.3.3 Pendekatan Berorientasi Objek

Langkah awal yang signifikan terhadap desain berorientasi objek terlahir dalam komunitas pengguna bahasa pemrograman *Ada*. Hal ini bermula ketika Grady Booch merasionalkan metode yang diciptakan Russell J. Abbott pada tahun 1983. Grady Booch kemudian menyebut karyanya sebagai *object-oriented design*. Keduanya merekomendasikan bahwa untuk memulai membangun desain berorientasi objek, dimulai dari deskripsi informal dari sistem yang akan dibangun. Dari deskripsi informal tersebut pengembang sistem dapat mengidentifikasi *class* dan objek dari kata benda dan operasi dari kata kerja yang ditemukan. Beberapa waktu setelahnya Grady Booch mengemukakan konsep baru yang tidak lagi berbasis

pada deskripsi naratif melainkan menggabungkan *object-oriented design* dengan metodologi yang sudah ada, kemudian menyebutnya sebagai *object-oriented development* (Capretz, 2003).

Pada tahun 1990-an Grady Booch bersama dengan dua orang rekannya, yaitu James Rumbaugh dan Ivar Jacobson mulai mengerjakan suatu metode yang diharapkan bersifat universal. Metode tersebut dibangun berdasarkan kombinasi dari fitur-fitur terbaik pada metode-metode *object-oriented analisis* dan *design method* yang ada. Metode ini juga mengadopsi fitur-fitur yang diajukan oleh ahli lain dalam rekayasa perangkat lunak. Karya ketiga orang tersebut yang dikenal sekarang dengan *Unified Modeling Language* (*UML*). *UML* berisi notas-notasi yang dapat digunakan untuk pemodelan dan pengembangan sistem berorientasi objek. Tujuh tahun setelahnya, yaitu pada tahun 1997 *UML* menjadi standar *de facto* untuk membangun perangkat lunak berorientasi objek (Pressman, 2010).

Membangun perangkat lunak dengan pendekatan berorientasi objek memiliki aktivitas *object oriented analysis (OOA)* yang di dalamnya terdapat proses ekstraksi *class* atau *object* dari pernyataan kebutuhan. Selanjutnya terdapat aktivitas *object oriented design (OOD)* yang di dalamnya terdapat proses *refine* dan *extend class* yang didapatkan dari *OOA*. Dalam pengembangan ini umumnya terdapat *boundary class* dan *controller class*. *Boundary class* bertindak sebagai pembatas atau *interface* antara pengguna dan sistem. *Boundary class* bertanggung jawab mengatur cara sebuah entitas *object* direpresentasikan kepada pengguna. *Controller class* mengatur pembuatan atau pembaharuan pada entitas *object*, penghubung antar *boundary* dan data yang didapatkan, dan komunikasi antar *object* (Pressman, 2010).

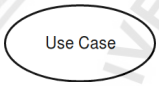
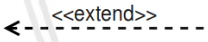
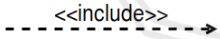
2.3.4 Pemodelan Berorientasi Objek

Notasi yang digunakan untuk pemodelan berorientasi objek adalah *Unified Modeling Language* (*UML*). *UML* merupakan standar *de facto* dalam pemodelan berorientasi objek. Terdapat beberapa diagram inti dalam *UML*, diantaranya adalah *use case diagrams, sequence diagrams* dan *class diagrams*. *Use case diagrams* menunjukkan interaksi antar pengguna dan sistem secara visual. *Sequence diagrams* menunjukkan interaksi antar komponen atau objek di dalam sistem. *Class diagrams* menunjukkan struktur sistem dan relasi antar komponen pada sistem (Sommerville, 2014).

2.3.4.1 *Use Case Diagram*

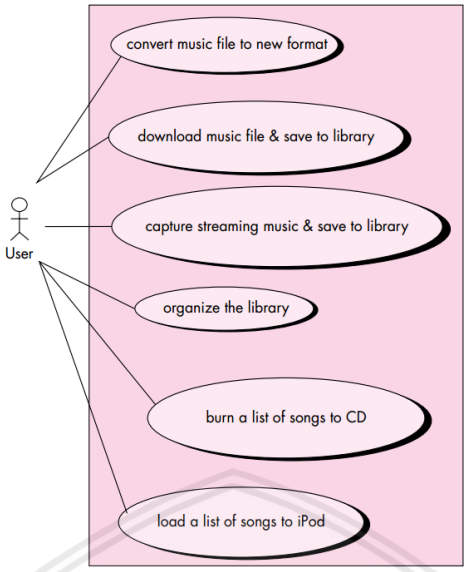
Use case diagram merupakan sebuah diagram statis yang menggambarkan interaksi antara pengguna dan sistem secara visual. *Use case diagram* membantu menentukan fungsionalitas sebuah sistem dari sudut pandang pengguna (Pressman, 2010). *Use case diagram* tidak menggambarkan tahapan. *Use case* digambarkan dengan bentuk oval dan aktor digambarkan dengan gambar orang (*stick figure*). Notasi *use case diagram* dan deskripsinya dipaparkan pada Tabel 2.1, dan contoh *use case diagram* terdapat dalam Gambar 2.2.

Tabel 2.1 Notasi *use case diagram* (Rumbaugh, Jacobson, & Booch, 2004)

Notasi	Deskirpsi
	Notasi aktor menggambarkan pengguna atau sistem luar yang berinteraksi dengan sistem.
	Notasi *use case* merepresentasikan fungsionalitas yang dapat dijalankan oleh aktor.
	Notasi asosiasi merupakan jalur komunikasi antara aktor dan *use case*.
	Notasi *extends* merupakan sisipan fungsionalitas tambahan terhadap *base use case* yang tidak mengetahui sisipan tersebut.
	Notasi *include* merupakan sisipan fungsionalitas tambahan terhadap *basecase* yang secara eksplisit menjelaskan sisipan tersebut.
	Notasi *use case generalization* merupakan relasi antara *general use case* dan *specific use case* yang mewarisi dan menambahkan fitur padanya.

2.3.4.2 *Sequence Diagram*

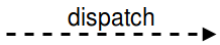
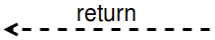
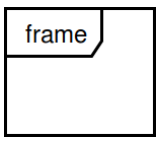
Sequence diagram merupakan sebuah diagram dinamis yang menggambarkan interaksi antar objek di dalam sistem pada saat sistem mengeksekusi sebuah tugas. Terjadi pertukaran *messages* antar objek pada sistem untuk menyelesaikan

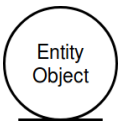
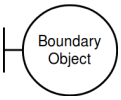
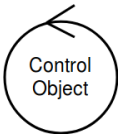


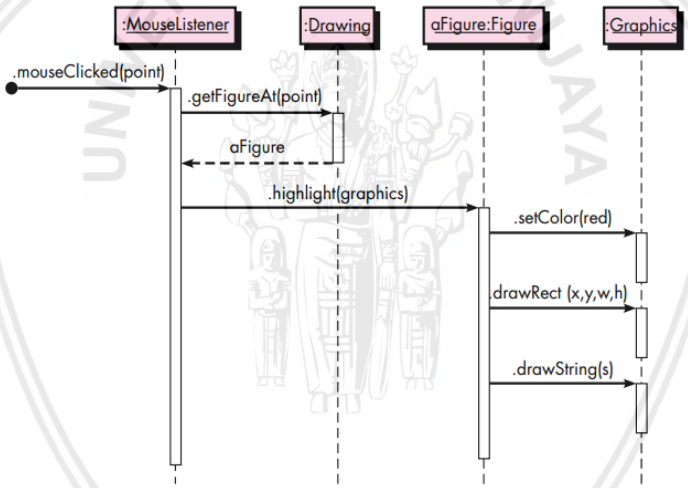
Gambar 2.2 Contoh *use case diagram* (Pressman, 2010)

sebuah tugas (Pressman, 2010). Notasi *sequence diagram* dan deskripsinya terdapat pada Tabel 2.2, dan contoh *sequence diagram* terdapat dalam Gambar 2.3.

Tabel 2.2 Notasi *sequence diagram*

Notasi	Deskripsi
	Notasi *lifeline* menggambarkan waktu aktif (*life*) suatu objek.
	Notasi *object* merepresentasikan objek yang berinteraksi dengan objek lainnya.
	Notasi *activation bar* merepresentasikan lama suatu objek aktif berinteraksi.
	Notasi *method invocation* merepresentasikan suatu *method* dari sebuah objek yang panggil oleh objek lainnya.
	Notasi *return* merepresentasikan data yang dikembalikan oleh method *calle* kepada method *caller*.
	Notasi *frame* menggambarkan lingkup suatu interaksi dengan interaksi lainnya atau lingkup suatu interaksi dan dunia luar.

	Notasi *entity object* merepresentasikan objek yang bertanggung jawab untuk mengatur data dalam sebuah sistem.
	Notasi *boundary object* merepresentasikan objek yang bertanggung jawab untuk mengatur alur sebuah objek direpresentasikan kepada aktor.
	Notasi *controller object* merepresentasikan suatu objek yang mengatur pembuatan atau pembaharuan entitas *object*, penghubung antar *boundary* dan *entity* dan komunikasi antar *object*.



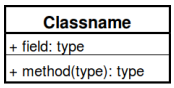
Gambar 2.3 Contoh *sequence diagram* (Pressman, 2010)

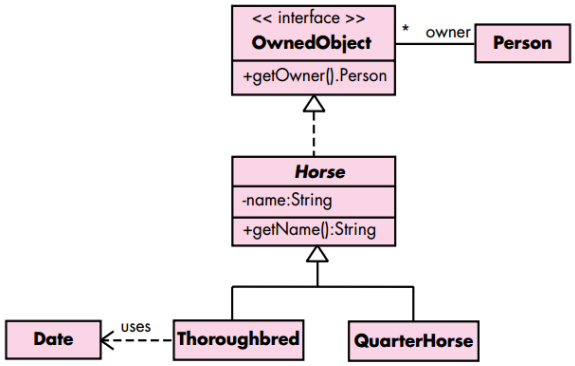
2.3.4.3 *Class Diagram*

Class diagram merupakan sebuah diagram statis atau *structural view* yang memodelkan *class*, *class attribute*, *class operation*, hubungan, dan asosiasi antar *class*. Elemen utama dari *class diagram* adalah kotak yang terbagi menjadi beberapa bagian. Bagian atas merepresentasikan nama *class*, bagian tengah merepresentasikan *attribute*, dan bagian bawah merepresentasikan *operation*. *Attribute* merupakan apa yang sebuah *class* atau objek ketahui, *operation* merupakan apa yang sebuah *class* atau objek dapat lakukan (Pressman, 2010). Notasi dan deskri-

psi dari notasi tersebut dipaparkan pada Tabel 2.3, dan Contoh dari *class diagram* terdapat dalam Gambar 2.4.

Tabel 2.3 Notasi *class diagram*

Notasi	Deskripsi
	Notasi *class* merepresentasikan suatu *class* yang berisikan *attribute* dan *method*.
	Notasi *association* menggambarkan hubungkan antar suatu *class* dengan *class* lainnya.
	Notasi *composition* menggambarkan hubungan kuat antar suatu *class* dengan *class* lain, di mana keduanya saling membutuhkan secara kuat dan saling membutuhkan.
	Notasi *agregation* menggambarkan hubungan lemah antar suatu *class* dengan *class* yang lain, di mana keduanya tidak saling membutuhkan atau keduanya bisa berdiri sendiri.
	Notasi *generalization* merupakan relasi antara *parent class* dan *child class*, di mana *child class* mewarisi dan menambahkan fitur dari *parent class*.



Gambar 2.4 Contoh *class diagram* (Pressman, 2010)

2.3.5 Pengujian Perangkat Lunak

Penggunaan perangkat lunak yang masif tanpa adanya standardisasi membuat *IEEE* mempublikasikan sebuah *issue* pada bulan november hingga desember di tahun 1999. *IEEE* dan *ACM* kemudian membentuk tim gabungan untuk mendefinisikan standardisasi pada proses pengembangan perangkat lunak. Standardisasi tersebut memuat tentang *scientific principles, engineering processes, standards, methods, tools, measurement* dan *best practices*, proses pengujian atau *testing* termasuk di dalamnya (Burnstein, 2006).

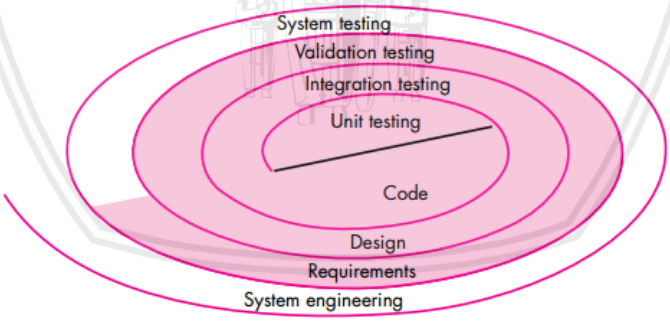
Pengujian bertujuan untuk menemukan cacat dan menguji kualitas suatu perangkat lunak. Dalam proses pengujian terdapat beberapa tahapan, metode, dan teknik. Setiap komponen pada bagian-bagian tersebut memiliki kelemahan dan kekurangan tertentu. Proses testing juga akan menghasilkan artefak-artefak seperti *test-scenario, test-case, test-data* dan *test scripts* (Burnstein, 2006).

2.3.5.1 Tahapan Pengujian

Terdapat beberapa tahapan dalam pengujian. Tahapan-tahapan tersebut diantaranya *unit testing, integration testing*, dan *validation testing* (Presman, 2010). Tahapan awal dalam pengujian adalah tahapan pengujian *unit*. Terdapat beberapa pendapat berbeda mengenai "*unit*" dalam *unit testing*. Fowler (2018) berpendapat bahwa definisi *unit* dapat berarti *single method, single class* maupun kumpulan dari beberapa *class*. Osherove (2015) tidak mengikat definisi *unit* pada *class* maupun *method*, melainkan mengikatnya dengan definsi atribut seperti *performance, reliability* dan *consistency*. *Google* tidak menggunakan istilah *unit, integration* ataupun *system* untuk merujuk pada tingkatan tertentu, melainkan menggunakan istilah *small, medium, large* (Whittaker, Arbon, & Carollo, 2012). Di sisi lain, Presman (2010) berpendapat bahwa fokus pengujian *unit* pada perangkat lunak konvensional adalah sebuah modul dan fokus pengujian *unit* pada perangkat lunak berorientasi objek adalah *class* dengan *testable unit* terkecil adalah *operation* atau *method*. Tahapan selanjutnya adalah *Integration testing*. *Integrartion testing* merupakan pengujian yang dilakukan pada kumpulan *unit* yang telah diuji. Hal ini dilakukan untuk memastikan suatu *unit* akan berjalan dengan baik jika dijalankan bersamaan dengan *unit* yang lain. Meskipun setiap *unit* telah diuji secara individu, terdapat beberapa kemungkinan galat yang terjadi ketika *unit-unit* tersebut diuji secara bersamaan. Beberapa kemungkinan galat yang akan terjadi di antaranya adalah hilangnya data ketika melintasi *interface*, dan komponen yang tidak berjalan semestinya ketika digabungkan. Pada proses pengujian *unit* maupun *integrar-*

tion, komponen tidak berupa *stand-alone program*. Maka dibutuhkan pembuatan *stub* ataupun *driver* selama proses pengujian. *Driver* hanyalah *main program* tiruan yang menjalankan kasus uji, sedangkan *stub* merupakan sebuah *dummy subprogram* yang bekerja menggantikan komponen-komponen pengujian yang asli.

Pada sistem dengan arsitektur berorientasi objek pengujian integrasi dengan pendekatan *incremental top-down integration* maupun *incremental bottom-up integration* tidak dapat digunakan, karena sistem berorientasi objek tidak memiliki kontrol hierarki yang jelas. Maka dibutuhkan pendekatan yang berbeda, yaitu dengan pendekatan integrasi *thread-based testing* dan *use-based testing*. *Thread-based testing* bekerja dengan mengintegrasikan sekumpulan *class* yang dibutuhkan untuk *input* tertentu. Setiap *thread* diuji dan diintegrasikan secara individu. *Use-based testing* berkeja dengan menguji *class* independen yang memiliki ketergantungan paling kecil dengan *class* lainnya. Setelah *class-class* tersebut diuji, lapisan *class* selanjutnya (*dependent classess*) yang menggunakan *independent class* diuji hingga seluruh sistem selesai diuji (Presman, 2010). Tahapan terakhir adalah pengujian validasi atau *validation testing*. Pengujian validasi fokus terhadap *user-visible action* dan *user-recognizable output* dari sistem, atau dengan kata lain pengujian validasi menguji apakah sistem yang dibangun sudah sesuai dengan seluruh skenario kebutuhan yang didefinisikan (Pressman, 2010). Tahapan pengujian terlihat dalam Gambar 2.5.



Gambar 2.5 Tahapan pengujian (Presman, 2010)

2.3.5.2 Metode Pengujian

Metode pengujian atau *point of view* dalam pengujian merupakan sudut pandang seorang *tester* tatkala mendesain sebuah *test case*. Terdapat di antaranya *white-box testing* dan *black-box testing*. Kedua metode tersebut memiliki kekurangan masing-masing. Metode *white-box testing* tidak dapat menemukan kebutuhan yang belum diimplementasikan (Dijkstra, 1970). Hal ini dapat diselesaikan dengan penggunaan metode *black-box testing*. Begitu juga dengan kekurangan

metode *black-box testing* yang dapat menghasilkan *test-case* yang tidak tepat dan tidak dapat menemukan bagian yang belum diuji (Savenkov, 2008). Kelemahan pada metode *black-box testing* ini dapat diselesaikan dengan menggunakan metode *white-box testing*. Oleh karena itu, penggunaan kedua metode dalam pengujian sistem menyelesaikan semua masalah pada masing-masing kekurangan yang dimiliki kedua metode.

White-box testing merupakan suatu metode pengujian di mana proses pengujiannya dilakukan dengan melihat internal perangkat lunak. Maka pada tahapan ini seorang penguji wajib memiliki kemampuan pemrograman. Pengujian ini dilakukan untuk menguji apakah *logic* dan *data* pada perangkat lunak berfungsi sebagaimana mestinya (Myers, Sandler, & Badgett, 2011). Di sisi lain *black-box testing* hanya menguji bagian luar suatu perangkat lunak. Pada tahapan ini fungsionalitas suatu perangkat lunak diuji tanpa harus mengetahui internalnya. Oleh karena itu, tidak dibutuhkan kemampuan pemrograman saat membuat *test case* pada pengujian *black-box*. Penguji hanya mengetahui bagaimana seharusnya perangkat lunak berjalan (*what*), bukan bagaimana perangkat lunak itu melakukan sesuatu (*how*). Pada tahapan ini penguji hanya mempertimbangkan masukan (*input*) dan keluaran (*output*) perangkat lunak selama mendesain *test-case* (Myers, Sandler, & Badgett, 2011).

2.3.5.3 Teknik Pengujian

Terdapat beberapa teknik dalam pengujian. Pengujian semua *path* atau *rigorous testing* pada proses *white-box testing* tidak mungkin dilakukan sehingga digunakan teknik *basis path testing* untuk menentukan *path* yang akan diuji (Gregory, 2007). *Rigorous testing* atau *exhaustive testing* ($C\infty$) atau menguji semua kemungkinan juga tidak mungkin dilakukan pada saat pengujian *black-box* sehingga penggunaan teknik *equivalence partitioning* untuk menentukan masukan yang *valid* dan *invalid* diperlukan. Teknik *boundary value analysis* menjadi pelengkap teknik *equivalence partitioning*. *Boundary value analysis* digunakan untuk mendapatkan nilai masukan yang berada pada *boundary* atau *corner*, karena nilai-nilai pada bagian tersebut memiliki kemungkinan galat yang besar (Presman, 2010). Maka tiga teknik utama dalam pengujian perangkat lunak adalah:

1. *Basis path testing*
2. *Equivalence partitioning*
3. *Boundary value analysis*

Basis path testing merupakan suatu teknik pengujian pada metode *white-box* yang diajukan oleh McCabe pada tahun 1996. Teknik ini menggunakan penelitian McCabe yang sebelumnya yaitu *cyclomatic complexity* yang ia temukan pada tahun 1976. Meski saat ini *cyclomatic complexity* banyak dikaitkan secara erat dengan rumus untuk menentukan jumlah *independent path*. *Cyclomatic complexity* yang awalnya ditemukan pada tahun 1976 diajukan untuk mengukur tingkat kompleksitas suatu modul dalam sebuah program. Modul yang melebihi nilai *cyclomatic complexity* sepuluh direkomendasikan untuk dipisah atau dilakukan *refactoring*. Tujuan kedua dari rumus tersebut adalah untuk melihat tingkat terstrukturnya (*structuredness*) suatu modul. Hal yang membuat *cyclomatic complexity* dikaitkan secara erat dengan proses *basis path testing* karena kemudian McCabe sadar bahwa *cyclomatic complexity* dapat menemukan *independent path* pada suatu modul. *Independent path* adalah suatu jalur dalam sebuah program yang memperkenalkan setidaknya suatu rangkaian pemrosesan baru atau kondisi baru. Hal tersebut yang mengantarnya pada pengajuan *basis path testing* pada tahun 1996. *Basis path testing* memiliki keunggulan karena V(G) atau *vector space* memiliki nilai yang menjadi batas atas *upper bound* untuk membuat *test-case* pada suatu program. Hal ini menguntungkan penguji karena dapat menentukan ukuran batas selesainya suatu pengujian. Selain itu, *basis path testing* memiliki nilai yang lebih baik dari pada *branch coverage*, yaitu *branch coverage ≤ cyclomatic complexity ≤ all of paths* (Gregory, 2007). Langkah-langkah pengujian *basis path* adalah sebagai berikut (Presman, 2010):

1. Membuat *flow graph* dari desain atau *code*.

   Flow graph memiliki beberapa notasi seperti *sequence, if, while, until* dan *case*. Gambar notasi-notasi tersebut terlihat pada Gambar 2.6. Terlihat *flow graph* pada Gambar 2.7 yang dihasilkan dari *pseudocode* 2.1.

2. Menentukan *cyclomatic complexity* dari *flow graph* yang dihasilkan.

   Cyclomatic complexity adalah suatu metrik perangkat lunak yang memberikan ukuran kuantitatif kompleksitas logis dari suatu program. *Cyclomatic complexity* dapat ditentukan dengan menggunakan rumus 2.1:

$$\begin{aligned} V(G) &= jumlah\, region \\ V(G) &= E\text{–}N + 2 \\ V(G) &= P + 1,\ dimana\, P\text{–}predicate\, node \end{aligned} \tag{2.1}$$

Maka Perhitungan *cyclomatic complexity* yang dilakukan pada *flow graph* 2.7 menghasilkan nilai sebagai berikut:

$$
\begin{aligned}
V(G) &= 2\,regions \\
V(G) &= 4\,edges - 4\,nodes + 2 = 2 \\
V(G) &= 1\,predicate\,node + 1 = 2
\end{aligned}
\tag{2.2}
$$

Jadi, *flow graph* pada Gambar 2.7 memiliki nilai *cyclomatic complexity* = 2.

3. Menentukan *independent path*.

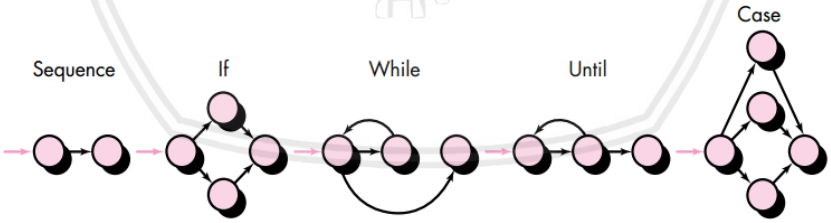
Independent path adalah suatu jalur dalam sebuah program yang memperkenalkan setidaknya suatu rangkaian pemrosesan baru atau kondisi baru. Nilai dari V(G) memberikan batas atas dari banyaknya *independent path* dari sebuah program. Dari contoh *pseudocode procedure hallo* kita mendapatkan 2 jalur *independent* sebagai berikut:

Jalur 1: 1 - 2 - 4

Jalur 2: 1 - 3 - 4

4. Membuat *test case* yang akan menjalankan setiap jalur pada *basis path*.

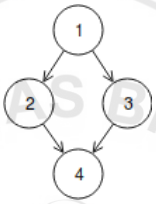
Terdapat dua *test case* yang akan dihasilkan. Pertama *test case* yang harus melewati jalur 1 sehingga pada *test case* pertama nilai *variable nama* harus bernilai *true*. Pada *test case* kedua nilai *variable nama* harus bernilai *false* agar jalur 2 dijalankan. *Test case* yang dihasilkan tampak seperti pada Tabel 2.4.



Gambar 2.6 Notasi *flow graph* (Presman, 2010)

No	hallo
1	`procedure hallo(nama)`
2	`   IF nama == "Budi"  (1)`
3	`      RETURN "Hai" + nama  (2)`
4	`   ELSE`
5	`      RETURN "Nama kosong"  (3)`
6	`   ENDIF  (4)`
7	`end`

Tabel Kode 2.1 Contoh *pseudocode*



Gambar 2.7 Representasi *control flow graph* dari *pseudocode* 2.1

Tabel 2.4 *Test case* dari contoh *pseudocode* 2.1

Jalur	Prosedur Uji	*Expected Result*
1	Memberikan nilai *"Budi"* pada variable *nama*	Program menampilkan "Hai Budi"
2	Tidak memberikan nilai apapun pada variable *nama*	Program menampilkan "Nama kosong"

Equivalence partitioning merupakan suatu teknik dalam metode pengujian *black-box* di mana prosesnya adalah membagi masukan kepada kelas-kelas data. Dari kelas-kelas tersebut *test case* nantinya didapatkan. *Equivalence class* merepresentasikan keadaan valid atau tidak validnya suatu masukan. Beberapa kondisi masukan di antaranya *numeric value, range of values, set of related values,* atau *boolean condition*. *Equivalence partitioning* dapat di definisikan sesuai dengan panduan berikut (Presman, 2010):

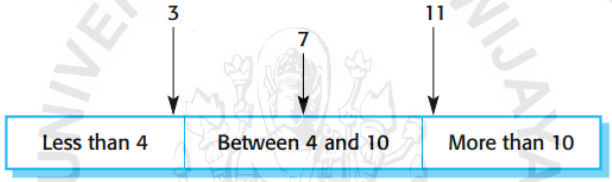
1. Jika kondisi masukan adalah *range*. Maka satu valid dan dua invalid *equivalence class* di definisikan.

2. Jika kondisi masukan adalah *specific value*. Maka satu valid dan dua invalid *equivalence class* di definisikan.

3. Jika kondisi masukan adalah *member of a set*. Maka satu valid dan satu invalid *equivalence class* di definisikan.

4. Jika kondisi masukan adalah *boolean*. Maka satu valid dan satu invalid *equivalence class* di definisikan.

Jika data masukan bertipe *range* dan masukan yang valid adalah *range* bernilai 4 hingga 10. Maka masukan yang tidak valid adalah semua angka yang kurang dari 4 dan semua angka yang lebih dari 10 seperti terlihat pada Gambar 2.8. Data yang digunakan untuk *test case* pada tipe masukan *range* berjumlah 3. Satu bernilai valid dan dua bernilai invalid. Oleh karena itu, data masukan *valid* yang kita gunakan adalah 11 dan 7. Sedangkan data *invalid* yang digunakan adalah 3.



Gambar 2.8 Contoh *equivalence partitioning* (Sommerville, 2014)

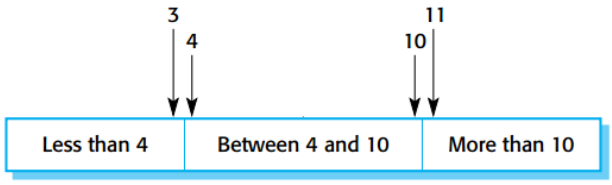
Boundary value analysis merupakan suatu teknik yang melengkapi *equivalence partitioning*. *Boundary value analysis* dikembangkan untuk menganalisis nilai yang berada pada *boundary input*, karena nilai pada *boundary* atau *corner* memiliki peluang galat yang lebih tinggi. *Boundary value analysis* tidak berdiri sendiri, melainkan merupakan teknik yang melengkapi *equivalence partitioning*. Maka *boundary value analysis* mengambil nilai yang bersifat "pojok" dari nilai hasil *equivalence partitioning*. Panduan pemilihan nilai *boundary value analysis* di antaranya sebagai berikut (Presman, 2010):

1. Jika masukan merupakan sebuah *range* dari a hingga b. Maka nilai yang digunakan untuk *test case* adalah satu nilai di atas a dan satu nilai di bawah b.

2. Jika masukan merupakan sebuah *number of values*. Maka nilai yang digunakan untuk *test case* adalah nilai maksimum dan minimum serta satu nilai di atas maksimum dan satu nilai di bawah minimum.

Jika data masukan bertipe *range* dari 4 hingga 10. Maka nilai yang didapatkan untuk pengujian adalah nilai maksimum dan minimum serta satu nilai di

atas maksimum dan satu nilai di atas minimum. Jadi nilai yang didapatkan adalah 4, 10, 3, dan 11 seperti terlihat pada Gambar 2.9.



Gambar 2.9 Contoh *boundary value analysis* (Sommerville, 2014)

2.3.5.4 *Automated Testing*

Menjalankan semua *test case* secara manual sangat menyita waktu. Bahkan mereka dapat menyita 50% dari pada waktu pengembangan (Brooks Jr, 1995). *Automated testing* dapat digunakan untuk mempercepat proses pengujian. *Automated testing* adalah proses menjalakan *test case* secara otomatis. *Automated testing* berjalan dengan adanya *trigger*, baik dalam bentuk *event* maupun *time*. Prosesnya adalah dengan menjalankan *test case* dan membandingkannya dengan *output* yang sudah ditentukan. Hasil perbandingan tersebut akan dilaporkan kepada penguji secara otomatis setelah *automated testing* selesai dijalankan (Nshimiyiman, 2018). *Automated testing* dibangun dengan menggunakan *script* sesuai dengan *test case* yang didapatkan dari tahapan pengujian *white-box* maupun *black-box*. Tampak pada Tabel Kode 2.2 merupakan *script* yang dibangun untuk melakukan pengujian secara otomatis. *Script* tersebut didapatkan dari *test case* untuk pengujian *pseudocode hallo* pada Tabel Kode 2.4. *Script* ini nantinya dijalankan dengan kakas bantu *Pytest* untuk melakukan pengecekan berhasil atau gagalnya suatu pengujian secara otomatis. Sedangkan *trigger* untuk menjalankan *Pytest* secara otomatis dilakukan dengan bantuan kakas bantu *Gitlab-CI*.

No	test_hallo
1	`def test_case_1():`
2	`    assert hallo("Budi") == "Hai Budi"`
3	
4	`def test_case_2():`
5	`    assert hallo("Ani") == "Nama Kosong"`

Tabel Kode 2.2 Contoh *script atomated testing*

2.3.5.5 Pengujian *Compatibility*

Pengujian *compatibility* adalah pengujian yang dilakukan pada sistem yang dibangun di atas beberapa media yang berbeda, seperti di atas *display devices, operating systems* dan *browser* yang berbeda-beda. Masalah *compatibility* yang kecil tidak memberikan dampak signifikan pada sistem, tetapi masalah yang besar dapat mengakibatkan dampak besar (Pressman, 2010). Terdapat berbagai macam *desktop environment* yang berbeda-beda pada sistem operasi *GNU/Linux*. Setiap *desktop environment* memiliki format nama *window* tersendiri. Sebuah sistem harus dapat menangani perbedaan format nama *window* agar dapat *compatible* antar *desktop environment*. Pengujian *compatibility* terhadap perbedaan format nama *window* antar *desktop environment* dilakukan dengan menjalankan sistem pada berbagai *desktop environment* yang berbeda. Pengujian berhasil jika sistem dapat menangani perbedaan format-format tersebut. Beberapa format nama *window* pada *desktop environment* di antaranya:

- "_NET_ACTIVE_WINDOW(WINDOW): window id # 0x3c00009"
- "_NET_ACTIVE_WINDOW(WINDOW): window id # 0x4a0000c, 0x0"

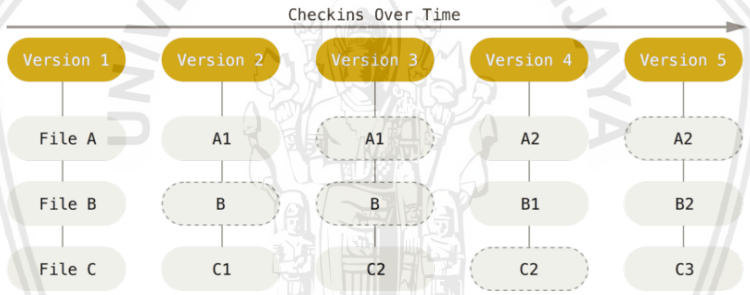
2.4 *Snapshotting*

Snapshot merupakan proses untuk merekam suatu keadaan media penyimpanan dalam suatu waktu tertentu. *Snapshot* yang dihasilkan dapat dikembalikan kapan pun pengguna butuhkan. Umumnya *snapshot* dapat dihasilkan dari beberapa kakas bantu seperti *data protection software, data analisys*, dan *replication application*. Implementasi untuk melakukan *snapshot* berbeda-beda, tergantung pada kebutuhan aplikasi(Garimella, 2018). *Version Control Software* (*VCS*) merupakan salah satu kakas bantu yang dapat digunakan untuk membuat *snapshot*. *VCS* lahir dimulai dengan adanya *Source Code Control System (SCCS)* pada tahun 1972 yang dibangun oleh Marc J. Rochkind, disusul kemunculan *Revision Control System (RCS)* pada tahun 1980, *Concurrent Versions System (CVS)* pada tahun 1985, kemudian *subversion* dan *git* pada tahun-tahun berikutnya (Ruparelia, 2010)

Git merupakan salah satu *version control software* yang digunakan secara umum. Hal ini disebabkan karena *git* memiliki sifat *distributed* sehingga memiliki kelebihan yang tidak dimiliki *version control* terpusat, salah satunya adalah kemampuan untuk melakukan seluruh operasi secara lokal (Ruparelia, 2010). *Snapshot* pada *git* disimpan di dalam *repository* dan digunakan untuk menyimpan suatu berkas dalam keadaan tertentu. Proses pembuatan *snapshot* dapat dilakukan

dengan meminta *git* untuk menambahkan semua berkas yang ada menggunakan perintah "*add*". Proses selanjutnya dilakukan dengan menyimpan *snapshot* secara permanen dengan menggunakan perintah "*commit*". Proses ini akan menyimpan *snapshot* secara permanen pada *database git*. Perintah lainnya yang dapat kita gunakan adalah "*log*" untuk melihat *history* dari *snapshot*, dan "*diff*" untuk melihat perbedaan antar *snapshot* (Chacon & Straub, 2014).

Umumnya *version control* beroperasi dengan menyimpan *diff* antar berkas pada suatu saat tertentu, cara ini biasa disebut dengan *delta-based version control*. Hal ini berbeda dengan *git* yang menyimpan keseluruhan *snapshot* pada suatu waktu tertentu. Setiap "*commit*" akan menyimpan seluruh keadaan berkas, kecuali berkas yang tidak berubah. *Git* melihat *project* layaknya *stream of snapshots*. Semua *snapshot* tersebut memiliki integritas yang berupa 40 karakter *hexadecimal* dihasilkan dari kalkulasi berkas pada suatu *snapshot* dengan algoritme *SHA-1*. Dengan *hash* tersebut, *git* dapat merujuk suatu *snapshot* tertentu (Chacon & Straub, 2014).



Gambar 2.10 Penyimpanan data sebagai *snapshot* dari waktu ke waktu (Chacon & Straub, 2014)

2.5 *User Activity Logging*

Log merupakan sebuah berkas yang memuat rekaman semua aktivitas yang terjadi pada perangkat lunak yang sedang berjalan. Pada umumnya *log* memiliki format berupa *plain text* dan memuat informasi berupa waktu, kategori dan deskripsi. *Log* secara umum digunakan pada *web server, operating system* dan berbagai macam perangkat lunak lain. Merekam aktivitas perangkat lunak maupun pengguna dan menyimpannya ke dalam *log* bertujuan untuk melakukan *debugging, monitoring, optimization, forensic analysis* dan beberapa tujuan lain. *Log* memiliki beberapa macam tipe, di antaranya *event logs, message logs* dan *transaction log* (DeLaRosa, 2018).

User activity logging adalah sebuah proses menyimpan aktivitas pengguna ke dalam sebuah *log*. *User activity* merupakan aktivitas pengguna dalam menggunakan perangkat lunak. *User activity* diantaranya adalah *keystrokes*, gerakan *mouse*, suara, gerakan, gerakan mata, tempat yang dikunjungi, berkas yang dibuka, dan aplikasi yang dijalankan (Macbeth, et al., 2007).

2.6 *Edit-distance Algorithm*

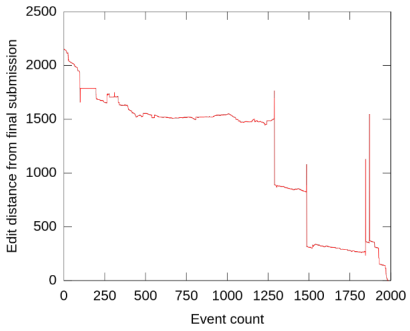
Edit-distance merupakan algoritme yang melakukan perhitungan pada penghapusan, sisipan, maupun substitusi antara dua buah kata. Algoritme ini lebih umum dikenal dengan *edit distance*, tetapi umumnya *edit distance* merujuk pada *levenshtein distance*. Algoritme ini ditemukan oleh matematikawan rusia, Vladimir Levenshteinn pada 1965. Pada algoritme ini semua karakter yang ada, memiliki *unit cost* (harga yang nantinya dikalkulasi) kecuali substitusi yang dilakukan pada karakter itu sendiri. Secara umum algoritme ini memiliki nilai yang sama dengan nilai minimal operasi yang diperlukan untuk mengubah karakter "a" menjadi "b" (Navarro, 2001). Secara formal dapat ditulis sebagai:

$$^{w}ins^{(x)}, ^{w}del^{(x)}, ^{w}sub^{(x,y)} \tag{2.3}$$

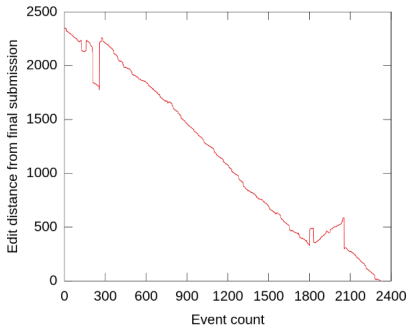
Contoh perhitungan dari algoritme ini, seperti perubahan dari kata "tekkom" menjadi "filkom", yang memiliki nilai *edit-distance* berjumlah 3.

1. tekkom $\rightarrow$ fekkom (penggantian dari "t" ke "f").
2. fekkom $\rightarrow$ fikkom (penggantian dari "e" ke "i").
3. fikkom $\rightarrow$ filkom (penggantian dari "k" ke "l").

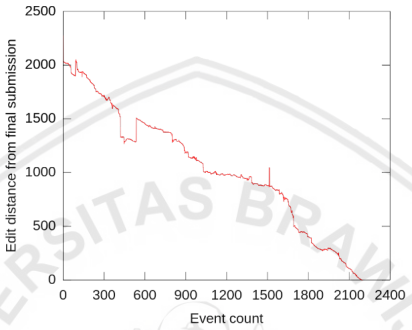
Algoritme *edit-distance* nantinya dapat digunakan untuk mengukur kemurnian proses pengerjaan tugas. Grafik akan dikalkulasi dari *snapshot* satu ke yang lain. Tampak pada Gambar 2.11a menunjukkan siswa yang mengerjakan tugasnya sendiri kemudian mengalami kesulitan, pada akhirnya ia meminta jawaban temannya kemudian ditempelkan pada tugasnya sendiri. Gambar 2.11b menunjukkan siswa yang meminta jawaban teman sejak awal, kemudian meniru jawaban tersebut tanpa melakukan salin-tempel, tetapi melakukan beberapa modifikasi di akhir. Gambar 2.11c menunjukkan siswa yang mengerjakan tugasnya sendiri dan terlihat tidak adanya proses salin-tempel selama pengerjaan tugas (Hellas, Leinonen, & Ihantola, 2017).



(a) Kasus A



(b) Kasus B



(c) Kasus C

Gambar 2.11 Grafik *edit-distance*

2.7 Teknologi Pengembangan Perangkat Lunak

2.7.1 *Python*

Python adalah bahasa pemrograman dengan *syntax* yang elegan dan sederhana yang dikembangkan oleh Guido van Rossum pada tahun 1990. *Python* bersifat *interpreted* dan mendukung paradigma *object-oriented*. *Python* juga memiliki dukungan struktur data tingkat tinggi seperti *dictionaries* dan *list*. Berbeda dengan bahasa *scripting* yang lain, *Python* dapat digunakan secara gratis, bahkan untuk tujuan komersial (Sanner, et al., 1999).

2.7.2 *Git (Distributed Version Control)*

Distributed version control merupakan sebuah konsep di mana aplikasi *version control* dapat merekam semua perubahan yang terjadi pada sebuah berkas secara terdistribusi atau tidak terpusat. Sifatnya yang tidak terpusat memiliki kelebihan untuk melakukan semua operasi secara lokal. Berbeda dengan *version control* lainnya yang menggunakan *diff* untuk menyimpan berkas dari suatu waktu ke waktu, *git* menggunakan model *snapshot* atau merekam seluruh berkas pada suatu

waktu tertentu. Hal ini memiliki kelebihan seperti kecepatan dalam konstruksi berkas yang diminta pada waktu tertentu. Pada mulanya *git* dikembangkan oleh Linus Torvalds pada tahun 2005, Linus kemudian menunjuk Junio Hamano untuk menjadi *maintainer* hingga sekarang (Chacon & Straub, 2014). *Git* sejak awal diciptakan secara modern sehingga memiliki integritas menggunakan *hash SHA1* yang berupa 40 karakter heksadesimal. Hal ini yang dijelaskan oleh Linus Torvalds sebagai *content adressable* layaknya *filesystem* (Torvalds, 2018). Tabel Kode 2.3 menunjukkan perintah pada *git* untuk menyimpan perubahan suatu berkas.

No	commit
1	`git commit -m "saved"`

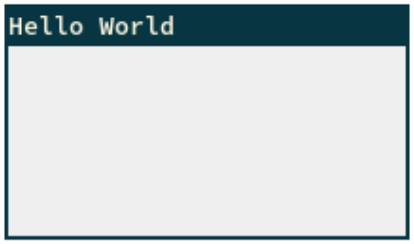
Tabel Kode 2.3 Perintah untuk menyimpan perubahan pada *git*

2.7.3 *Qt (Widget Toolkit)*

Qt merupakan pustaka yang digunakan untuk membangun *graphical user interface*. Awalnya *Qt* dikembangan oleh Haavard Nord dan Eirik Chambe-Eng. Pada saat ini *Qt* dikembangan oleh *The Qt Company* (*Qt History* 2018). *Qt* memiliki beberapa lisensi, di antaranya lisensi komersial dan lisensi *GPL 2.0*. Modul-modul yang dimiliki *Qt* di antaranya adalah *Qt Core, Qt GUI,* dan *Qt Widgets* (*Qt About* 2018). Secara umum *Qt* digunakan untuk membangun *graphical user interface* agar memudahkan interaksi pengguna dengan aplikasi. Terlihat pada gambar 2.12 *Qt* digunakan untuk membuat *window* sederhana dengan menggunakan kode pada Tabel Kode 2.4.

No	hello
1	`import sys`
2	`from PyQt5.QtWidgets import QApplication, QWidget`
3	
4	`if __name__ == '__main__':`
5	`    app = QApplication(sys.argv)`
6	`    window = QWidget()`
7	`    window.resize(250, 150)`
8	`    window.setWindowTitle('Hello World')`
9	`    window.show()`
10	`    sys.exit(app.exec_())`

Tabel Kode 2.4 Kode untuk membuat *window* sederhana



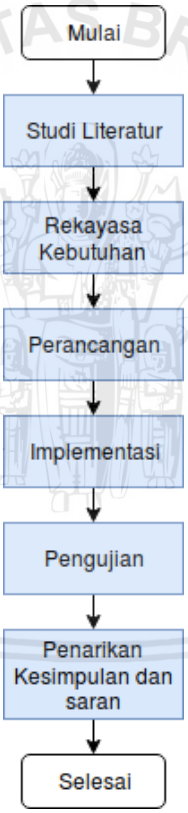
Gambar 2.12 *Window* sederhana dengan *Qt*

2.7.4 *Draw.io*

Draw.io merupakan sebuah kakas bantu untuk menggambar diagram ataupun *mock up*. *Draw.io* memiliki lisensi perangkat lunak bebas (*Apache v2*) dan dapat diakses melalui peramban secara daring. *Draw.io* dibangun oleh Gaudenz Alder menggunakan bahasa pemrograman *javascript* pada tahun 2000. Peramban yang didukung diantaranya *IE 11, Chrome 32+, Firefox 38+, Safari 9.1.x,* dan *Opera 20+* (*About draw.io* 2019).

BAB 3 METODOLOGI

Pada bab ini akan dijelaskan langkah-langkah penelitian. Langkah pertama diawali dengan studi literatur. Langkah selanjutnya diikuti dengan rekayasa kebutuhan, perancangan, implementasi dan pengujian yang merupakan urutan tahapan pada pengembangan sistem yang menggunakan model *waterfall*. Pemilihan model *waterfall* dalam pengembangan sistem pada penelitian ini disebabkan kebutuhan sistem yang sudah dapat didefinisikan dengan baik pada tahap awal pengembangan sistem. Pengujian merupakan langkah terakhir dalam pengembangan sistem pada penelitian ini, hal ini sesuai dengan rumusan masalah penelitian. Langkah terakhir penelitian ditutup dengan penarikan kesimpulan dan saran. Langkah-langkah tersebut dapat dilihat pada pada Gambar 3.13.



Gambar 3.13 Diagram alir metodologi

3.1 Studi Literatur

Studi literatur dibutuhkan untuk mendalami teori-teori dan urgensi tentang plagiarisme lebih dalam. Selain itu, studi literatur juga dibutuhkan untuk mendalami teori rekayasa perangkat lunak untuk mengembangkan perangkat lunak pence-

gah plagiarisme dengan menggunakan metode *snapshotting* dan *user activity logging*. Sumber dari literatur tersebut berasal dari jurnal, buku, situs resmi dan dokumentasi perangkat lunak yang digunakan. Daftar literatur yang dipelajari terkait dengan:

1. Penelitian-penelitian sebelumnya yang terkait dengan plagiarisme
2. Metode *user activity logging, snapshotting* dan *edit-distance algorithm*
3. Teori pengembangan perangkat lunak, yang di dalamnya meliputi teori rekayasa kebutuhan, perancangan, implementasi, dan pengujian
4. Teknologi yang digunakan dalam penelitian

3.2 Rekayasa Kebutuhan

Rekayasa kebutuhan merupakan kumpulan tugas dan teknik untuk memahami kebutuhan. Terdapat empat tahapan dalam rekayasa kebutuhan, yaitu analisis kebutuhan, spesifikasi kebutuhan, manajemen kebutuhan, dan pemodelan kebutuhan. Penjelasan tentang tahapan-tahapan yang dilakukan dalam rekayasa kebutuhan adalah sebagai berikut:

1. Analisis kebutuhan

Pada tahapan ini dilakukan penggalian kebutuhan sistem. Kebutuhan fungsional sistem didapatkan dari:

(a) Metode yang telah digunakan pada penelitian (Hellas, Leinonen, & Ihantola, 2017).

(b) "*Future work*" penelitian Hellas, Leinonen, & Ihantola (2017), dan penelitian Leung & Cheng (2017) yang memaparkan cara-cara plagiarisme di mana cara-cara tersebut belum dapat ditangani oleh penelitian Hellas, Leinonen, & Ihantola (2017).

(c) Kelemahan dari penelitian Hellas, Leinonen, & Ihantola (2017) yang akan disempurnakan.

Pada sumber kebutuhan fungsional pertama didapatkan kebutuhan fungsional untuk merekam waktu mulai pengerjaan, waktu selesai pengerjaan, rekaman alamat *IP*, dan menghitung nilai *edit-distance* pada berkas tugas. Sumber kebutuhan fungsional kedua dan ketiga menghasilkan kebutuhan fungsional untuk merekam

perubahan berkas tugas dengan metode *snapshotting*, merekam aktivitas siswa menggunakan metode *user activity logging*, dan mengembangkan sistem dengan sifat *agnostic* agar tidak terikat pada *platform* tertentu. Komponen aktivitas siswa yang direkam adalah seluruh *window*, *window* yang sedang aktif, nama *login*, dan nama *device*.

Kebutuhan non-fungsional didapatkan dari batasan sistem yang hanya dapat berjalan pada sistem operasi *GNU/Linux*, terdapat berbagai macam *desktop environment* pada sistem operasi *GNU/Linux* sehingga sistem yang dibangun harus *compatible* dengan enam *desktop environment* yang umum digunakan. Keenam *desktop environment* tersebut adalah *GNOME, KDE, XFCE, Unity, Cinnamon* dan *Phanteon*.

2. Spesifikasi kebutuhan

Pada tahapan ini hasil dari analisis kebutuhan pada tahapan sebelumnya dijelaskan secara lebih mendetail dengan jelas, tidak ambigu, mudah dipahami, dan konsisten. Kebutuhan fungsional mendeskripsikan apa yang dapat dilakukan oleh sistem yang dibangun dan kebutuhan non-fungsional mendeskripsikan batasan atau kualitas dari sistem yang dibangun.

3. Manajemen kebutuhan

Tahapan ini dilakukan untuk memudahkan dalam mengidentifikasi, mengontrol dan melacak kebutuhan. Maka pada tahapan ini dilakukan kodefikasi pada kebutuhan-kebutuhan yang ada. Kode LUP-F-1 dan LUPR-F-1 untuk kebutuhan fungsional, LUPR-NF-1 dan LUPR-NR-1 untuk kebutuhan non-fungsional.

4. Pemodelan kebutuhan

Tahap terakhir adalah pemodelan kebutuhan. Untuk menggambarkan fungsionalitas dan batasan sistem secara visual, penelitian ini menggunakan alat bantu *use case diagram*, sedangkan untuk menjelaskan skenario interaksi antar aktor dan sistem secara mendetail, penelitian ini menggunakan alat bantu *use case scenario*. Penggunaan kedua diagram tersebut dalam tahapan ini dapat memodelkan kebutuhan sistem baik secara visual (*use case diagram*) maupun secara tekstual dengan detail (*use case scenario*).

3.3 Perancangan

Tahapan perancangan sistem menjelaskan tentang rancangan sistem yang akan dibangun berdasarkan hasil rekayasa kebutuhan pada tahapan sebelumnya.

Rancangan ini nantinya akan menjadi acuan pada tahap implementasi dan pengujian. Tahap-tahap yang dilakukan pada perancangan sistem adalah:

1. Perancangan arsitektur

Sistem akan dibangun dengan arsitektur *Model-View-Controller* (*MVC*). Arsitektur *MVC* dipilih agar komponen fungsionalitas sistem terpisah dari komponen antarmuka sistem. Hal ini memudahkan pengembangan selanjutnya, karena pengembang cukup fokus pada masing-masing komponen yang terpisah. Arsitektur ini juga memudahkan proses *porting* maupun penggantian antarmuka, karena pengembang cukup fokus pada komponen antarmuka sistem. Untuk menggambarkan interaksi yang terjadi antar komponen pada sistem, penelitian ini menggunakan alat bantu *sequence digaram*. Terdapat tiga sampel *sequence diagram* yang akan dipaparkan pada tahapan ini. Untuk menggambarkan struktur sistem dan relasi antar komponen pada sistem, penelitian ini menggunakan alat bantu *class diagram*.

2. Perancangan komponen

Pada tahapan ini komponen sistem akan dirancang menggunakan *pseudocode*. *Pseudocode* inilah yang nantinya menjadi acuan pada tahapan implementasi. Pada tahapan ini terdapat tiga sampel perancangan komponen yang akan dipaparkan.

3. Perancangan antarmuka

Pada tahapan ini rancangan antarmuka sistem dan tata letak komponen antarmuka di dalamnya digambarkan dengan gambaran sederhana. Perancangan antarmuka dilakukan dengan kakas bantu *draw.io*. Terdapat tiga sampel perancangan antarmuka yang akan dipaparkan pada tahapan ini.

3.4 Implementasi

Pada tahapan implementasi, hasil perancangan pada tahapan sebelumnya diimplementasikan, terdapat dua tahapan dalam implementasi yaitu:

1. Implementasi kode program

Pada tahapan ini hasil rancangan komponen sistem yang berupa *pseudocode* diimplementasikan menjadi *working code* menggunakan bahasa pemrograman *Python*. Selain menggunakan *Python*, metode *snapshotting* yang diimplementasikan akan menggunakan bantuan kakas bantu *git*.

2. Implementasi antarmuka

Pada tahapan ini hasil rancangan antarmuka sistem diimplementasikan menggunakan *widget toolkit Qt*.

3.5 Pengujian

Pada tahapan pengujian, dilakukan pengujian terhadap sistem yang telah selesai diimplementasikan. Tahapan-tahapan yang akan dilakukan adalah:

1. Pengujian unit

Tahap pengujian unit dilakukan untuk memastikan hasil implementasi kode program sesuai dengan hasil perancangan komponen. Penelitian ini mengikuti pendapat Pressman (2010) terkait dengan pemahaman *unit*. Maka tahapan ini menguji setiap unit terkecil dari sistem yaitu sebuah *class*. *Class controller* dan *model* diuji hingga batas *code coverage* mencapai 100%, karena semua kode inti program terdapat pada kedua *class* tersebut. Berbeda dengan *class view* hanya memanggil fungsi-fungsi yang terdapat pada *class controller* dan *model* sehingga pada penelitian ini pengujian unit pada *class view* hanya diuji hingga *code coverage* mencapai 70%. Pengujian unit dilakukan menggunakan metode *white-box testing* dan kasus uji didapatkan dengan teknik *basis path testing*. Pengujian setiap unit pada tahapan ini dilakukan secara terisolasi sehingga dilakukan pembuatan *stub* atau "*dummy subprogram*". Terdapat tiga sampel pengujian unit yang akan dipaparkan pada tahapan ini.

2. Pengujian integrasi

Setelah semua unit selesai diuji, maka dilakukan tahapan pengujian integrasi. Pengujian integrasi dilakukan untuk memastikan integrasi antar unit berjalan dengan baik. Maka pada pengujian integrasi, *function* pada suatu *class* yang memanggil *function* pada *class* lain tidak digantikan dengan *stub* atau *fake* sebagaimana ketika pada pengujian unit. Pada penelitian ini perangkat lunak yang dikembangkan menggunakan pendekatan dan pengembangan berorientasi objek sehingga pendekatan pengujian integrasi yang digunakan adalah *use-based testing*. Integrasi diawali dari *class* yang memiliki *dependent class* paling sedikit, yaitu dimulai dari *class model*, kemudian *class controller*, dan yang terakhir adalah *class view*. Pengujian integrasi menggunakan metode *white-box testing* dan teknik *basis path testing*. Terdapat satu sampel yang dipaparkan pada tahapan ini.

3. Pengujian validasi

Pengujian validasi menguji apakah sistem yang dibangun sudah sesuai dengan kebutuhan yang didefinisikan. Pengujian validasi fokus terhadap *user-visible action* dan *user-recognizable output* dari sistem (Pressman, 2010). Maka pada tahapan ini seluruh skenario kebutuhan diuji. Pengujian validasi dilakukan menggunakan metode *black-box testing*. Teknik yang digunakan untuk mendapatkan kasus uji dan data uji adalah teknik *equivalence partitioning* dan *boundary value analysis*. Kedua teknik tersebut digunakan untuk menghindari *rigorous testing* atau mencoba semua kemungkinan.

4. *Automated testing*

Automated testing dilakukan untuk meminimalisir waktu yang digunakan untuk pengujian. Pada tahapan ini dibangun *script* yang menjalankan kasus uji secara otomatis. Kasus uji didapatkan dari pengujian unit dan pengujian integrasi yang telah dilakukan pada tahapan sebelumnya. *Automated testing* berjalan secara otomatis jika terjadi perubahan pada berkas *source code* sistem. Penelitian ini menggunakan kakas bantu *Pytest* untuk membangun *script*, dan menggunakan *Gitlab-CI* untuk menjalankan *script* secara otomatis. Terdapat tiga sampel *automated testing* yang dipaparkan pada tahapan ini.

5. Pengujian *compatibility*

Pengujian *compatibility* merupakan pengujian terhadap kebutuhan non-fungsional sistem. Pada pengujian ini sistem *Lup Recorder* dan *Lup Viewer* dijalankan pada enam *desktop environment* yang umum digunakan pada sistem operasi *GNU/Linux*. Keenam *desktop environment* tersebut adalah *GNOME, KDE, XFCE, Unity, Cinnamon* dan *Phanteon*. Pengujian ini dilakukan untuk memastikan apakah sistem yang dibangun sudah *compatible* dengan berbagai macam *desktop environment*.

3.6 Penarikan Kesimpulan

Kesimpulan akan ditarik pada akhir penelitian. Penarikan kesimpulan dapat dilakukan setelah tahap analisis kebutuhan, perancangan, implementasi, dan pengujian selesai dilakukan. Kesimpulan akan ditarik berdasarkan dari analisis hasil sistem yang telah dibangun. Kesimpulan tersebut akan menjawab rumusan masalah yang telah dipaparkan dan memuat catatan untuk pengembangan sistem di waktu mendatang.

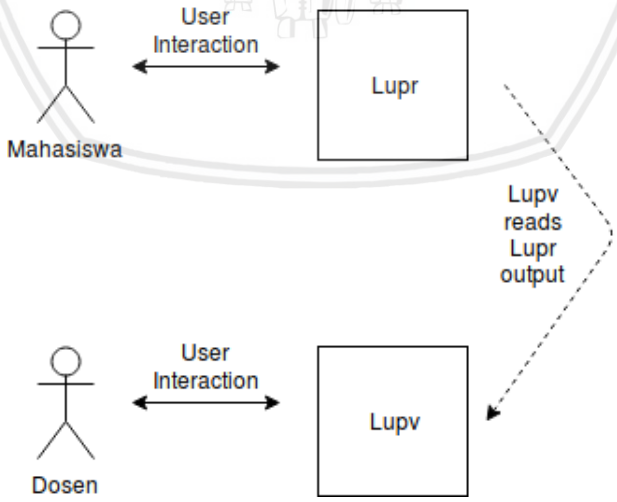
BAB 4 REKAYASA KEBUTUHAN

4.1 Deskripsi Umum Sistem

Sistem yang dikembangkan pada penelitian ini adalah sistem pencegah plagiarisme dengan menggunakan metode *snapshotting* dan *user activity logging*. Sistem ini dibangun dengan tujuan untuk meminimalisir adanya tindak plagiarisme. Sistem ini dibagi menjadi dua bagian, pertama yaitu *Lupv* (*Lup Viewer*) yang digunakan oleh dosen untuk membaca hasil rekaman yang dilakukan oleh sistem kedua. *Lupr* hanya dapat membaca berkas tugas yang berbentuk *plain text format*. Sistem kedua yaitu *Lupr* (*Lup Recorder*), sistem ini digunakan untuk merekam perubahan tugas dan aktivitas siswa selama pengerjaan tugas berlangsung. *Lupv* digunakan oleh dosen sedangkan *Lupr* digunakan oleh mahasiswa.

Lupv memiliki beberapa fungsionalitas, di antaranya menampilkan seluruh daftar rekaman, melihat aktivitas yang dilakukan selama pengerjaan, dan melihat perubahan berkas tugas dari waktu ke waktu. Di lain sisi, *Lupr* memiliki fungsionalitas di antaranya adalah merekam perubahan berkas tugas, alamat *IP*, *login name*, *device name*, seluruh *window* dan *window* yang aktif.

Cara kerja sistem ini dimulai dengan mahasiswa yang menjalankan *Lupr* pada berkas tugas yang diberikan dosen. Setelah tugas selesai dikerjakan, maka berkas tersebut kemudian dikirimkan ke dosen yang bersangkutan. Dosen yang bersangkutan kemudian dapat melihat seluruh hasil rekaman menggunakan *Lupv*.



Gambar 4.14 Gambaran umum sistem

4.2 Analisis Kebutuhan

Analisis kebutuhan pada sistem ini didapatkan dari beberapa sumber. Sumber pertama adalah metode penelitian yang telah dilakukan oleh Hellas, Leinonen, & Ihantola (2017). Sumber kedua didapatkan dari "*future work*" penelitian Hellas, Leinonen, & Ihantola (2017) dan penelitian Leung & Cheng (2017). Penelitian Leung & Cheng (2017) memaparkan cara-cara plagiarisme di mana cara-cara tersebut belum dapat ditangani oleh metode penelitian yang telah dilakukan Hellas, Leinonen, & Ihantola (2017). Sumber kebutuhan ketiga adalah kelemahan dari penelitian Hellas, Leinonen, & Ihantola (2017) yang akan disempurnakan. Ketiga sumber tersebut menghasilkan kebutuhan-kebutuhan pokok sistem yaitu mengimplementasikan metode *snapshotting* untuk merekam perubahan berkas, metode *user activity logging* untuk merekam aktivitas siswa saat mengerjakan tugas, serta membangun sistem dengan sifat *agnostic* agar tidak terikat dengan *platform* tertentu.

Sistem yang dikembangkan memiliki batasan hanya dapat berjalan pada sistem operasi *GNU/Linux*. Terdapat banyak *desktop environment* yang ada pada sistem operasi *GNU/Linux*. Maka sistem harus dapat berjalan dengan baik pada berbagai macam *desktop enviroment* yang memiliki format nama *window* yang berbeda-beda. Oleh karena itu, kebutuhan non-fungsional yang harus dimiliki oleh sistem adalah *compatibility* sistem pada berbagai macam *desktop environments*. Jumlah *desktop environment* yang tidak terbatas membuat penelitian ini membatasi jumlah *desktop environment* yang didukung. *Desktop environment* yang didukung terbatas pada enam *desktop environment* yang umum digunakan, yaitu *GNOME, KDE, XFCE, Unity, Cinnamon* dan *Phanteon*.

4.3 Identifikasi Aktor

Berdasarkan proses analisis kebutuhan yang dilakukan, terdapat satu aktor utama pada sistem *Lup Viewer* yaitu dosen, dan satu aktor utama pada sistem *Lup Recorder* yaitu mahasiswa. Tabel 4.5 memaparkan daftar aktor yang terlibat dalam sistem *Lup Viewer* beserta deskripsinya, dan Tabel 4.6 memaparkan daftar aktor yang terlibat pada sistem *Lup Recorder* beserta deskripsinya.

Tabel 4.5 Identifikasi aktor pada sistem *Lup Viewer*

No	Aktor	Deskirpsi
1	Dosen	Dosen merupakan suatu individu atau kelompok yang berperan dalam memberikan tugas dan menganalisis hasil tugas yang telah dikerjakan oleh mahasiswa.

Tabel 4.6 Identifikasi aktor pada sistem *Lup Recorder*

No	Aktor	Deskirpsi
1	Mahasiswa	Mahasiswa merupakan suatu individu atau kelompok yang berperan dalam mengerjakan tugas yang diberikan oleh dosen.

4.4 Kebutuhan Fungsional Sistem

Kebutuhan fungsional merupakan sebuah kebutuhan yang harus disediakan oleh sistem, bagaimana sistem bereaksi terhadap suatu masukan, dan bagaimana sistem berperilaku pada suatu keadaan (Sommerville, 2014). Daftar kebutuhan fungsional didapatkan dari proses analisis kebutuhan yang telah dilakukan sebelumnya. Dalam penelitian ini, kebutuhan fungsional memiliki sistem penomoran LUPV-F-XX-YY untuk sistem *Lup Viewer* dan LUPR-F-XX-YY untuk sistem *Lup Recorder*. *LUPV* dan *LUPR* merupakan nama sistem, F menunjukkan tipe kebutuhan yaitu kebutuhan fungsional, XX menunjukkan nomor kebutuhan, dan YY menunjukkan nomor spesifikasi kebutuhan. Tabel 4.7 dan Tabel 4.8 memaparkan daftar kebutuhan fungsional, beserta definisi, aktor yang terlibat, dan pemetaan nama *use case* yang didapatkan dari hasil analisis kebutuhan untuk sistem *Lup Recorder* dan *Lup Viewer*.

Tabel 4.7 Spesifikasi kebutuhan fungsional, definisi, dan pemetaan nama *use case* sistem *Lup Recorder*

No	Kode Kebutuhan	Deskripsi Kebutuhan	Aktor	Nama *Use Case*
1	LUPR-F-01	Sistem harus menyediakan fungsi untuk merekam pengerjaan tugas.	Mahasiswa	Merekam pengerjaan tugas.
1.1	LUPR-F-01-01	Sistem harus menyediakan tombol "*Start Recording*" sebagai sarana untuk menjalankan perekaman tugas.		
1.2	LUPR-F-01-02	Sistem harus menyediakan fungsi untuk memilih direktori tempat tugas dikerjakan.		
1.3	LUPR-F-01-03	Data yang harus direkam adalah berkas tugas, alamat *IP*, seluruh *window*, *window* yang aktif, nama *login*, dan nama piranti.		
1.4	LUPR-F-01-04	Sistem harus menampilkan notifikasi "*Lup is recording*" ketika perekaman mulai berjalan.		
1.5	LUPR-F-01-05	Sistem harus menyediakan tombol "*Stop and Quit*" sebagai sarana untuk menghentikan perekaman tugas dan keluar dari sistem.		
2	LUPR-F-02	Sistem harus menyediakan fungsi untuk mengubah nilai interval rekaman.	Mahasiswa	Mengubah interval rekaman.

No	Kode Kebutuhan	Deskripsi Kebutuhan	Aktor	Nama *Use Case*
2.1	LUPR-F-02-01	Sistem harus menyediakan tombol "*Set Interval*" sebagai sarana untuk mengubah interval rekaman.		
2.2	LUPR-F-02-02	Sistem harus menyediakan kolom dialog untuk mengisi nilai interval.		
2.3	LUPR-F-02-03	Nilai interval hanya dapat diisi angka bilangan bulat diatas 0.		
2.4	LUPR-F-02-04	Sistem harus menampilkan peringatan "*Interval must higher than 0*" jika kolom interval diisi dengan selain bilangan bulat diatas 0.		
2.5	LUPR-F-02-05	Sistem harus menonaktifkan tombol "*Set Interval*" ketika perekaman belum dijalankan.		
2.6	LUPR-F-02-06	Sistem harus menampilkan notifikasi "*Interval changed to «nilai interval»*" ketika nilai interval berhasil dirubah.		

Tabel 4.8 Spesifikasi kebutuhan fungsional, definisi, dan pemetaan nama *use case* sistem *Lup Viewer*

No	Kode Kebutuhan	Definisi Kebutuhan	Aktor	Nama *Use Case*
1	LUPV-F-01	Sistem harus menyediakan fungsi untuk menampilkan seluruh daftar rekaman mahasiswa.	Dosen	Melihat seluruh daftar rekaman.
1.1	LUPV-F-01-01	Sistem harus menyediakan tombol "*Open Records*" sebagai sarana untuk memilih rekaman yang akan ditampilkan.		
1.2	LUPV-F-01-02	Sistem hanya dapat membaca direktori dengan format "«namamahasiswa-nimmahasiswa»" ("alfabet-angka").		
1.3	LUPV-F-01-03	Sistem hanya dapat membaca direktori yang memuat rekaman hasil dari sistem *Lup Recorder*.		
1.4	LUPV-F-01-04	Sistem harus menampilkan data berupa nama mahasiswa, NIM, total rekaman, waktu rekaman awal,waktu rekaman akhir, dan durasi pengerjaan tugas (*name, student ID, total records, first record, last record, work duration*).		

No	Kode Kebutuhan	Definisi Kebutuhan	Aktor	Nama *Use Case*
1.5	LUPV-F-01-05	Sistem harus menampilkan pesan peringatan "*Not a valid Task directory (baris baru) Contains invalid Task: «direktori yang tidak valid»*" jika terdapat direktori yang tidak valid.		
2	LUPV-F-02	Sistem harus menyediakan fungsi untuk menampilkan hasil rekaman pengerjaan tugas.	Dosen	Melihat hasil rekaman.
2.1	LUPV-F-02-01	Sistem harus memiliki tombol *combo box* "*Filename*" untuk memilih berkas tugas yang akan ditampilkan.		
2.2	LUPV-F-02-02	Sistem harus memiliki tombol *radio button* "*Diff Mode*" dan "*Show Mode*" sebagai sarana untuk mengalihkan mode tampilan rekaman berkas.		
2.3	LUPV-F-02-03	Sistem harus memiliki tombol *check box* "*Line Insertion and Deletion*" sebagai sarana untuk menampilkan jumlah baris yang dihapus dan ditambah pada berkas.		
2.4	LUPV-F-02-04	Sistem harus menampilkan seluruh daftar (per interval waktu) rekaman pengerjaan tugas.		

No	Kode Kebutuhan	Definisi Kebutuhan	Aktor	Nama *Use Case*
2.5	LUPV-F-02-05	Sistem harus menampilkan data berupa tanggal dan waktu rekaman, isi perubahan berkas, jumlah baris yang dihapus pada berkas, jumlah baris yang ditambah pada berkas, alamat *IP*, seluruh *window*,*window* yang aktif, nama *login*, dan nama piranti.		
2.6	LUPV-F-02-06	Sistem menampilkan pesan informasi “No file selected, please select one.” jika tidak ada berkas tugas yang dipilih.		
2.7	LUPV-F-02-07	Sistem menampilkan pesan informasi “No availiable record for «nama berkas» in this period” jika tidak ada perubahan tugas pada rekaman tertentu.		
2.8	LUPV-F-02-08	Sistem harus memiliki fungsi untuk mengalihkan isi perubahan berkas dari mode *diff* ke *show* mode dan sebaliknya.		
2.9	LUPV-F-02-09	Sistem harus memiliki fungsi untuk memberikan warna merah pada baris yang dihapus dan warna hijau pada baris yang ditambah dalam mode *diff*.		

No	Kode Kebutuhan	Definisi Kebutuhan	Aktor	Nama *Use Case*
2.10	LUPV-F-02-10	Sistem harus menampilkan pesan peringatan "*Please choose a file*" jika tombol *check box* "*Line Insertion and Deletion*" diaktifkan tanpa ada berkas yang dipilih.		
3	LUPV-F-03	Sistem harus menyediakan fungsi untuk mencari tersangka plagiarisme.	Dosen	Mencari tersangka.
3.1	LUPV-F-03-01	Sistem harus menyediakan kolom "*Line Insertion Limit*" untuk mengisi nilai *threshold* atau *limit* pada baris yang ditambahkan di dalam rekaman berkas tugas.		
3.2	LUPV-F-03-02	Sistem harus menyediakan tombol *combo box* "*Filename*" sebagai sarana untuk memilih berkas tugas.		
3.3	LUPV-F-03-03	Sistem harus menyediakan tombol "*Search*" sebagai sarana untuk memulai proses pencarian.		
3.4	LUPV-F-03-04	Data yang ditampilkan adalah nama, nim, nama berkas, *line insertion*, dan waktu (*name, student ID, filename, line insertion, date time*).		
3.5	LUPV-F-03-05	Sistem harus menampilkan peringatan "*Please select a file*" jika tidak ada berkas yang dipilih.		

No	Kode Kebutuhan	Definisi Kebutuhan	Aktor	Nama *Use Case*
3.6	LUPV-F-03-06	Sistem harus menampilkan peringatan "*Please set limit higher than 0 (baris baru) Above 10 is recommended*" jika kolom *line insertion limit* bernilai 0.		
3.7	LUPV-F-03-07	Sistem harus menampilkan informasi "*No suspect found, for «limit» insertion limit*" jika tidak ada tersangka yang ditemukan.		
4	LUPV-F-04	Sistem harus menyediakan fungsi untuk mencari window yang terbuka selama pengerjaan tugas.	Dosen	Mencari *window*.
4.1	LUPV-F-04-01	Sistem harus menyediakan kolom "*Window Name*" sebagai sarana untuk mengisi nama *window* yang akan dicari.		
4.2	LUPV-F-04-02	Sistem harus menyediakan tombol "*Search*" sebagai sarana untuk memulai proses pencarian.		
4.3	LUPV-F-04-03	Data yang ditampilkan adalah nama *window*, nama mahasiswa, nim, dan waktu (*window name, name, student ID, date time*).		

No	Kode Kebutuhan	Definisi Kebutuhan	Aktor	Nama *Use Case*
4.4	LUPV-F-04-04	Sistem harus menampilkan pesan peringatan "*Please supply the window name*" apabila kolom nama *window* kosong.		
4.5	LUPV-F-04-05	Sistem harus menampilkan informasi "*No window name for «input» found*" jika tidak ada *window* yang ditemukan.		
5	LUPV-F-05	Sistem harus menyediakan fungsi untuk menampilkan alamat *IP* seluruh mahasiswa selama pengerjaan tugas.	Dosen	Melihat seluruh alamat *IP*.
5.1	LUPV-F-05-01	Sistem harus menyediakan tombol "*Show all IP addresses*" sebagai sarana untuk menjalankan fungsi menampilkan seluruh alamat *IP*.		
5.2	LUPV-F-05-02	Data yang ditampilkan adalah alamat *IP*, nama mahasiswa, nim, dan waktu (*IP address, name, student ID, date time*).		
6	LUPV-F-06	Sistem harus menyediakan fungsi menampilkan grafik *edit-distance*.	Dosen	Melihat grafik *edit-distance*.
6.1	LUPV-F-06-01	Sistem harus menyediakan tombol "*Show*" sebagai sarana untuk menampilkan grafik *edit-distance*.		

No	Kode Kebutuhan	Definisi Kebutuhan	Aktor	Nama *Use Case*
6.2	LUPV-F-06-02	Sistem harus menyediakan tombol *combo box* "*Current Student*" sebagai sarana untuk memilih mahasiswa.		
6.3	LUPV-F-06-03	Sistem harus menyediakan tombol *combo box* "*Filename*" sebagai sarana untuk memilih nama berkas.		
6.4	LUPV-F-06-04	Sistem harus menampilkan pesan peringatan "*Please fill one of the pairs*" jika salah satu atau kedua *combo box* "*Filename*" dan "*Current Student*" tidak diisi.		
6.5	LUPV-F-06-05	Data yang harus ada dalam grafik *edit-distance* adalah nama mahasiswa, nim, legenda rekaman, dan legenda *edit-distance*.		
7	LUPV-F-07	Sistem harus menyediakan fungsi untuk melihat grafik dari nilai *edit-distance* tugas sebelumnya.	Dosen	Melihat grafik *edit-distance* tugas sebelumnya.
7.1	LUPV-F-07-01	Sistem harus menyediakan tombol "*Load Edit-distance File*" sebagai sarana untuk memilih berkas *edit-distance* sebelumnya.		

No	Kode Kebutuhan	Definisi Kebutuhan	Aktor	Nama *Use Case*
7.2	LUPV-F-07-02	Sistem harus menampilkan pesan peringatan "*Please load edit-distance file first*" jika berkas sebelumnya belum dimuat.		
7.3	LUPV-F-07-03	Sistem harus menggunakan fungsionalitas yang telah ada pada spesifikasi kebutuhan LUPV-F-06-01 hingga LUPV-F-06-05, dengan modifikasi nama kolom "*Current Studet*" menjadi "*Previous Student*"		
8	LUPV-F-08	Sistem harus menyediakan fungsi menyimpan grafik *edit-distance*.	Dosen	Menyimpan grafik *edit-distance*.
8.1	LUPV-F-08-01	Sistem harus menyediakan tombol "*Save Graph*" sebagai sarana untuk menyimpan berkas.		
8.2	LUPV-F-08-02	Sistem harus menyimpan grafik pada direktori tugas dengan membuat direktori baru bernama "*lupv-notes*".		
8.3	LUPV-F-08-03	Sistem harus menyimpan grafik dengan format "«namamahasiswa-nimmahasiswa».png", dan menggunakan tanda hubung "_" jika terdapat dua mahasiswa.		

No	Kode Kebutuhan	Definisi Kebutuhan	Aktor	Nama *Use Case*
8.4	LUPV-F-08-04	Sistem harus menggunakan fungsionalitas yang telah ada pada spesifikasi kebutuhan LUPV-F-06-04.		
8.5	LUPV-F-08-05	Sistem harus menampilkan pesan informasi "*Graph saved to «lokasi berkas»*" jika grafik berhasil disimpan.		
9	LUPV-F-09	Sistem harus menyediakan fungsi untuk mengekspor semua nilai *edit-distance*.	Dosen	Mengekspor semua nilai *edit-distance*.
9.1	LUPV-F-09-01	Data yang harus ada adalah nama mahasiswa, nim, perhitungan rekaman, nilai *edit-distance*, dan nama tugas.		
9.2	LUPV-F-09-02	Sistem harus menyediakan tombol "*Export Edit-distance*" sebagai sarana untuk mengekspor semua nilai *edit-distance*.		
9.3	LUPV-F-09-03	Sistem harus menyediakan *dialog window* yang di dalamnya terdapat tombol *combo box* untuk memilih berkas yang akan diekspor.		
9.4	LUPV-F-09-04	Sistem harus menyimpan berkas ekspor pada direktori tugas dengan membuat direktori baru bernama "*lupv-notes*".		

No	Kode Kebutuhan	Definisi Kebutuhan	Aktor	Nama *Use Case*
9.5	LUPV-F-09-05	Sistem harus menyimpan berkas ekspor dengan nama berkas "«namatugas»-editdistance.lup".		
9.6	LUPV-F-09-06	Sistem harus menampilkan pesan informasi "*Edit-distance exported to «lokasi berkas»*" jika ekspor berhasil.		
10	LUPV-F-10	Sistem harus menyediakan fungsi untuk mengalihkan format waktu.	Dosen	Mengalihkan format waktu.
10.1	LUPV-F-10-01	Sistem harus menyediakan dua format tanggal untuk dialihkan, yaitu format *real date time* dan format *relative date time*.		
10.2	LUPV-F-10-02	Format *real date time* berbentuk "«nama pendek hari, tanggal bulan tahun, jam:menit:detik»" dan format *relative date time* berbentuk "«x *hours ago*»".		
10.3	LUPV-F-10-03	Sistem harus menyediakan tombol *radio button* "*Relative DateTime*" dan "*Real DateTime*" sebagai sarana untuk mengalihkan antar format waktu.		

4.5 Kebutuhan Non-Fungsional Sistem

Kebutuhan non-fungsional merupakan batasan (*constraint*) dari sebuah sistem (Sommerville, 2014). Daftar kebutuhan non-fungsional didapatkan dari proses analisis kebutuhan yang telah dilakukan sebelumnya. Dalam penelitian ini, kebutuhan non-fungsional memiliki sistem penomoran LUPV-NF-XX-YY untuk sistem *Lup Viewer* dan LUPR-NF-XX-YY untuk sistem *Lup Recorder*. *LUPV* dan *LUPR* merupakan nama sistem, NF menunjukkan tipe kebutuhan yaitu kebutuhan non-fungsional, XX menunjukkan nomor kebutuhan, dan YY menunjukkan nomor spesifikasi kebutuhan. Tabel 4.9 dan Tabel 4.10 memaparkan daftar kebutuhan non-fungsional, definisi, dan parameter kebutuhan non-fungsional.

Tabel 4.9 Spesifikasi kebutuhan non-fungsional, parameter, dan deskripsi kebutuhan sistem *Lup Recorder*

No	Kode Kebutuhan	Parameter	Deskripsi Kebutuhan
1	LUPR-NF-01	*compatibility*	Sistem harus berjalan dengan baik pada enam *desktop environment* yang berbeda.

Tabel 4.10 Spesifikasi kebutuhan non-fungsional, parameter, dan deskripsi kebutuhan sistem *Lup Viewer*

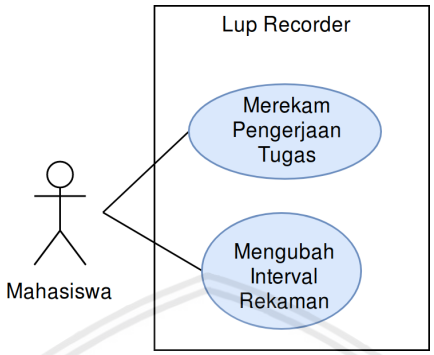
No	Kode Kebutuhan	Parameter	Deskripsi Kebutuhan
1	LUPV-NF-01	*compatibility*	Sistem harus berjalan dengan baik pada enam *desktop environment* yang berbeda.

4.6 Pemodelan Kebutuhan

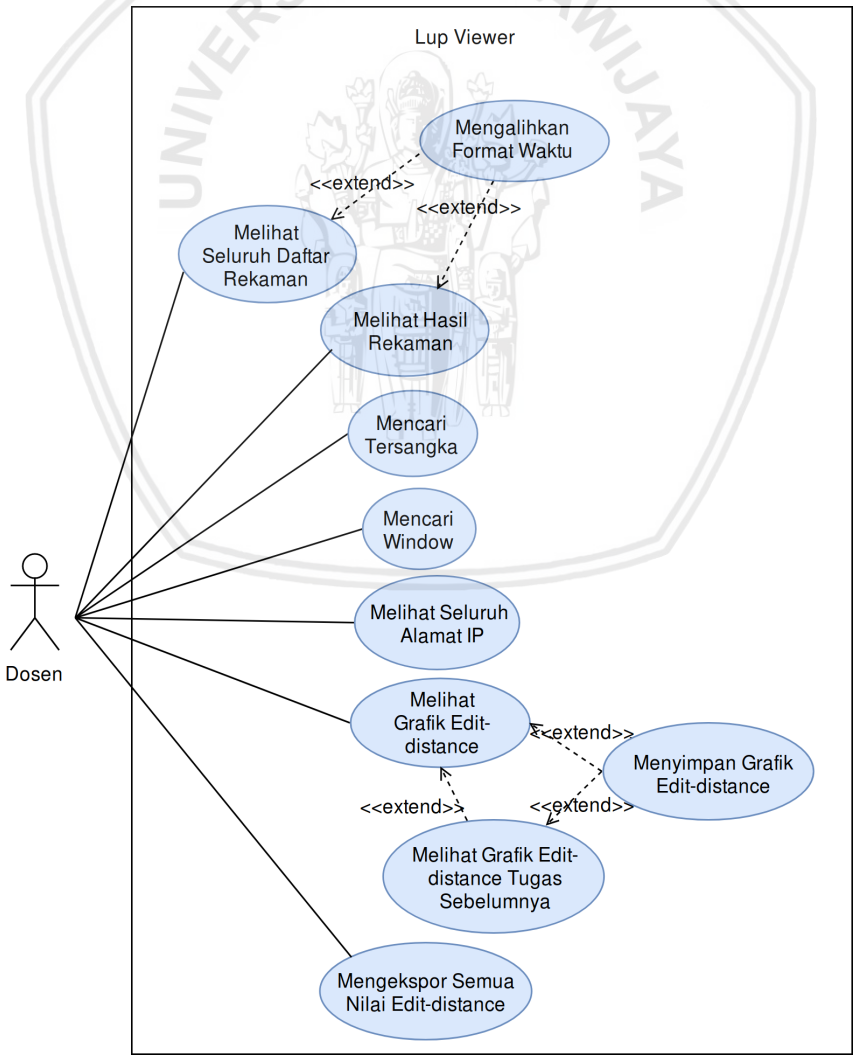
4.6.1 *Use Case Diagram*

Use case diagram digunakan untuk menggambarkan fungsionalitas dan batasan sistem secara visual. Selain itu, *use case diagram* juga menggambarkan interaksi antar aktor dengan sistem dari sudut aktor dalam bentuk visual. Aktor ditun-

jukkan dengan *stick figure* dan *use case* ditunjukkan dengan oval. *Use case* yang digambarkan didapatkan dari hasil analisis kebutuhan sebelumnya. Gambar 4.15 menunjukkan *use case diagram* untuk sistem *Lup Recorder* dan Gambar 4.16 menunjukkan *use case diagram* untuk sistem *Lup Viewer*.



Gambar 4.15 *Use case diagram Lup Recorder*



Gambar 4.16 *Use case diagram Lup Viewer*

4.6.2 *Use Case Scenario*

Use case scenario digunakan untuk menjelaskan skenario interaksi antar aktor dan sistem secara tekstual dengan detail. Dalam setiap *use case scenario* dipaparkan *objective* atau tujuan *use case*, aktor yang terlibat, kondisi sebelumnya yang harus dipenuhi *pre-condition*, alur utama *main flow*, alur alternatif *alternative flow*, dan kondisi setelah *use case* dijalankan *post-condition*. *Use case scenario* untuk sistem *Lup Recorder* dipaparkan pada Tabel 4.11 hingga Tabel 4.12, dan *use case scenario* sistem *Lup Viewer* dipaparkan pada Tabel 4.13 hingga 4.22.

Tabel 4.11 *Use case scenario* untuk Merekam Pengerjaan Tugas

Flow of events untuk Merekam Pengerjaan Tugas	
Kode Kebutuhan	LUPR-F-01
Objective	Mahasiswa dapat merekam pengerjaan tugas.
Actor	Mahasiswa
Pre-condition	Tampilan utama sistem *Lup Recorder* sudah tersedia.
Main flow	1. Mahasiswa memilih tombol "*Start Recording*" untuk mulai merekam pengerjaan tugas. 2. Sistem menampilkan dialog pemilihan letak direktori tugas. Pada dialog tersedia tombol "*Choose*" dan "*Cancel*". 3. Mahasiswa memilih letak direktori tugas dan memilih tombol "*Choose*". 4. Sistem menjalankan perekaman pengerjaan tugas dengan data yang direkam meliputi berkas tugas, alamat *IP*, seluruh *window*, *window* yang aktif, nama *login*, dan nama piranti. 5. Sistem menampilkan pesan notifikasi "*Lup is recording*" menandakan bahwa perekaman sudah dijalankan.
Alternative flows	1.1 Sistem berhenti dan keluar jika mahasiswa memilih tombol "*Stop and Quit*". 2.1 Sistem menutup dialog jika mahasiswa memilih tombol "*Cancel*".
Post-condition	Pengerjaan tugas direkam oleh sistem.

Tabel 4.12 *Use case scenario* untuk Mengubah Interval Rekaman

Flow of events untuk Mengubah Interval Rekaman	
Kode Kebutuhan	LUPR-F-02
Objective	Mahasiswa dapat mengubah interval rekaman.
Actor	Mahasiswa
Pre-condition	Perekaman tugas sudah dijalankan.
Main flow	1. Mahasiswa memilih tombol "*Set Interval*". 2. Sistem menampilkan dialog interval yang terdiri dari kolom nilai interval. Pada dialog interval tersedia tombol "*Apply*" dan "*Cancel*". 3. Mahasiswa memberikan nilai bilangan bulat diatas 0 pada kolom interval. 4. Mahasiswa memilih tombol "*Apply*" pada kolom interval. 5. Sistem melakukan validasi nilai interval dan mengubah nilai interval rekaman. 6. Sistem menampilkan pesan notifikasi "*Interval changed to «nilai interval»*" menandakan bahwa nilai interval berhasil dirubah.
Alternative flows	5.1 Apabila kolom bernilai selain bilangan bulat diatas 0, maka sistem menampilkan pesan peringatan "*Interval must higher than 0*".
Post-condition	Nilai interval rekaman diubah oleh sistem.

Tabel 4.13 *Use case scenario* untuk Melihat Seluruh Daftar Rekaman

Flow of events untuk Melihat Seluruh Daftar Rekaman	
Kode Kebutuhan	LUPV-F-01
Objective	Dosen dapat melihat seluruh daftar rekaman.
Actor	Dosen
Pre-condition	Tampilan utama sistem *Lup Viewer* sudah tersedia.

Main flow	1. Dosen memilih tombol "*Open Records*". 2. Sistem menampilkan dialog untuk memilih direktori rekaman yang akan dibuka. Pada dialog tersebut terdapat tombol "*Choose*" dan "*Cancel*". 3. Dosen memilih tombol "*Choose*". 4. Sistem melakukan validasi direktori rekaman dan menampilkan seluruh daftar rekaman mahasiswa. Data yang ditampilkan meliputi nama, nim, total rekaman, waktu rekaman awal, waktu rekaman akhir, dan durasi pengerjaan.
Alternative flows	3.1 Apabila dosen memilih tombol "*Cancel*", maka berkas rekaman tidak dibuka. 4.1 Apabila direktori tidak valid, yaitu tidak sesuai dengan spesifikasi kebutuhan LUPV-F-01-02 dan LUPV-F-01-03, maka sistem menampilkan pesan peringatan "*Not a valid Task directory (baris baru) Contains invalid Task: «direktori yang tidak valid»*". 4.2 Perluasan ke *use case* dengan kode kebutuhan LUPV-F-10.
Post-condition	Seluruh daftar rekaman ditampilkan oleh sistem.

Tabel 4.14 *Use case scenario* untuk Menampilkan Hasil Rekaman

Flow of events untuk Melihat Hasil Rekaman	
Kode Kebutuhan	LUPV-F-02
Objective	Dosen dapat melihat hasil rekaman.
Actor	Dosen
Pre-condition	Tampilan seluruh daftar rekaman mahasiswa sudah tersedia.

Main flow	1. Dosen memilih *clickable* baris mahasiswa pada seluruh daftar rekaman. 2. Sistem menampilkan *clickable list* daftar rekaman. 3. Dosen memilih *clickable list* daftar rekaman. 4. Sistem menampilkan rekaman alamat *IP*, seluruh *window*, *window* yang aktif, nama *login*, dan nama piranti. 5. Dosen memilih nama berkas pada tombol *combo box* "*Filename*". 6. Sistem menampilkan hasil rekaman isi perubahan berkas. 7. Dosen memilih tombol "*Line Insertion and Deletion*". 8. Sistem menampilkan jumlah baris yang dihapus dan jumlah baris yang ditambah pada berkas. 9. Dosen memilih tombol "*Show Mode*" dan "*Diff Mode*". 10. Sistem menampilkan format perubahan berkas dengan mode *show* dan mode *diff*.
Alternative flows	4.1 Perluasan ke *use case* dengan kode kebutuhan LUPV-F-10. 6.1 Sistem menampilkan pesan informasi "*No file selected, please select one*" jika tidak ada berkas tugas yang dipilih. 6.2 Sistem menampilkan pesan informasi "*No avaiiliable record for «nama berkas» in this period*" jika tidak ada perubahan tugas pada rekaman yang dipilih. 7.1 Sistem menampilkan pesan peringatan "*Please choose a file*" jika tidak ada berkas yang dipilih.
Post-condition	Hasil rekaman ditampilkan oleh sistem.

Tabel 4.15 *Use case scenario* untuk Mencari Tersangka.

Flow of events untuk Mencari Tersangka	
Kode Kebutuhan	LUPV-F-03
Objective	Dosen dapat mencari tersangka.

Actor	Dosen
Pre-condition	Tampilan pencarian tersangka sudah tersedia.
Main flow	1. Dosen memilih nama berkas dari tombol *combo box* "*Filename*" dan mengisi nilai "*Line Insertion Limit*". 2. Dosen memilih tombol "*Search*" untuk menjalankan pencarian. 3. Sistem melakukan validasi nilai "*Line Insertion Limit*" dan menampilkan hasil pencarian tersangka.
Alternative flows	3.1 Sistem menampilkan pesan peringatan "*Please set limit higher than 0 (baris baru) Above 10 is recommended*" apabila nilai pada kolom *insertion* bernilai selain bilangan bulat di atas 0. 3.2 Sistem menampilkan pesan informasi "*No suspect found, for «limit» insertion limit*" jika tidak ada tersangka yang ditemukan. 3.3 Sistem menampilkan pesan peringatan "*Please select a file*" apabila tidak ada berkas tugas yang dipilih.
Post-condition	Hasil pencarian tersangka ditampilkan oleh sistem.

Tabel 4.16 *Use case scenario* untuk Mencari *Window*

Flow of events untuk Mencari *Window*	
Kode Kebutuhan	LUPV-F-04
Objective	Dosen dapat mencari *window*.
Actor	Dosen
Pre-condition	Tampilan pencarian *window* telah tersedia.
Main flow	1. Dosen mengisi kolom "*Window Name*". 2. Dosen memilih tombol "*Search*" untuk menjalankan pencarian. 3. Sistem melakukan validasi nilai kolom "*Window Name*" dan menampilkan hasil pencarian *window*.

Alternative flows	3.1 Sistem menampilkan pesan informasi "*No window name for «nama window» found*" jika tidak ada *window* yang ditemukan. 3.1 Sistem menampilkan pesan peringatan "*Please supply the window name*" apabila kolom "*Window Name*" kosong.
Post-condition	Hasil pencarian *window* ditampilkan oleh sistem.

Tabel 4.17 *Use case scenario* untuk Menampilkan Seluruh Alamat *IP*

Flow of events untuk Melihat Seluruh Alamat *IP*	
Kode Kebutuhan	LUPV-F-05
Objective	Dosen dapat melihat seluruh alamat *IP*.
Actor	Dosen
Pre-condition	Tampilan untuk melihat seluruh alamat *IP* telah tersedia.
Main flow	1. Dosen memilih tombol "*Show all IP addresses*". 2. Sistem menampilkan seluruh alamat *IP*.
Alternative flows	-
Post-condition	Seluruh alamat *IP* ditampilkan oleh sistem

Tabel 4.18 *Use case scenario* untuk Melihat Grafik *Edit-distance*.

Flow of events untuk Melihat Grafik *Edit-distance*	
Kode Kebutuhan	LUPV-F-06
Objective	Dosen dapat melihat grafik *edit-distance*.
Actor	Dosen
Pre-condition	Tampilan untuk melihat grafik *edit-distance* telah tersedia.
Main flow	1. Dosen memilih mahasiswa dari tombol *combo box* "*Current Student*" dan memilih berkas dari tombol *combo box* "*Filename*". 2. Dosen memilih tombol "*Show Mode*". 3. Sistem melakukan validasi nilai tombol *combo box* "*Current Student*" dan "*Filename*", kemudian menampilkan grafik *edit distance*.

Alternative flows	1.2 Perluasan ke *use case* dengan kode kebutuhan LUPV-F-07 dan LUPV-F-08. 2.1 Sistem menampilkan pesan peringatan "*Please fill one of the pairs*" jika salah satu atau kedua *combo box* "*Filename*" dan "*Current Student*" tidak diisi.
Post-condition	Grafik *edit distance* ditampilkan oleh sistem.

Tabel 4.19 *Use case scenario* untuk Melihat Grafik *Edit-distance* Tugas Sebelumnya

Flow of events untuk Melihat Grafik *Edit-distance* Tugas Sebelumnya	
Kode Kebutuhan	LUPV-F-07
Objective	Dosen dapat melihat grafik *edit-distance* tugas sebelumnya.
Actor	Dosen
Pre-condition	Tampilan untuk melihat grafik *edit-distance* telah tersedia.
Main flow	1. Dosen memilih tombol "*Load Edit-distance*". 2. Sistem menampilkan dialog untuk memilih berkas *edit-distance* tugas sebelumnya. Dialog memiliki tombol "*Open*" dan "*Cancel*". 3. Dosen memilih berkas dan memilih tombol "*Open*". 4. Sistem membaca berkas *edit-distance* tugas sebelumnya. 5. Dosen memilih mahasiswa dari tombol *combo box* "*Previous Student*" dan memilih berkas dari tombol *combo box* "*Filename*". 6. Dosen memilih tombol "*Show Mode*". 7. Sistem melakukan validasi nilai tombol *combo box* "*Previous Student*" dan "*Filename*", kemudian menampilkan grafik *edit distance* tugas sebelumnya.

Alternative flows	3.1 Sistem tidak membaca berkas jika dosen memilih tombol "*Cancel*". 5.1 Sistem menampilkan pesan peringatan "*Please load edit-distance file first*" jika dosen belum membuka berkas *edit-distance* tugas sebelumnya. 5.2 Sistem menampilkan pesan peringatan "*Please fill one of the pairs*" jika salah satu atau kedua *combo box* "*Filename*" dan "*Previous Student*" belum dipilih. 6.1 Perluasan ke *use case* dengan kode kebutuhan LUPV-F-08.
Post-condition	Grafik *edit distance* tugas sebelumnya ditampilkan oleh sistem.

Tabel 4.20 *Use case scenario* untuk Menyimpan Grafik *Edit-distance* .

Flow of events untuk Menyimpan Grafik *Edit-distance*	
Kode Kebutuhan	LUPV-F-08
Objective	Dosen dapat menyimpan grafik *edit-distance*.
Actor	Dosen
Pre-condition	Grafik *edit-distance* telah tersedia.
Main flow	1. Dosen memilih tombol "*Save Graph*". 2. Sistem menyimpan grafik pada direktori "*lupv-notes*" pada direktori tugas. 3. Sistem menampilkan pesan informasi "*Graph saved to «lokasi berkas»*" menandakan bahwa grafik telah disimpan.
Alternative flows	-
Post-condition	Grafik *edit distance* disimpan oleh sistem.

Tabel 4.21 *Use case scenario* untuk Mengekspor Semua Nilai *Edit-distance*

Flow of events untuk Mengekspor Semua Nilai *Edit-distance*	
Kode Kebutuhan	LUPV-F-09
Objective	Dosen dapat mengekspor semua nilai *edit-distance*.
Actor	Dosen

Pre-condition	Tampilan seluruh daftar rekaman mahasiswa telah tersedia.
Main flow	1. Dosen memilih tombol "*Export Edit-distance*". 2. Sistem menampilkan dialog pemilihan berkas yang akan di ekspor. Pada dialog tersedia tombol "*Export*" dan "*Cancel*". 3. Dosen memilih tombol "*Export*". 4. Sistem mengekspor semua nilai *edit-distance* pada berkas yang dipilih dan menyimpan hasil ekspor dengan nama "«namatugas»-editdistance.lup". 5. Sistem menampilkan pesan informasi "*Edit-distance exported to «lokasi berkas»*" menandakan bahwa semua nilai *edit-distance* telah diekspor.
Alternative flows	3.1 Apabila dosen memilih tombol "*Cancel*", maka sistem menutup dialog pemilihan berkas dan nilai *edit-distance* tidak diekspor.
Post-condition	Semua nilai *edit-distance* diekspor oleh sistem.

Tabel 4.22 *Use case scenario* untuk Mengalihkan Format Waktu .

Flow of events untuk Mengalihkan Format Waktu	
Kode Kebutuhan	LUPV-F-10
Objective	Dosen dapat mengalihkan format waktu.
Actor	Dosen
Pre-condition	Tampilan waktu rekaman telah tersedia.
Main flow	1. Dosen memilih tombol "*Real DateTime*" dan "*Relative DateTime*". 2. Sistem mengalihkan format waktu. Format bentuk "«nama pendek hari, tanggal bulan tahun, jam:menit:detik»" untuk format *real* dan "«x *hours ago*»" untuk format *relative*.
Alternative flows	-
Post-condition	Format waktu dialihkan oleh sistem.

BAB 5 PERANCANGAN DAN IMPLEMENTASI

5.1 Perancangan Sistem

Tahapan perancangan sistem merupakan sebuah tahapan yang dilakukan setelah selesainya proses rekayasa kebutuhan. Hasil tahapan perancangan nantinya digunakan sebagai landasan tahapan implementasi sistem. Tiga bagian yang terdapat pada tahapan ini adalah perancangan arsitektur, perancangan komponen, dan perancangan antarmuka.

5.1.1 Perancangan Arsitektur

Sistem dibangun menggunakan arsitektur *Model-View-Controller (MVC)*. Maka komponen fungsionalitas inti sistem terpisah dari komponen antarmuka sistem. Pemisahan ini memberikan manfaat untuk pengembangan sistem pada tahap selanjutnya. Salah satu manfaat tersebut adalah kemudahan dalam mengembangkan antarmuka lain untuk sistem, hal ini dikarenakan pengembang hanya fokus pada komponen antarmuka yang akan dibangun. Perancangan arsitektur dilakukan dengan menggunakan notasi *UML sequence diagram* dan *class diagram*. *Class diagram* bertujuan menggambarkan struktur sistem dan relasi antar komponen dalam sistem secara statis. Pada *class diagram* digambarkan *class*, *method* atau *function*, dan relasi antar *class*. *Sequence diagram* merupakan diagram dinamis yang berfungsi untuk menggambarkan interaksi antar komponen atau objek yang berada di dalam sistem. Terdapat tiga sampel *sequence diagram* yang akan dipaparkan pada bagian ini. Tiga sampel tersebut adalah *sequence diagram* untuk merekam pengerjaan tugas, *sequence diagram* untuk melihat hasil rekaman, dan *sequence diagram* untuk melihat grafik *edit-distance* tugas sebelumnya. Sementara itu, *class diagram* sistem akan dipaparkan seluruhnya.

5.1.1.1 *Sequence Diagram* Merekam Pengerjaan Tugas

Sequence diagram untuk merekam pengerjaan tugas dapat dilihat pada Gambar 5.17. Pada *sequence diagram* tersebut terdapat beberapa objek yang berinteraksi. Satu objek *boundary* "*MainView*", dua objek *controller* "*Controller*" dan "*RecordWorker*", dan satu objek *entity* "*Model*". Aktor yang terlibat adalah mahasiswa.

5.1.1.2 *Sequence Diagram* Melihat Hasil Rekaman

Sequence diagram untuk melihat hasil rekaman dapat dilihat pada Gambar 5.18. Pada *sequence diagram* tersebut terdapat beberapa objek yang berinteraksi. Dua objek *boundary* "*MainView*" dan "*LogView*", dua objek *controller* "*LogController*" dan "*MainController*", dan satu objek *entity* "*LogModel*". Aktor yang terlibat adalah dosen.

5.1.1.3 *Sequence Diagram* Melihat Grafik *Edit-Distance* Tugas Sebelumnya

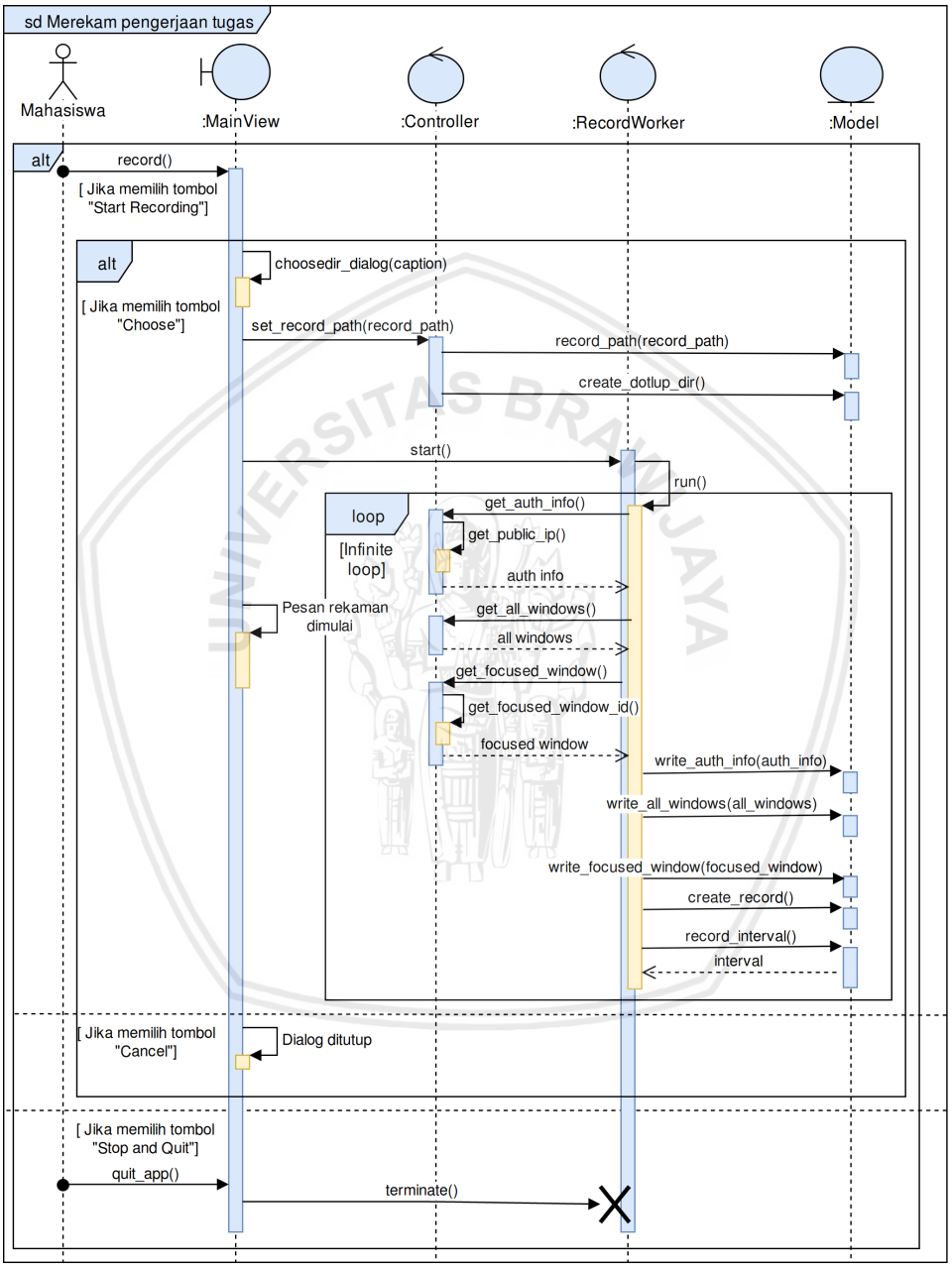
Sequence diagram untuk melihat grafik *edit-distance* tugas sebelumnya dapat dilihat pada Gambar 5.19. Pada *sequence diagram* tersebut terdapat beberapa objek yang berinteraksi. Satu objek *boundary* "*SearchView*", satu objek *controller* "*SearchController*", dan dua objek *entity* "*SearchModel*" dan "*MainModel*". Aktor yang terlibat adalah dosen.

5.1.1.4 *Class Diagram*

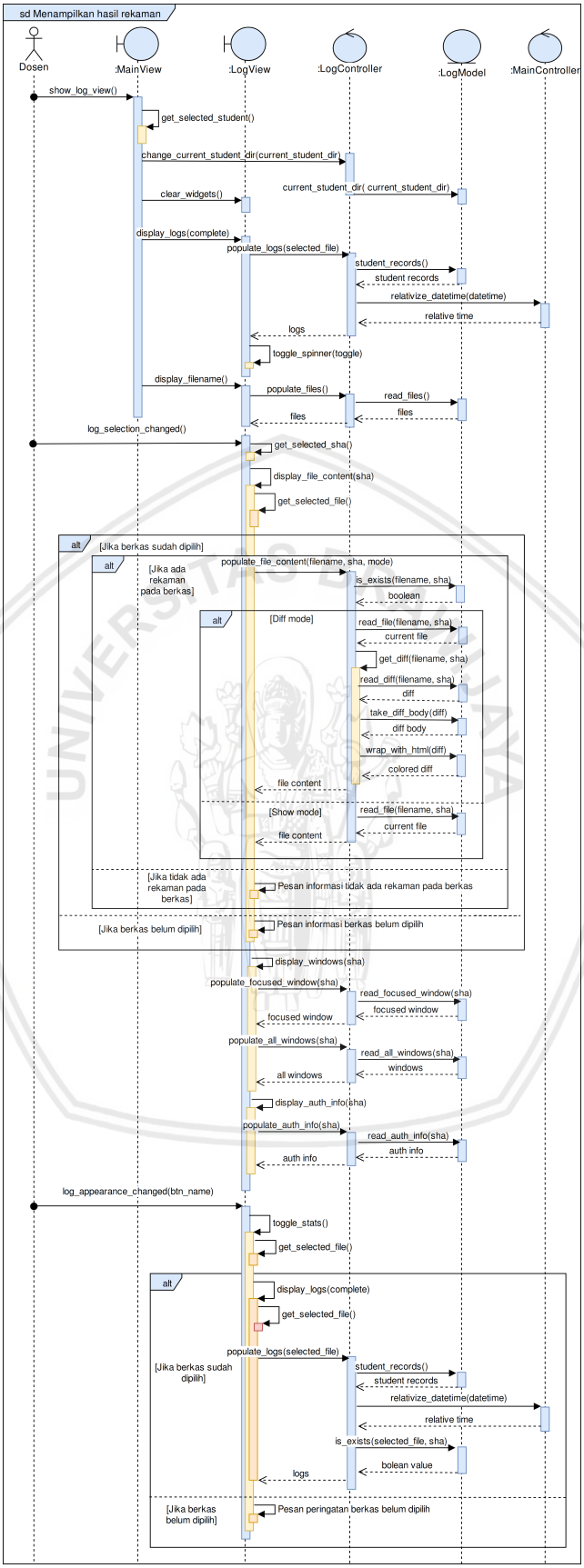
Pada bagian ini akan dipaparkan *class-class* yang menyusun sistem yang dibangun. Gambar 5.20 merupakan *class diagram* dari sistem *Lup Recorder*, dan Gambar 5.21 merupakan *class diagram* dari sistem *Lup Viewer*. Sistem yang dibangun menggunakan arsitektur *Model-View-Controller* sehingga *class-class* yang menyusun sistem pada *class diagram* umumnya terdiri dari *class Model*, *class View* dan *class Controller*. Pada *class diagram* juga dipaparkan relasi antar *class* yang umumnya merupakan relasi agregasi dan komposisi.

5.1.2 Perancangan Komponen

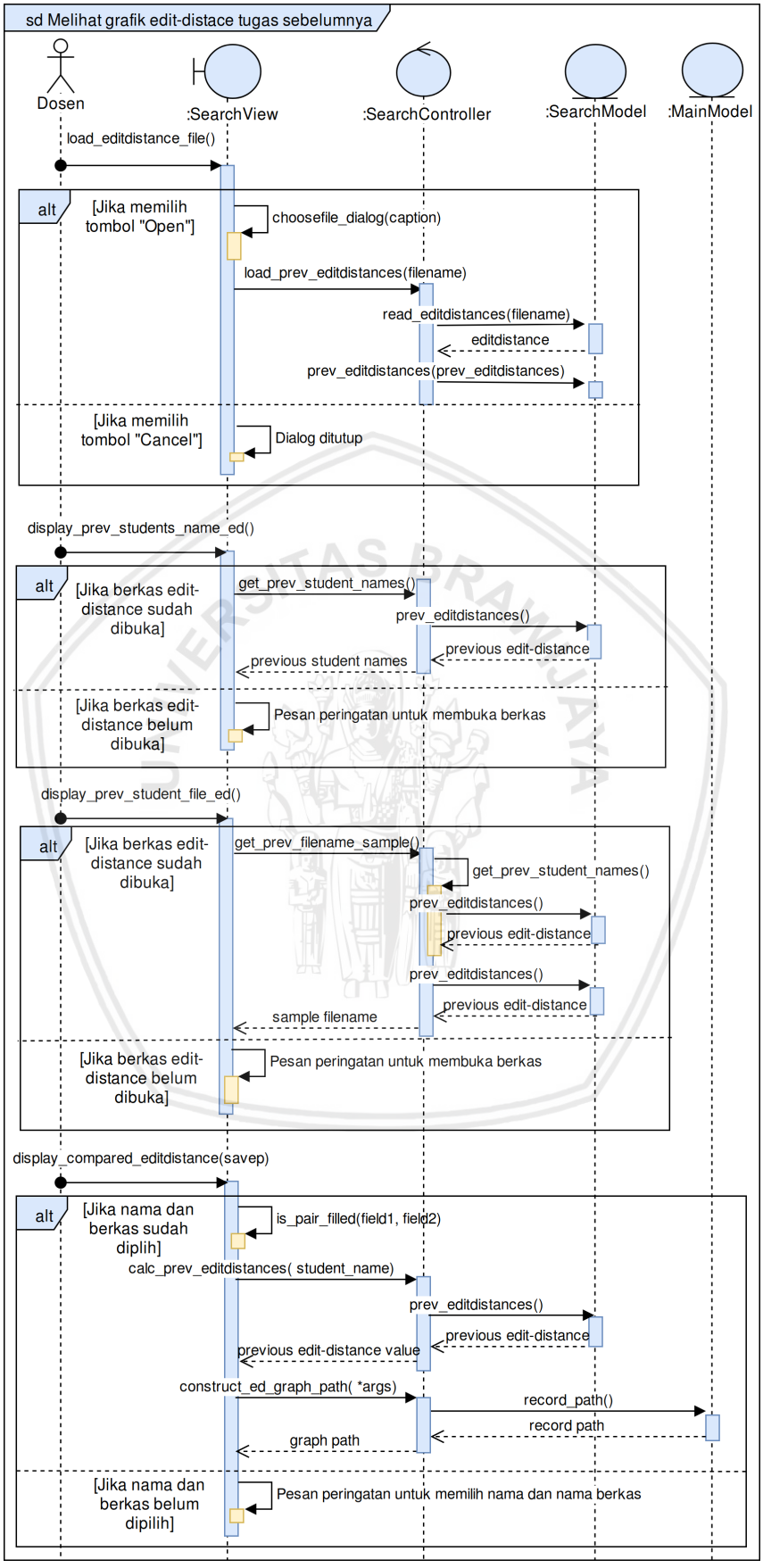
Pada tahapan perancangan komponen, komponen-komponen yang membentuk sistem akan dirancang menggunakan *pseudocode*. Komponen yang membentuk sistem dalam sistem berorientasi objek adalah *class*. Maka tahapan ini akan memaparkan perancangan tiga sampel *class*. Tidak semua *procedure* atau *function* di dalam suatu *class* akan dipaparkan, melainkan hanya satu sampel *function* dari tiap-tiap *class*. Tabel Kode 5.5 memaparkan *pseudocode* untuk *function get_all_windows* dari *class Controller* pada sistem *Lup Recorder*. Tabel Kode 5.6 memaparkan *pseudocode* untuk *function populate_logs* dari *class LogController* pada sistem *Lup Viewer* dan, Tabel Kode 5.7 memaparkan *pseudocode* untuk *function construct_ed_graph_path* dari *class SearchController* pada sistem *Lup Viewer*.



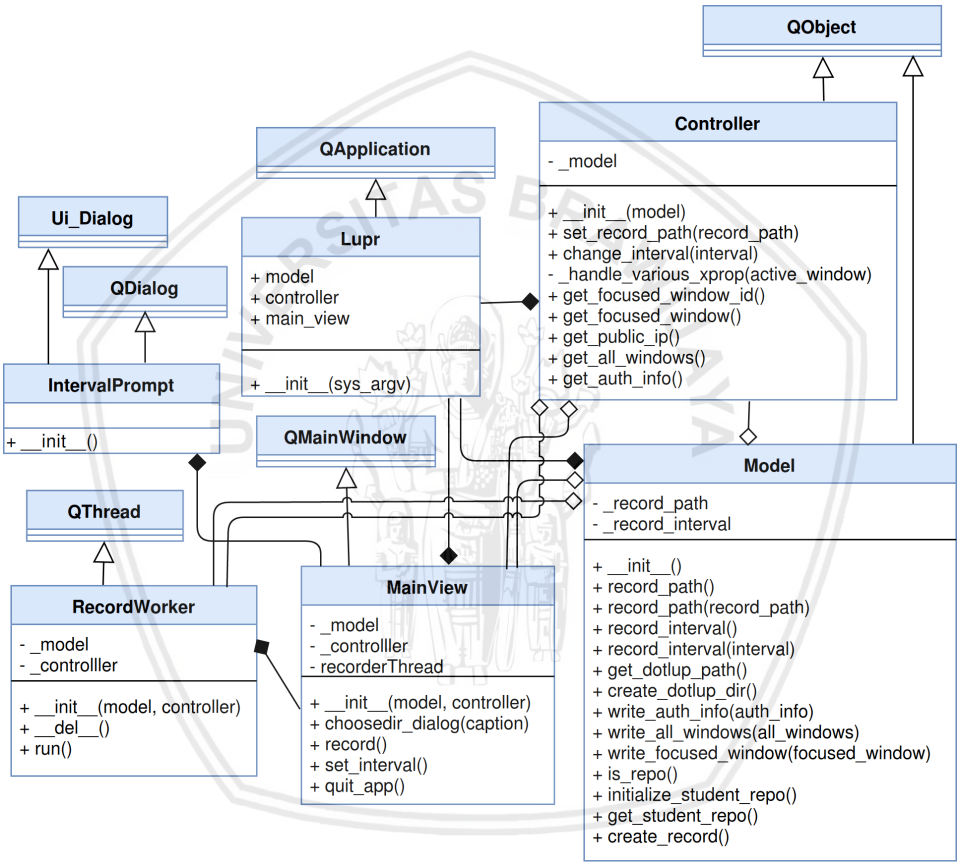
Gambar 5.17 *Sequence diagram* Merekam Pengerjaan Tugas



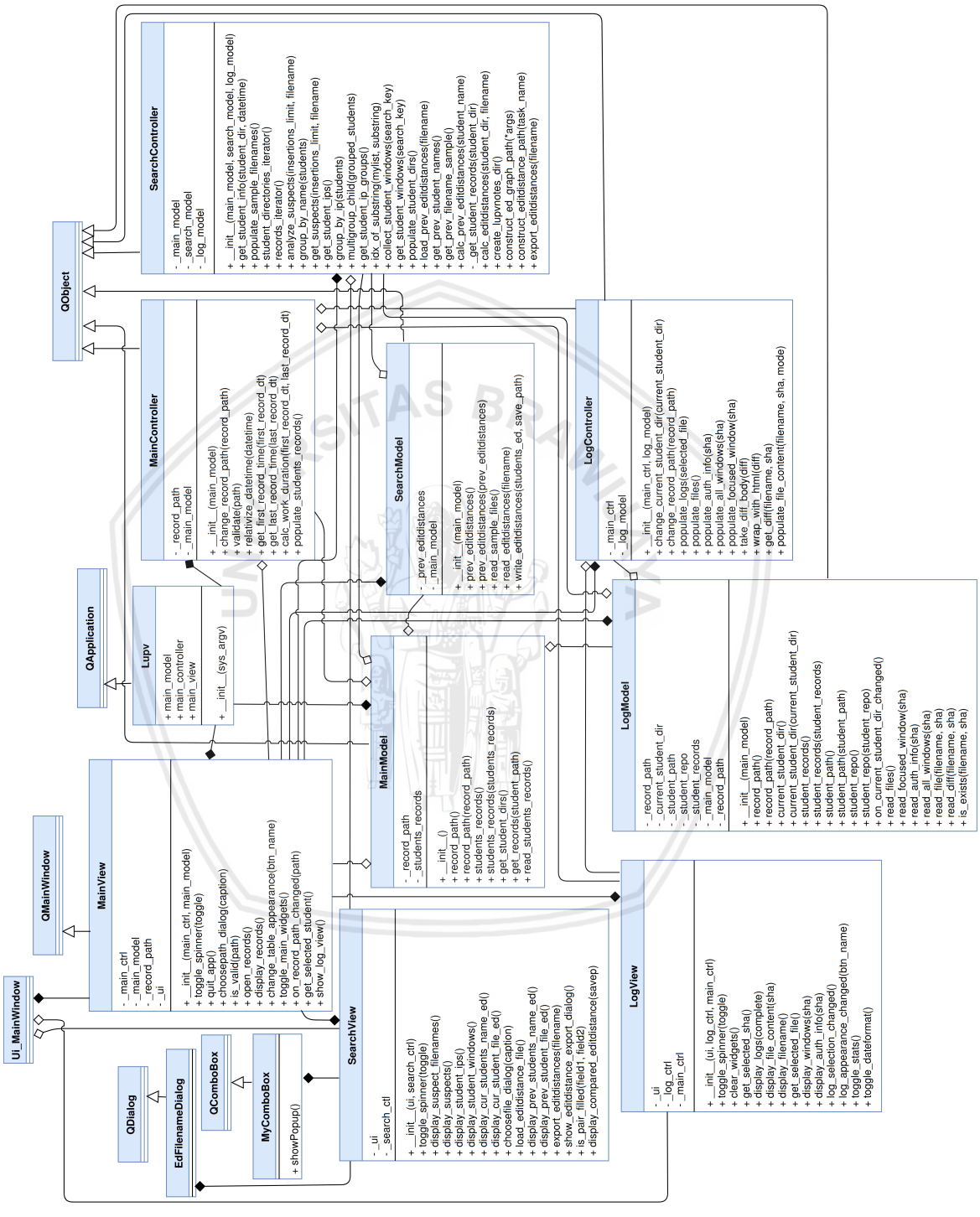
Gambar 5.18 *Sequence diagram* Melihat Hasil Rekaman



Gambar 5.19 *Sequence diagram* Melihat Grafik *Edit-Distance* Tugas Sebelumnya



Gambar 5.20 Pemodelan *class diagram Lup Recorder*



Gambar 5.21 Pemodelan *class diagram Lup Viewer*

5.1.2.1 Perancangan Komponen *Class Controller*

Pada Tabel Kode 5.5 dipaparkan *pseudocode get_all_windows* pada sistem *Lup Recorder*. *Pseudocode get_all_windows* dirancang untuk membaca seluruh *window* yang sedang dibuka. Sistem operasi memiliki banyak atribut selain nama *window*, maka *get_all_windows* hanya mengambil nilai nama *window* dan mengembalikannya.

No	get_all_windows
1	`START`
2	`  READ all_windows_dirty`
3	`  IF all_windows_dirty is true`
4	`    SET all_windows_dirty to split every line`
5	`    FOR each window in all_windows_dirty`
6	`      GET window_name`
7	`      SET all_windows = window_name`
8	`    ENDFOR`
9	`  ENDIF`
10	`  RETURN all_windows`
11	`END`

Tabel Kode 5.5 *Pseudocode function get_all_windows*

5.1.2.2 Perancangan Komponen *Class LogController*

Pada Tabel Kode 5.6 dipaparkan *pseudocode populate_log* pada sistem *Lup Viewer*. *Pseudocode populate_log* dirancang untuk membaca seluruh rekaman mahasiswa, dari setiap rekaman diambil nilai-nilai yang dibutuhkan di antaranya adalah waktu rekaman, baris yang ditambah, dan baris yang dihapus.

No	populate_logs
1	`START`
2	`  CALL log_model RETURNING student_records`
3	
4	`  FOR each record in student_records`
5	`    CALL main_ctrl with relativize_datetime RETURNING relative_time`
6	`    GET time`
7	`    GET sha`
8	`    SET insetion to 0`

```
 9       SET deletion to 0
10       IF selected_file is true
11         IF CALL log_model with is_exist RETURNING true
12           GET insertion
13           GET deletion
14         ENDIF
15       ENDIF
16     ENDFOR
17     SET log to (relative_time, time, sha, insertion,
       ↪ deletion)
18     RETURN log
19   END
```

Tabel Kode 5.6 *Pseudocode function populate_logs*

5.1.2.3 Perancangan Komponen *Class SearchController*

Pada Tabel Kode 5.7 dipaparkan *pseudocode construct_ed_graph_path* pada sistem *Lup Viewer*. *Pseudocode construct_ed_graph_path* dirancang untuk membangun *path* untuk nama grafik *edit-distance* yang disimpan. *Path* tersebut mengikuti jumlah mahasiswa yang ada. Jika terdapat lebih dari satu mahasiswa, maka diberikan "_" sebagai penghubung nama antar mahasiswa.

No	construct_ed_graph_path
1	START
2	CALL main_model RETURNING record_path
3	
4	IF passed argument == 1
5	SET graph_fmt to argument + ".png"
6	ELSE
7	SET graph_fmt to argument + "_" + ".png"
8	ENDIF
9	SET graph_path record_path + "lupv_notes" + graph_fmt
10	RETURN graph_path
11	END

Tabel Kode 5.7 *Pseudocode function construct_ed_graph_path*

5.1.3 Perancangan Antarmuka

Perancangan antarmuka merupakan proses merancang antarmuka sistem dengan menggambar gambaran dasar antarmuka sistem. Rancangan antarmuka dibuat dengan bantuan kakas bantu *draw.io*. Gambaran rancangan ini nantinya dijadikan landasan implementasi antarmuka. Terdapat tiga sampel perancangan antarmuka yang akan dipaparkan, yaitu perancangan antarmuka tampilan *tray* sistem *Lup Recorder*, perancangan antarmuka tampilan hasil rekaman sistem *Lup Viewer*, dan perancangan antarmuka tampilan *edit-distance* tugas sebelumnya.

5.1.3.1 Perancangan Antarmuka Tampilan *Tray* Pada Sistem *Lup Recorder*

Gambar 5.22 merupakan rancangan antarmuka tampilan *tray* pada sistem *Lup Recorder*. Antarmuka ini merupakan rancangan antarmuka utama pada sistem *Lup Recoder* dan digunakan untuk mengakses seluruh menu utama sistem *Lup Recorder*. Penjelasan komponen-komponen yang terdapat pada rancangan tersebut terdapat pada Tabel 5.23.

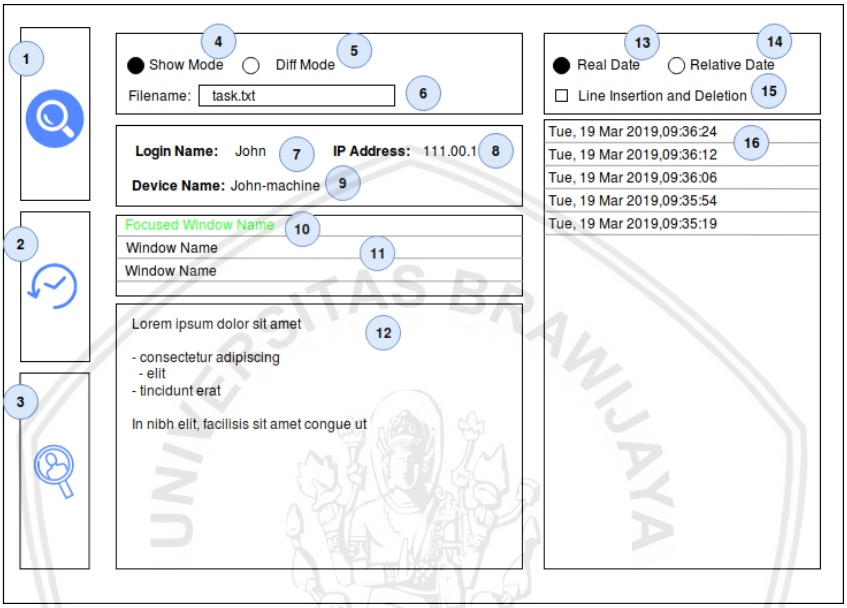
Gambar 5.22 Perancangan antarmuka tampilan *Tray* sistem *Lup Recorder*

Tabel 5.23 Perancangan antarmuka tampilan *Tray* sistem *Lup Recorder*

No	Nama Objek	Tipe	Keterangan
1	*Lup Recorder tray*	Tombol *tray*	Tombol untuk mengakses menu pada sistem *Lup Recorder*.
2	Tombol "*Start Recording*"	Tombol	Tombol memulai merekam tugas.
3	Tombol "*Set Interval*"	Tombol	Tombol mengubah interval rekaman.
4	Tombol "Stop and Quit"	Tombol	Tombol untuk menghentikan perekaman tugas.

5.1.3.2 Perancangan Antarmuka Tampilan Hasil Rekaman Pada Sistem *Lup Viewer*

Gambar 5.23 merupakan rancangan antarmuka tampilan hasil rekaman pada sistem *Lup Viewer*. Antarmuka ini merupakan rancangan antarmuka yang digunakan untuk melihat hasil rekaman. Penjelasan komponen-komponen yang terdapat pada rancangan tersebut terdapat pada Tabel 5.24.



Gambar 5.23 Perancangan antarmuka tampilan Hasil Rekaman sistem *Lup Viewer*

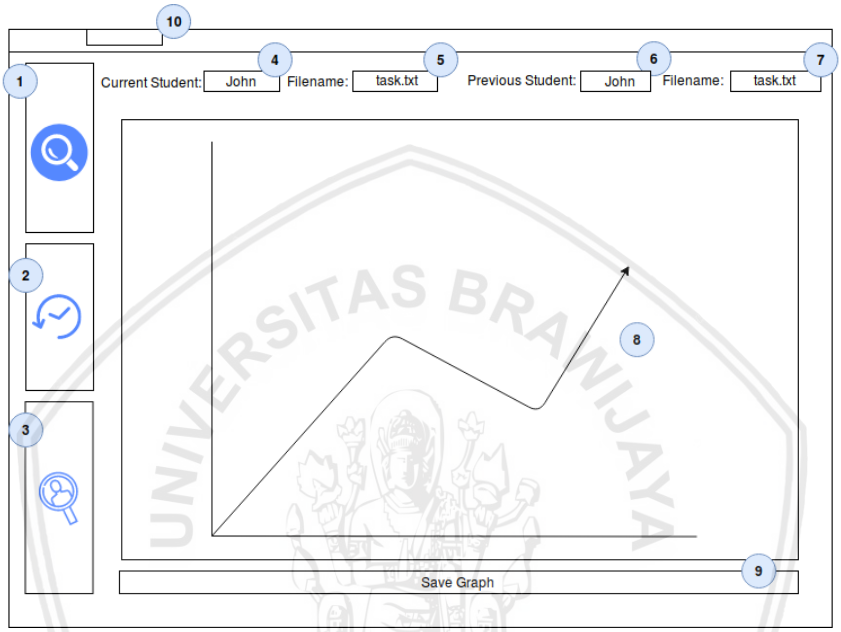
Tabel 5.24 Penjelasan antarmuka tampilan Hasil Rekaman sistem *Lup Viewer*

No	Nama Objek	Tipe	Keterangan
1	Tombol "*Main View*"	Tombol	Tombol untuk menuju tampilan utama seluruh daftar rekaman.
2	Tombol "*Log View*"	Tombol	Tombol untuk menuju tampilan hasil rekaman.
3	Tombol "*Search View*"	Tombol	Tombol untuk menuju tampilan analisis hasil rekaman.
4	Tombol "*Show Mode*"	*Radio Button*	*Radio button* untuk mengalihkan mode tampilan rekaman berkas ke mode *show*.

5	Tombol "*Diff Mode*"	*Radio Button*	*Radio button* untuk mengalihkan mode tampilan rekaman berkas ke mode *diff*.
6	Tombol "*Filename*"	*Combo Box*	*Combo box* untuk memilih nama berkas tugas.
7	Label "*Login Name*"	Label	Label untuk menampilkan nama *login*.
8	Label "*Device Name*"	Label	Label untuk menampilkan nama piranti.
9	Label "*IP Address*"	Label	Label untuk menampilkan alamat *IP*.
10	Daftar "*Active Window*"	*List Widget*	Daftar untuk menampilkan *window* yang aktif.
11	Daftar "*All Window*"	*List Widget*	*List widget* untuk menampilkan semua *window*.
12	*Widget* berkas rekaman	*Widget*	*Widget* untuk menampilkan rekaman berkas.
13	Tombol "*Real Date*"	*Radio Button*	*Radio button* untuk mengalihkan mode format waktu rekaman ke dalam format waktu *real*.
14	Tombol "*Relative Date*"	*Radio Button*	*Radio button* untuk mengalihkan mode format waktu rekaman ke dalam format waktu relatif.
15	Tombol "*Line Insertion and Deletion*"	*Check Box*	Tombol untuk menampilkan jumlah baris yang ditambah dan dihapus.
16	Daftar Rekaman	*List Widget*	*List widget* untuk menampilkan daftar rekaman.

5.1.3.3 Perancangan Antarmuka Tampilan *Edit-distance* Tugas Sebelumnya Pada Sistem *Lup Viewer*

Gambar 5.24 merupakan rancangan antarmuka tampilan *edit-distance* tugas sebelumnya pada sistem *Lup Viewer*. Rancangan antarmuka ini digunakan untuk melihat grafik *edit-distance* tugas sebelumnya. Penjelasan komponen-komponen yang terdapat pada rancangan tersebut terdapat pada Tabel 5.25.



Gambar 5.24 Perancangan antarmuka tampilan *Edit-distance* Tugas Sebelumnya

Tabel 5.25 Penjelasan antarmuka tampilan *Edit-distance* Tugas Sebelumnya

No	Nama Objek	Tipe	Keterangan
1	Tombol "*Main View*"	Tombol	Tombol untuk menuju tampilan utama seluruh daftar rekaman.
2	Tombol "*Log View*"	Tombol	Tombol untuk menuju tampilan hasil rekaman.
3	Tombol "*Search View*"	Tombol	Tombol untuk menuju tampilan analisis hasil rekaman.
4	Tombol "*Current Student*"	*Combo Box*	*Combo box* untuk memilih mahasiswa pada tugas saat ini.
5	Tombol "*Filename*"	*Combo Box*	*Combo box* untuk memilih berkas tugas saat ini.

6	Tombol "*Previous Student*"	*Combo Box*	*Combo box* untuk memilih mahasiswa pada tugas sebelumnya.
7	Tombol "*Filename*"	*Combo Box*	*Combo box* untuk memilih mahasiswa pada tugas sebelumnya.
8	*Widget Edit-distance*	*Widget*	*Widget* untuk menampilkan grafik *edit-distance*.
9	Tombol "*Save Graph*"	Tombol	Tombol untuk menyimpan grafik *edit-distance*.
10	Tombol "*Load Edit-distance File*"	Tombol	Tombol untuk membuka berkas *edit-distance* tugas sebelumnya.

5.2 Implementasi Sistem

Tahapan implementasi sistem merupakan sebuah tahapan yang dilakukan setelah selesainya proses perancangan sistem. Pada tahapan ini hasil perancangan komponen yang berupa *pseudocode* diimplementasikan menjadi *working code* menggunakan bahasa pemrograman *Python* dan bantuan kakas bantu seperti *git*. Selain itu, Hasil perancangan antarmuka juga diimplementasikan dengan bantuan *widget toolkit Qt*.

5.2.1 Spesifikasi Sistem

Spesifikasi perangkat keras yang digunakan untuk membangun sistem dipaparkan pada Tabel 5.26. Pada Tabel tersebut terdapat spesifikasi *processor*, *HDD* dan *RAM*. Sementara itu, spesifikasi perangkat lunak yang digunakan dipaparkan pada Tabel 5.27, di dalamnya terdapat spesifikasi sistem operasi, *kernel*, editor dokumentasi, editor pemrograman, dan bahasa pemrograman.

Tabel 5.26 Spesfikasi perangkat keras

Nama Komponen	Spesifikasi
Processor	Intel i5-7200U (4) @ 3.1GHz
Hard Disk	500 GB
RAM	4 GB

Tabel 5.27 Spesfikasi perangkat lunak

Nama Komponen	Spesifikasi
Sistem Operasi	Debian GNU/Linux 9.6 (stretch) x86_64
Kernel	Kernel: 4.9.0-6-amd64
Editor Dokumentasi	GNU Emacs 26.1
Editor Pemrograman	GNU Emacs 26.1
Bahasa Pemrograman	Python 3.6.4

5.2.2 Implementasi Kode Program

Pada tahapan implementasi kode program. Hasil rancangan komponen sistem pada tahapan perancangan komponen sistem diimplementasikan menjadi *working code* dengan bahasa pemrograman *Python*. Subbab 5.2.2.1 hingga 5.2.2.3 memaparkan implementasi dari tiga sampel *class* pada tahapan perancangan komponen sebelumnya, yaitu *class Controller* pada sistem *Lup Recorder*, *class LogController* pada sistem *Lup Viewer*, dan *class SearchController* pada sistem *Lup Viewer*.

5.2.2.1 Implementasi Kode Program *Class Controller* Pada Sistem *Lup Recorder*

Pada Tabel Kode 5.8 dipaparkan *source code get_all_windows*. *Source code get_all_windows* mengimplementasikan hasil rancangan *pseudocode* pada tahapan sebelumnya. *Source code get_all_windows* mengambil seluruh informasi *window* yang kemudian melakukan filter dan hanya mengembalikan nama *window* saja.

No	get_all_windows

```
1  def get_all_windows(self):
2    all_windows = ""
3    all_windows_proc = Popen(["wmctrl", "-l"], stdout=PIPE)
4    all_windows_dirty, err = all_windows_proc.communicate()
5    if all_windows_dirty:
6      all_windows_dirty = all_windows_dirty.splitlines()
7      for line in all_windows_dirty:
8        windows_name = line.split(None, 3)[-1].decode()
```

```
 9         all_windows += "{}\n".format(windows_name)
10     return all_windows
```

Tabel Kode 5.8 Source code *function get_all_windows*

5.2.2.2 Implementasi Kode Program *Class LogController* Pada Sistem *Lup Viewer*

Pada Tabel Kode 5.9 dipaparkan *source code populate_log*. *Source code populate_log* mengimplementasikan hasil rancangan *pseudocode* pada tahapan sebelumnya. *Source code populate_log* membaca seluruh rekaman mahasiswa, dari setiap rekaman diambil nilai-nilai yang dibutuhkan di antaranya adalah waktu rekaman, baris yang ditambah, dan baris yang dihapus.

No	populate_logs

```
 1  def populate_logs(self, selected_file=None):
 2    student_records = self._log_model.student_records
 3
 4    for record in student_records:
 5      relative_time = self._main_ctrl.relativize_datetime(
 6        record.committed_datetime
 7      )
 8      time_format = "{:%a, %d %b %Y, %H:%M:%S}"
 9      time = time_format.format(record.committed_datetime)
10      sha = record.hexsha
11      insertions = 0
12      deletions = 0
13
14      if selected_file:
15        if self._log_model.is_exists(selected_file, sha):
16          insertions =
            ↪  record.stats.files[selected_file]["insertions"]
17          deletions =
            ↪  record.stats.files[selected_file]["deletions"]
18
19      log = dict(
20        relative_time=relative_time,
21        time=time,
22        sha=sha,
23        insertions=insertions,
24        deletions=deletions,
```

```
25          )
26          yield log
```

Tabel Kode 5.9 Source code *function populate_logs*

5.2.2.3 Implementasi Kode Program *Class SearchController* Pada Sistem *Lup Viewer*

Pada Tabel Kode 5.9 dipaparkan *source code construct_ed_graph_path*. *Source code construct_ed_graph_path* mengimplementasikan hasil rancangan *pseudocode* pada tahapan sebelumnya. *Source code construct_ed_graph_path* membangun *path* untuk nama grafik *edit-distance* yang disimpan. *Path* tersebut mengikuti jumlah mahasiswa yang ada. Jika terdapat lebih dari satu mahasiswa, maka diberikan "_" sebagai penghubung nama antar mahasiswa.

No	construct_ed_graph_path
1	`def construct_ed_graph_path(self, *args):`
2	`  record_path = self._main_model.record_path`
3	
4	`  if len(args) == 1:`
5	`    graph_fmt = "{}.png".format(args[0])`
6	`  else:`
7	`    names = "_".join(args)`
8	`    graph_fmt = "{}.png".format(names)`
9	
10	`  graph_path = join(record_path, "lupv-notes", graph_fmt)`
11	`  return graph_path`

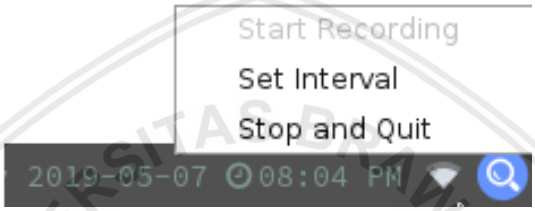
Tabel Kode 5.10 *Source code function construct_ed_graph_path*

5.2.3 Implementasi Antarmuka

Pada Tahapan ini, hasil rancangan antarmuka pada tahapan sebelumnya diimplementasikan menggunakan *widget toolkit Qt*. Tiga sampel implementasi antarmuka yang dipaparkan adalah antarmuka tampilan *tray* pada sistem *Lup Recorder* pada Gambar 5.25, antarmuka tampilan hasil rekaman pada sistem *Lup Viewer* pada Gambar 5.26, dan antarmuka tampilan *edit-distance* tugas sebelumnya pada sistem *Lup Viewer* pada Gambar 5.27.

5.2.3.1 Implementasi Antarmuka Tampilan *Tray* Pada Sistem *Lup Recorder*

Gambar 5.25 merupakan implementasi antarmuka tampilan *tray* pada sistem *Lup Recorder*. Implementasi ini dibangun berlandaskan tahapan perancangan antarmuka sebelumnya. Antarmuka ini merupakan rancangan antarmuka utama pada sistem *Lup Recoder* dan digunakan untuk mengakses seluruh menu utama sistem *Lup Recorder*.



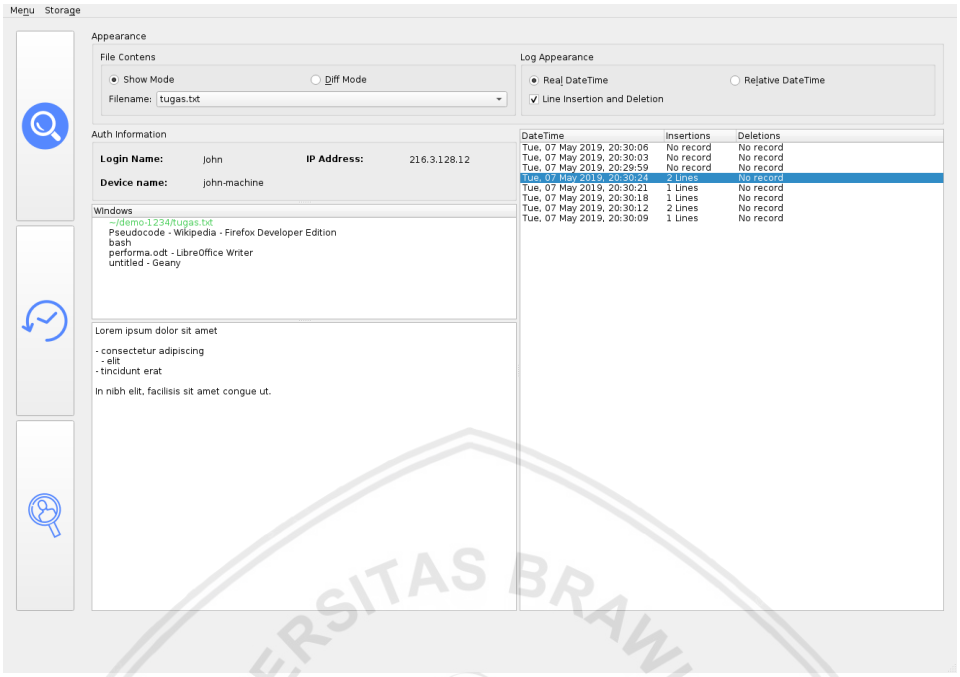
Gambar 5.25 Implementasi antarmuka tampilan *Tray* pada sistem *Lup Recorder*

5.2.3.2 Implementasi Antarmuka Tampilan Hasil Rekaman Pada Sistem *Lup Viewer*

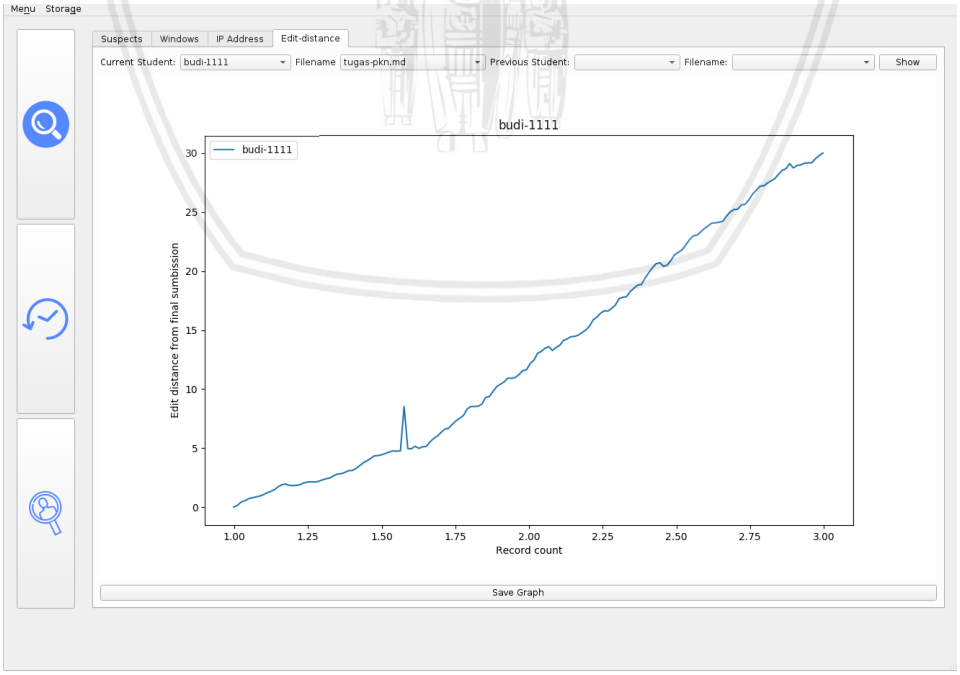
Gambar 5.26 merupakan implementasi antarmuka tampilan hasil rekaman pada sistem *Lup Viewer*. Implementasi ini dibangun berlandaskan tahapan perancangan antarmuka sebelumnya. Antarmuka ini merupakan rancangan antarmuka yang digunakan untuk melihat hasil rekaman.

5.2.3.3 Perancangan Antarmuka Tampilan *Edit-distance* Tugas Sebelumnya Pada Sistem *Lup Viewer*

Gambar 5.27 merupakan implementasi antarmuka tampilan *edit-distance* tugas sebelumnya pada sistem *Lup Viewer*. Implementasi ini dibangun berlandaskan tahapan perancangan antarmuka sebelumnya. Antarmuka ini digunakan untuk melihat grafik *edit-distance* tugas sebelumnya.



Gambar 5.26 Implementasi antarmuka tampilan Hasil Rekaman pada sistem *Lup Viewer*



Gambar 5.27 Perancangan antarmuka tampilan *Edit-distance* Tugas Sebelumnya pada sistem *Lup Viewer*

BAB 6 PENGUJIAN

6.1 Pengujian Unit

Pengujian unit dilakukan untuk memastikan hasil implementasi kode program telah sesuai dengan hasil perancangan komponen. Komponen terkecil dari sebuah sistem berorientasi objek adalah *class* dengan *testable unit* terkecilnya adalah *function* atau *method* (Pressman, 2010). Pengujian unit pada *class model* dan *class controller* dilakukan hingga mencapai 100% *code coverage*. *Class view* hanya memanggil fungsi-fungsi yang terdapat pada *class controller* dan *class model* sehingga pada penelitian ini pengujian unit pada *class view* hanya diuji hingga *code coverage* mencapai 70%. Metode pengujian yang digunakan adalah *white-box testing*, dan teknik pengujian yang digunakan adalah *basis path testing*. *Driver* digunakan sebagai *"main program"* untuk memanggil *function* yang diuji. *Stub* atau *fake* digunakan agar pengujian unit benar-benar terisolasi, bukan sebagai *"stand-alone program"*. Terdapat tiga sampel pengujian unit yang akan di paparkan pada tahapan ini. Gambar 6.28 memperlihatkan hasil pengujian unit pada *class controller* dan *class model* yang mencapai 100% *code coverage*.

```
Name                                Stmts   Miss  Cover
-----------------------------------------------------
Lupv/__init__.py                        0      0   100%
Lupv/controllers/__init__.py            0      0   100%
Lupv/controllers/log.py                64      0   100%
Lupv/controllers/main.py               54      0   100%
Lupv/controllers/search.py            172      0   100%
Lupv/models/__init__.py                 0      0   100%
Lupv/models/logs.py                    75      0   100%
Lupv/models/main.py                    42      0   100%
Lupv/models/search.py                  30      0   100%
-----------------------------------------------------
TOTAL                                 437      0   100%
======================== 91 passed in 6.45 seconds
```

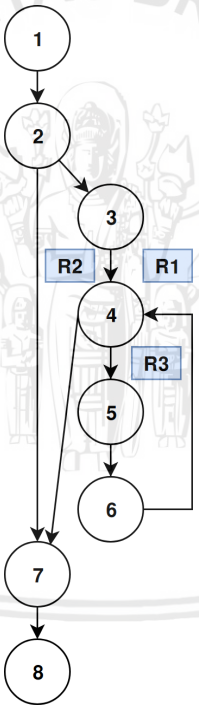
Gambar 6.28 Pengujian unit mencapai 100% *code coverage*

6.1.1 Pengujian Unit *Class Controller Function get_all_windows*

Pengujian *unit* dilakukan pada *function get_all_windows* secara terisolasi. Maka unit lain yang dipanggil di dalamnya akan digantikan dengan *stub*. Tabel Kode 6.11 dan Gambar 6.29 menjelaskan *basis path* dari *pseudocode function get_all_windows*. Bagian berikutnya menjelaskan perhitungan *cyclomatic complexity*, dan *independent path*. Tabel 6.28 menjelaskan hasil uji dan kasus uji dari *function get_all_windows*.

No	get_all_windows
1	START
2	READ all_windows_dirty
3	IF all_windows_dirty is true
4	SET all_windows_dirty to split every line
5	FOR each window in all_windows_dirty
6	GET window_name
7	SET all_windows = window_name
8	ENDFOR
9	ENDIF
10	RETURN all_windows
11	END

Tabel Kode 6.11 ***Pseudocode function get_all_windows***



Gambar 6.29 ***Flow graph*** **dari** ***pseudocode get_all_windows***

Cyclomatic Complexity

- V(G) = 3 *regions*
- V(G) = 9 *edge* – 8 *node* + 2 = 3
- V(G) = 2 *predicate node* + 1 = 3

Independent Path

- Jalur 1 : 1 - 2 - 7 - 8
- Jalur 2 : 1 - 2 - **3 - 4** - 7 - 8
- Jalur 2 : 1 - 2 - 3 - 4 - **5 - 6** - 4 - 7 - 8

Tabel 6.28 Kasus uji dan hasil uji *function get_all_windows*

Jalur	Prosedur Uji	*Expected Result*	*Result*	Status
1	1) *Set variable window_title* dengan nilai “None”. 2) *Class Driver TestController* memanggil *function get_all_windows*	*Return value function get_all_windows* bernilai “"”.	*As expe-cted*	Valid
2	1) *Set variable window_title* dengan nilai “"”. 2) *Class Driver TestController* memanggil *function get_all_windows*.	*Return value function get_all_windows* bernilai “"”.	*As expe-cted*	Valid
3	1) *Set variable window_title* dengan nilai “b"0x006000ab 0 machine-name foo_window_title "”. 2) *Class Driver TestController* memanggil *function get_all_windows*.	*Return value function get_all_windows* bernilai “foo_window_title”	*As expe-cted*	Valid

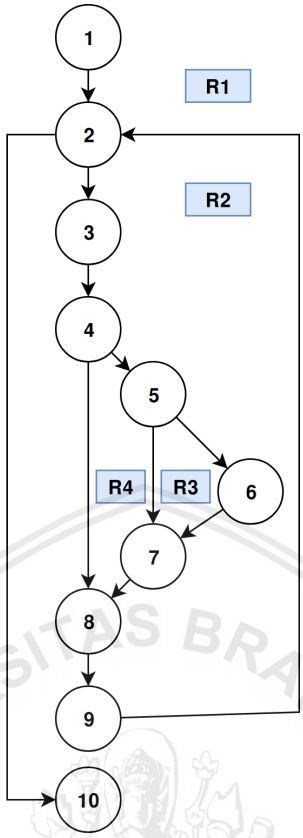
Pengujian unit yang dilakukan pada *function get_all_windows* menghasilkan perhitungan *cyclomatic complexity* dengan nilai 3 sehingga memiliki 3 *independent path* yang harus diuji. Hasil pengujian 3 *path* yang dipaparkan pada Tabel 6.28 bernilai valid.

6.1.2 Pengujian Unit *Class LogController Function populate_logs*

Pengujian *unit* dilakukan pada *function populate_logs* secara terisolasi. Maka *function student_record*, *function is_exist* pada unit *LogModel*, dan *function relativize_datetime* pada unit *MainController* yang dipanggil di dalamnya akan digantikan dengan *stub*. Tabel Kode 6.12 dan Gambar 6.30 menjelaskan *basis path* dari *pseudocode function populate_logs*. Bagian berikutnya menjelaskan perhitungan *cyclomatic complexity* dan *independent path*. Tabel 6.29 menjelaskan hasil uji dan kasus uji dari *function populate_logs*.

No	populate_logs	
1	START	
2	CALL log_model RETURNING student_records	1
3		
4	FOR each record in student_records	2
5	CALL main_ctrl with relativize_datetime \	3
6	RETURNING relative_time	
7	GET time	
8	GET sha	
9	SET insetion to 0	
10	SET deletion to 0	
11	IF selected_file is true	4
12	IF CALL log_model with is_exist \	
13	RETURNING true	5
14	GET insertion	6
15	GET deletion	
16	ENDIF	7
17	ENDIF	8
18	ENDFOR	9
19	SET log to (relative_time, time, sha,	10
20	insertion, deletion)	
21	RETURN log	
22	END	

Tabel Kode 6.12 *Pseudocode function populate_logs*



Gambar 6.30 *Flow graph* dari *pseudocode pupoulate_logs*

Cyclomatic Complexity

- V(G) = 4 *regions*
- V(G) = 12 *edge* – 10 *node* + 2 = 4
- V(G) = 3 *predicate node* + 1 = 4

Independent Path

- Jalur 1: 1 - 2 - 10
- Jalur 2: 1 - 2 - **3 - 4 - 8 - 9** - 2 - 10
- Jalur 3: 1 - 2 - 3 - 4 - **5 - 7** - 8 - 9 - 2 - 10
- Jalur 4: 1 - 2 - 3 - 4 - 5 - **6** - 7 - 8 - 9 - 2 - 10

Tabel 6.29 Kasus uji dan hasil uji *function populate_logs*

Jalur	Prosedur Uji	*Expected Result*	*Result*	Status
1	1) *Set variable student_records* dengan nilai "[]". 2) *Class Driver TestLogController* memanggil *function populate_logs*.	*Return value function populate_logs* bernilai "[]".	*As expe-cted*	Valid
2	*Class Driver TestLogController* memanggil *function populate_logs* tanpa *argument*.	*Return value function populate_logs* bernilai data rekaman tanpa nilai baris yang ditambah dan dihapus.	*As expe-cted*	Valid
3	*Class Driver TestLogController* memanggil *function populate_logs* dengan *argument* "selected_file="tugas-none.txt"".	*Return value function populate_logs* bernilai data rekaman tanpa nilai baris yang ditambah dan dihapus.	*As expe-cted*	Valid
4	*Class Driver TestLogController* memanggil *function populate_logs* dengan *argument* "selected_file="tugas-tif.txt"".	*Return value function populate_logs* bernilai data rekaman dengan nilai baris yang ditambah dan dihapus.	*As expe-cted*	Valid

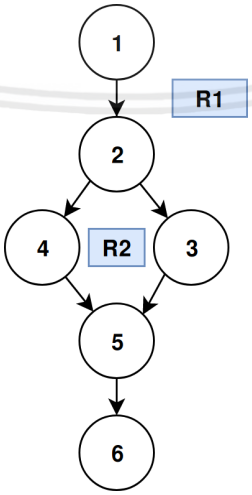
Pengujian unit yang dilakukan pada *function populate_logs* menghasilkan perhitungan *cyclomatic complexity* dengan nilai 4 sehingga memiliki 4 *independent path* yang harus diuji. Hasil pengujian 4 *path* yang dipaparkan pada Tabel 6.29 bernilai valid.

6.1.3 Pengujian Unit *Class SearchController Function construct_ed_graph_path*

Pengujian *unit* dilakukan pada *function construct_ed_graph_path* secara terisolasi. Maka unit *main_model* yang dipanggil di dalamnya akan digantikan dengan *stub*. Tabel Kode 6.13 dan Gambar 6.31 menjelaskan *basis path* dari *pseudocode function construct_ed_graph_path*. Bagian berikutnya menjelaskan perhitungan *cyclomatic complexity*, dan *independent path*. Tabel 6.30 menjelaskan hasil uji dan kasus uji dari *function construct_ed_graph_path*.

No	construct_ed_graph_path	
1	START	
2	CALL main_model RETURNING record_path	1
3		
4	IF passed argument == 1	2
5	SET graph_fmt to argument + ".png"	3
6	ELSE	
7	SET graph_fmt to argument + "_" + ".png"	4
8	ENDIF	5
9	SET graph_path record_path + "lupv_notes" \	
10	+ graph_fmt	6
11	Return graph_path	
12	END	

Tabel Kode 6.13 *Pseudocode function construct_ed_graph_path*



Gambar 6.31 *Flow graph* dari *pseudocode construct_ed_graph_path*

Cyclomatic Complexity

- V(G) = 2 *regions*
- V(G) = 6 *edge* – 6 *node* + 2 = 2
- V(G) = 1 *predicate node* + 1 = 2

Independent Path

- Jalur 1: 1 - 2 - 3 - 5 - 6
- Jalur 2: 1 - 2 - **4** - 5 - 6

Tabel 6.30 Kasus uji dan hasil uji *function construct_ed_graph_path*

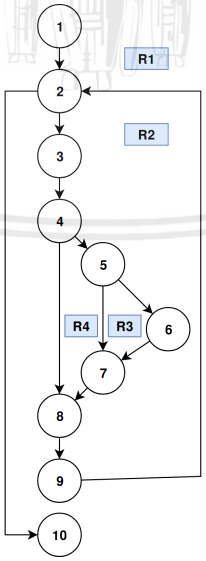
Jalur	Prosedur Uji	*Expected Result*	*Result*	Status
1	*Class Driver TestSearchController* memanggil *function construct_ed_graph_path* dengan *argument* "ani-1111".	*Return value function construct_ed_graph_path* bernilai "/lupv-notes/ani-1111.png".	*As expected*	Valid
2	*Class Driver TestSearchController* memanggil *function construct_ed_graph_path* dengan *argument* "ani-1111", "budi-2222".	*Return value function construct_ed_graph_path* bernilai "/lupv-notes/ani-1111_budi-2222.png".	*As expected*	Valid

Pengujian unit yang dilakukan pada *function construct_ed_graph_path* menghasilkan perhitungan *cyclomatic complexity* dengan nilai 2 sehingga memiliki 2 *independent path* yang harus diuji. Hasil pengujian 2 *path* yang dipaparkan pada Tabel 6.30 bernilai valid.

6.2 Pengujian Integrasi

Pengujian integrasi dilakukan untuk memastikan integrasi antar unit berjalan dengan baik. Pengujian integrasi dilakukan karena terdapat kemungkinan terjadinya masalah ketika unit dijalankan bersamaan. Sistem yang dikembangkan menggunakan pendekatan dan pengembangan berorientasi objek sehingga pendekatan pengujian integrasi yang digunakan adalah *use-based testing*. Pengujian integrasi diawali dari *class* yang memiliki *dependent class* paling sedikit, yaitu dimulai dari *class model*, kemudian *class controller*, dan yang terakhir adalah *class view*. Metode yang digunakan adalah *white-box testing*, dan teknik yang digunakan adalah *basis path testing*. Tujuan pengujian integrasi memastikan integrasi komponen sistem berjalan dengan baik. Maka pada pengujian ini, *function* pada suatu *class* yang memanggil *function* pada *class* lain tidak digantikan dengan *stub* atau *fake* sebagaimana ketika pada pengujian unit. Terdapat satu sampel pengujian integrasi yang dipaparkan, yaitu *function populate_logs* pada *class LogController* yang di dalamnya memanggil *function student_records* dan *function is_exist* pada *class LogModel*, dan *function relativize_datetime* pada *class MainController*.

Tabel Kode 6.14 dan Gambar 6.32 menjelaskan *basis path* dari *pseudocode function populate_logs_integration*. Bagian berikutnya menjelaskan perhitungan *cyclomatic complexity*, *independent path*. Tabel 6.31 menjelaskan hasil uji dan kasus uji dari *function populate_logs*.



Gambar 6.32 *Flow graph* dari *pseudocode pupoulate_logs*

No	populate_logs

```
1   START
2     CALL log_model RETURNING student_records
3
4     FOR each record in student_records
5       CALL main_ctrl with relativize_datetime \
6                           RETURNING relative_time
7       GET time
8       GET sha
9       SET insetion to 0
10      SET deletion to 0
11      IF selected_file is true
12        IF CALL log_model with is_exist \
13                              RETURNING true
14          GET insertion
15          GET deletion
16        ENDIF
17      ENDIF
18    ENDFOR
19    SET log to (relative_time, time, sha,
20                  insertion, deletion)
21    RETURN log
22  END
```

Tabel Kode 6.14 *Pseudocode function* populate_logs

Cyclomatic Complexity

- V(G) = 4 *regions*
- V(G) = 12 *edge* – 10 *node* + 2 = 4
- V(G) = 3 *predicate node* + 1 = 4

Independent Path

- Jalur 1: 1 - 2 - 10
- Jalur 2: 1 - 2 - **3 - 4 - 8 - 9** - 2 - 10
- Jalur 3: 1 - 2 - 3 - 4 - **5 - 7** - 8 - 9 - 2 - 10
- Jalur 4: 1 - 2 - 3 - 4 - 5 - **6** - 7 - 8 - 9 - 2 - 10

Tabel 6.31 Kasus uji dan hasil uji *function populate_logs*

Jalur	Prosedur Uji	*Expected Result*	*Result*	Status
1	1) *Set variable student_records* dengan nilai “[]”. 2) *Class Driver TestLogController* memanggil *function populate_logs*.	*Return value function populate_logs* bernilai “[]”.	*As expected*	Valid
2	*Class Driver TestLogController* memanggil *function populate_logs* tanpa *argument*.	*Return value function populate_logs* bernilai data rekaman tanpa nilai baris yang ditambah dan dihapus.	*As expected*	Valid
3	*Class Driver TestLogController* memanggil *function populate_logs* dengan *argument* “selected_file="tugas-none.txt"”.	*Return value function populate_logs* bernilai data rekaman tanpa nilai baris yang ditambah dan dihapus.	*As expected*	Valid
4	*Class Driver TestLogController* memanggil *function populate_logs* dengan *argument* “selected_file="tugas-tif.txt"”.	*Return value function populate_logs* bernilai data rekaman dengan nilai baris yang ditambah dan dihapus.	*As expected*	Valid

Pengujian integrasi yang dilakukan pada *function populate_logs* menghasilkan perhitungan *cyclomatic complexity* dengan nilai 4 sehingga memiliki 4 *independent path* yang harus diuji. Hasil pengujian 4 *path* yang dipaparkan pada Tabel 6.31 bernilai valid.

6.3 Pengujian Validasi

Pengujian validasi dilakukan untuk memastikan bahwa sistem yang dibangun sesuai dengan seluruh skenario kebutuhan yang didefinisikan. Pengujian validasi fokus terhadap *user-visible action* dan *user-recognizable output* dari sistem (Pressman, 2010). Dalam pengujian validasi metode pengujian yang digunakan adalah *black-box testing*. Teknik yang digunakan untuk memilih nilai masukan pada pengujian adalah teknik *boundary value analysis* dan *equvalence partitioning*, kedua teknik tersebut digunakan untuk menghindari *rigorous testing* atau mencoba semua nilai *input*.

6.3.1 Pengujian Validasi Merekam Pengerjaan Tugas

Pengujian validasi "Merekam Pengerjaan Tugas" dilakukan untuk menguji spesifikasi kebutuhan fungsional LUPR-F-01. Terdapat tiga kasus uji pada pengujian ini. Kasus uji pertama pada Tabel 6.32 memastikan bahwa sistem memulai perekaman tugas, kasus uji kedua pada Tabel 6.34 memastikan bahwa sistem berhenti dan keluar, dan kasus uji ketiga pada Tabel 6.33 memastikan sistem menutup dialog pemilihan direktori tugas

Tabel 6.32 Kasus uji dan hasil uji Merekam Pengerjaan Tugas

Kode Kebutuhan	LUPR-F-01
Nama Kasus Uji	Merekam Pengerjaan Tugas
Prosedur	1. Mengeklik tombol "*Start Recording*". 2. Memilih direktori tempat tugas. 3. Mengeklik tombol "*Choose*".
Expected Result	Sistem memulai perekaman pengerjaan tugas dan menampilkan pesan notifikasi "*Lup is recording*" menandakan bahwa perekaman telah dimulai.
Result	*As expected*
Status	Valid

Tabel 6.33 Kasus uji dan hasil uji Merekam Pengerjaan Tugas alternatif 1

Kode Kebutuhan	LUPR-F-01
Nama Kasus Uji	Merekam Pengerjaan Tugas alternatif 1
Prosedur	1. Mengeklik tombol "*Stop and Quit*".
Expected Result	Sistem berhenti dan keluar.

Result	*As expected*
Status	Valid

Tabel 6.34 Kasus uji dan hasil uji Merekam Pengerjaan Tugas alternatif 2

Kode Kebutuhan	LUPR-F-01
Nama Kasus Uji	Merekam Pengerjaan Tugas alternatif 2
Prosedur	1. Mengeklik tombol "*Start Recording*". 2. Memilih direktori tempat tugas. 3. Mengeklik tombol "*Cancel*".
Expected Result	Sistem menutup dialog pemilihan direktori tugas.
Result	*As expected*
Status	Valid

Hasil pengujian validasi "Merekam Pengerjaan Tugas" yang memiliki 3 kasus uji yaitu pada Tabel 6.32, Tabel 6.33, dan Tabel 6.34 menghasilkan nilai valid. Maka dapat dipastikan fungsionalitas "Merekam Pengerjaan Tugas" sudah sesuai dengan kebutuhan yang didefinisikan.

6.3.2 Pengujian Validasi Mengubah Interval Rekaman

Pengujian validasi "Mengubah Interval Rekaman" membutuhkan *range* nilai yang harus dimasukkan penguji untuk melakukan validasi. Agar terhindar dari *rigorous testing* digunakan teknik *boundary value analysis* dan *equvalence partitioning* untuk menentukan nilai yang dimasukkan dalam pengujian. Nilai yang valid adalah semua bilangan bulat di atas 0.

Kasus uji yang didapatkan dengan teknik *equivalence partitioning*:

1. Kasus uji *valid input*: "2"
2. Kasus uji *invalid input*: "kolom kosong"

Kasus uji yang didapatkan dengan teknik *boundary value analysis*:

1. Kasus uji *valid input*: "1"
2. Kasus uji *invalid input*: "0"

Terdapat dua kasus uji dengan teknik *equivalence partitioning*, pertama pada Tabel 6.35 dengan nilai *input* "2" memastikan bahwa sistem berhasil mengubah nilai interval, kedua pada Tabel 6.36 dengan mengosongkan kolom memastikan bahwa sistem menampilkan pesan peringatan. Teknik *boundary value analysis* juga menghasilkan dua kasus uji. Pertama pada Tabel 6.37 dengan nilai *input* "1" memastikan bahwa sistem berhasil mengubah nilai interval, kasus uji kedua pada Tabel 6.38 dengan nilai *input* "0" memastikan bahwa sistem menampilkan pesan peringatan.

Tabel 6.35 Kasus uji dan hasil uji Mengubah Interval Rekaman (*valid input equivalence partitioning*)

Kode Kebutuhan	LUPR-F-02
Nama Kasus Uji	Mengubah Interval Rekaman
Prosedur	1. Mengeklik tombol "*Set Interval*". 2. Mengisi nilai 2 pada dialog interval. 3. Mengeklik tombol "*Apply*".
Expected Result	Sistem mengubah nilai interval rekaman dan menampilkan pesan notifikasi "*Interval changed to 2*".
Result	*As expected*
Status	Valid

Tabel 6.36 Kasus uji dan hasil uji Mengubah Interval Rekaman alternatif 1 (*invalid input equivalence partitioning*)

Kode Kebutuhan	LUPR-F-02
Nama Kasus Uji	Mengubah Interval Rekaman alternatif 1
Prosedur	1. Mengeklik tombol "*Set Interval*". 2. Mengosongkan dialog interval. 3. Mengeklik tombol "*Apply*".
Expected Result	Sistem menampilkan pesan peringatan "*Interval must higher than 0*". Nilai interval rekaman tidak dirubah.
Result	*As expected*
Status	Valid

Tabel 6.37 Kasus uji dan hasil uji Mengubah Interval Rekaman (*valid input boundary value analysis*)

Kode Kebutuhan	LUPR-F-02
Nama Kasus Uji	Mengubah Interval Rekaman
Prosedur	1. Mengeklik tombol "*Set Interval*". 2. Mengisi nilai 1 pada dialog interval. 3. Mengeklik tombol "*Apply*".
Expected Result	Sistem mengubah nilai interval rekaman dan menampilkan pesan notifikasi "*Interval changed to 1*".
Result	*As expected*
Status	Valid

Tabel 6.38 Kasus uji dan hasil uji Mengubah Interval Rekaman alternatif 1 (*invalid input boundary value analysis*)

Kode Kebutuhan	LUPR-F-02
Nama Kasus Uji	Mengubah Interval Rekaman alternatif 1
Prosedur	1. Mengeklik tombol "*Set Interval*". 2. Mengisi nilai 0 pada dialog interval. 3. Mengeklik tombol "*Apply*".
Expected Result	Sistem menampilkan pesan peringatan "*Interval must higher than 0*". Nilai interval rekaman tidak dirubah.
Result	*As expected*
Status	Valid

Hasil pengujian validasi "Mengubah Interval Rekaman" yang memiliki 2 kasus uji dari teknik *equivalence partitioning* pada Tabel 6.35 dan Tabel 6.36 serta 2 kasus uji dengan teknik *boundary value analysis* pada Tabel 6.37 dan Tabel 6.38 menghasilkan nilai valid. Maka dapat dipastikan fungsionalitas "Mengubah Interval Rekaman" sudah sesuai dengan kebutuhan yang didefinisikan.

6.3.3 Pengujian Validasi Melihat Seluruh Daftar Rekaman

Pengujian validasi "Melihat Seluruh Daftar Rekaman" dilakukan untuk menguji spesifikasi kebutuhan fungsional LUPV-F-01. Terdapat tiga kasus uji pada pengujian ini. Kasus uji pertama pada Tabel 6.39 memastikan bahwa sistem menampilkan seluruh daftar rekaman mahasiswa, Tabel 6.40 memastikan bahwa sistem menu-

tup dialog pemilihan rekaman, dan kasus uji ketiga pada Tabel 6.41 memastikan sistem menampilkan pesan peringatan bahwa direktori berkas tidak valid.

Tabel 6.39 Kasus uji dan hasil uji Melihat Seluruh Daftar Rekaman

Kode Kebutuhan	LUPV-F-01
Nama Kasus Uji	Melihat Seluruh Daftar Rekaman
Prosedur	1. Mengeklik tombol "*Open Records*". 2. Memilih direktori rekaman. 3. Mengeklik tombol "*Choose*".
Expected Result	Sistem menampilkan seluruh daftar rekaman mahasiswa, dengan data yang ditampilkan meliputi nama, nim, total rekaman, waktu rekaman awal, waktu rekaman akhir dan durasi pengerjaan.
Result	*As expected*
Status	Valid

Tabel 6.40 Kasus uji dan hasil uji Melihat Seluruh Daftar Rekaman alternatif 1

Kode Kebutuhan	LUPV-F-01
Nama Kasus Uji	Melihat Seluruh Daftar Rekaman alternatif 1
Prosedur	1. Mengeklik tombol "*Open Records*". 2. Memilih direktori rekaman. 3. Mengeklik tombol "*Cancel*".
Expected Result	Sistem menutup dialog pemilihan rekaman dan berkas rekaman tidak dibuka.
Result	*As expected*
Status	Valid

Tabel 6.41 Kasus uji dan hasil uji Melihat Seluruh Daftar Rekaman alternatif 2

Kode Kebutuhan	LUPV-F-01
Nama Kasus Uji	Melihat Seluruh Daftar Rekaman alternatif 2
Prosedur	1. Mengeklik tombol "*Open Records*". 2. Memilih direktori rekaman yang tidak valid, yaitu tidak sesuai dengan spesifikasi kebutuhan kode LUPV-F-01-02 dan LUPV-F-01-03. 3. Mengeklik tombol "*Choose*".

Expected Result	Sistem menampilkan pesan peringatan "*Not a valid Task directory (baris baru) Contains invalid Task: «direktori yang tidak valid»*" dan berkas rekaman tidak dibuka.
Result	*As expected*
Status	Valid

Hasil pengujian validasi "Melihat Seluruh Daftar Rekaman" yang memiliki 3 kasus uji yaitu pada Tabel 6.39, Tabel 6.40, dan Tabel 6.41 menghasilkan nilai valid. Maka dapat dipastikan fungsionalitas "Melihat Seluruh Daftar Rekaman" sudah sesuai dengan kebutuhan yang didefinisikan.

6.3.4 Pengujian Validasi Melihat Hasil Rekaman

Pengujian validasi "Melihat Hasil Rekaman" dilakukan untuk menguji spesifikasi kebutuhan fungsional LUPV-F-02. Terdapat empat kasus uji pada pengujian ini. Kasus uji pertama pada Tabel 6.42 memastikan bahwa sistem menampilkan hasil rekaman, Tabel 6.43 memastikan bahwa sistem menampilkan pesan informasi bahwa tidak ada berkas yang dipilih, kasus uji ketiga pada Tabel 6.44 memastikan bahwa sistem menampilkan pesan informasi bahwa tidak ada rekaman pada suatu berkas, dan kasus uji keempat pada Tabel 6.45 memastikan bahwa sistem menampilkan pesan peringatan untuk memilih berkas tugas.

Tabel 6.42 Kasus uji dan hasil uji Melihat Hasil Rekaman

Kode Kebutuhan	LUPV-F-02
Nama Kasus Uji	Melihat Hasil Rekaman
Prosedur	1. Mengeklik *clickable* baris mahasiswa pada seluruh daftar rekaman. 2. Mengeklik *clickable* baris daftar rekaman. 3. Memilih nama berkas pada tombol *combo box* "*Filename*". 4. Memilih tombol "*Line Insertion and Deletion*". 5. Memilih tombol "*Show Mode*" dan "*Diff Mode*".

Expected Result	Sistem Menampilkan hasil rekaman dan mengalihkan tampilan berkas antar mode "*show*" dan "*diff*". Hasil rekaman yang ditampilkan terdiri dari tanggal dan waktu rekaman, isi perubahan berkas, jumlah baris yang dihapus dan ditambah pada berkas, alamat *IP*, seluruh *window*, *window* yang aktif, nama *login*, dan nama piranti.
Result	*As expected*
Status	Valid

Tabel 6.43 Kasus uji dan hasil uji Melihat Hasil Rekaman alternatif 1

Kode Kebutuhan	LUPV-F-02
Nama Kasus Uji	Melihat Hasil Rekaman alternatif 1
Prosedur	1. Mengeklik *clickable* baris mahasiswa pada seluruh daftar rekaman. 2. Mengeklik *clickable* baris daftar rekaman.
Expected Result	Sistem menampilkan pesan informasi pada berkas rekaman dengan pesan "*No file selected, please select one*".
Result	*As expected*
Status	Valid

Tabel 6.44 Kasus uji dan hasil uji Melihat Hasil Rekaman alternatif 2

Kode Kebutuhan	LUPV-F-02
Nama Kasus Uji	Melihat Hasil Rekaman alternatif 2
Prosedur	1. Mengeklik *clickable* baris mahasiswa pada seluruh daftar rekaman. 2. Mengeklik *clickable* baris daftar rekaman. 3. Memilih nama berkas pada tombol *combo box* "*Filename*".
Expected Result	Sistem menampilkan pesan informasi pada berkas rekaman dengan pesan "*No availiable record for «nama berkas» in this period*".
Result	*As expected*
Status	Valid

Tabel 6.45 Kasus uji dan hasil uji Melihat Hasil Rekaman alternatif 3

Kode Kebutuhan	LUPV-F-02
Nama Kasus Uji	Melihat Hasil Rekaman alternatif 3
Prosedur	1. Mengeklik *clickable* baris mahasiswa pada seluruh daftar rekaman. 2. Mengeklik *clickable* baris daftar rekaman. 3. Memilih tombol "*Line Insertion and Deletion*".
Expected Result	Sistem menampilkan pesan peringatan "*Please choose a file*".
Result	*As expected*
Status	Valid

Hasil pengujian validasi "Melihat Hasil Rekaman" yang memiliki 4 kasus uji yaitu pada Tabel 6.42, Tabel 6.43, Tabel 6.44 dan Tabel 6.45 menghasilkan nilai valid. Maka dapat dipastikan fungsionalitas "Melihat Hasil Rekaman" sudah sesuai dengan kebutuhan yang didefinisikan.

6.3.5 Pengujian Validasi Mencari Tersangka

Pengujian validasi "Mencari Tersangka" membutuhkan *range* nilai yang harus dimasukkan penguji untuk melakukan validasi. Agar terhindar dari *rigorous testing*, digunakan teknik *boundary value analysis* dan *equvalence partitioning* untuk menentukan nilai *input* yang dimasukkan dalam pengujian. Nilai yang valid adalah semua bilangan bulat di atas 0.

Kasus uji yang didapatkan dengan teknik *equivalence partitioning*:

- Kasus uji *valid input*: "2"
- Kasus uji *invalid input*: "kolom kosong"

Kasus uji yang didapatkan dengan teknik *boundary value analysis*:

- Kasus uji *valid input*: "1"
- Kasus uji *invalid input*: "0"

Terdapat dua kasus uji dengan teknik *equivalence partitioning*, pertama pada Tabel 6.46 dengan nilai *input* "2" memastikan bahwa sistem menampilkan hasil

pencarian, kasus uji kedua pada Tabel 6.47 dengan mengosongkan kolom memastikan bahwa sistem menampilkan pesan peringatan. Teknik *boundary value analysis* juga menghasilkan dua kasus uji. Pertama pada Tabel 6.48 dengan nilai *input* "1" memastikan bahwa sistem menampilkan hasil pencarian, kasus uji kedua pada Tabel 6.49 dengan nilai *input* "0" memastikan bahwa sistem menampilkan pesan peringatan.

Dua kasus uji lainnya terdapat pada Tabel 6.50 untuk memastikan bahwa sistem menampilkan informasi jika tidak ada *suspect* yang ditemukan, dan pada Tabel 6.51 untuk memastikan bahwa sistem menampilkan pesan peringatan jika tidak ada berkas yang dipilih.

Tabel 6.46 Kasus uji dan hasil uji Mencari Tersangka (*valid input equivalence partitioning*)

Kode Kebutuhan	LUPV-F-03
Nama Kasus Uji	Mencari Tersangka
Prosedur	1. Memilih nama berkas dari tombol *combo box* "*Filename*". 2. Mengisi nilai 2 pada kolom "*Insertion Limit*". 3. Mengeklik tombol "*Search*".
Expected Result	Sistem menampilkan hasil pencarian tersangka.
Result	*As expected*
Status	Valid

Tabel 6.47 Kasus uji dan hasil uji Mencari Tersangka alternatif 1 (*invalid input equivalence partitioning*)

Kode Kebutuhan	LUPV-F-03
Nama Kasus Uji	Mencari Tersangka alternatif 2
Prosedur	1. Memilih nama berkas dari tombol *combo box* "*Filename*". 2. Mengosongkan kolom "*Insertion Limit*". 3. Mengeklik tombol "*Search*".
Expected Result	Sistem menampilkan pesan peringatan "*Please set limit higher than 0. Above 10 is recommended*".
Result	*As expected*
Status	Valid

Tabel 6.48 Kasus uji dan hasil uji Mencari Tersangka (*valid input boundary value analysis*)

Kode Kebutuhan	LUPV-F-03
Nama Kasus Uji	Mencari Tersangka
Prosedur	1. Memilih nama berkas dari tombol *combo box* "*Filename*". 2. Mengisi nilai 1 pada kolom "*Insertion Limit*". 3. Mengeklik tombol "*Search*".
Expected Result	Sistem menampilkan hasil pencarian tersangka.
Result	*As expected*
Status	Valid

Tabel 6.49 Kasus uji dan hasil uji Mencari Tersangka alternatif 1 (*invalid input boundary value analysis*)

Kode Kebutuhan	LUPV-F-03
Nama Kasus Uji	Mencari Tersangka alternatif 2
Prosedur	1. Memilih nama berkas dari tombol *combo box* "*Filename*". 2. Mengisi nilai 0 pada kolom "*Insertion Limit*". 3. Mengeklik tombol "*Search*".
Expected Result	Sistem menampilkan pesan peringatan "*Please set limit higher than 0. Above 10 is recommended*".
Result	*As expected*
Status	Valid

Tabel 6.50 Kasus uji dan hasil uji Mencari Tersangka alternatif 2

Kode Kebutuhan	LUPV-F-03
Nama Kasus Uji	Mencari Tersangka alternatif 2
Prosedur	1. Memilih nama berkas dari tombol *combo box* "*Filename*". 2. Mengisi nilai 1000 pada kolom "*Insertion Limit*". 3. Mengeklik tombol "*Search*".
Expected Result	Sistem menampilkan pesan informasi "*No suspect found*".

Result	*As expected*
Status	Valid

Tabel 6.51 Kasus uji dan hasil uji Mencari Tersangka alternatif 3

Kode Kebutuhan	LUPV-F-03
Nama Kasus Uji	Mencari Tersangka alternatif 3
Prosedur	1. Mengisi nilai bilangan bulat di atas 0 pada kolom "*Insertion Limit*". 2. Mengeklik tombol "*Search*".
Expected Result	Sistem menampilkan pesan peringatan "*Please select a file*".
Result	*As expected*
Status	Valid

Hasil pengujian validasi "Mencari tersangka" memiliki 6 kasus uji. Keenam kasus uji pada Tabel 6.46, Tabel 6.47, Tabel 6.48, Tabel 6.49, Tabel 6.50, dan Tabel 6.51 menghasilkan nilai valid. Maka dapat dipastikan fungsionalitas "Mencari tersangka" sudah sesuai dengan kebutuhan yang didefinisikan.

6.3.6 Pengujian Validasi Mencari *Window*

Pengujian validasi "Mencari *Window*" dilakukan untuk menguji spesifikasi kebutuhan fungsional LUPV-F-04. Terdapat tiga kasus uji pada pengujian ini. Kasus uji pertama pada Tabel 6.52 memastikan bahwa sistem menampilkan hasil pencarian *window*, kasus uji kedua pada Tabel 6.53 memastikan bahwa sistem menampilkan pesan informasi bahwa pencarian *window* tidak ditemukan, dan kasus uji ketiga pada Tabel 6.54 memastikan sistem menampilkan pesan peringatan bahwa kolom pencarian *window* kosong.

Tabel 6.52 Kasus uji dan hasil uji Mencari *Window*

Kode Kebutuhan	LUPV-F-04
Nama Kasus Uji	Mencari *Window*
Prosedur	1. Mengisi nama window yang ingin dicari pada kolom "*Window Name*". 2. Mengeklik tombol "*Search*".
Expected Result	Sistem menampilkan hasil pencarian *window*.

Result	*As expected*
Status	Valid

Tabel 6.53 Kasus uji dan hasil uji Mencari *Window* alternatif 1

Kode Kebutuhan	LUPV-F-04
Nama Kasus Uji	Mencari *Window* alternatif 1
Prosedur	1. Mengisi "11111111" pada kolom "*Window Name*". 2. Mengeklik tombol "*Search*".
Expected Result	Sistem menampilkan pesan informasi "*No window name for "11111111" found*".
Result	*As expected*
Status	Valid

Tabel 6.54 Kasus uji dan hasil uji Mencari *Window* alternatif 2

Kode Kebutuhan	LUPV-F-04
Nama Kasus Uji	Mencari *Window* alternatif 2
Prosedur	1. Mengeklik tombol "*Search*".
Expected Result	Sistem menampilkan pesan peringatan "*Please supply the window name*".
Result	*As expected*
Status	Valid

Hasil pengujian validasi "Mencari *Window*" yang memiliki 3 kasus uji yaitu pada Tabel 6.52, Tabel 6.53, dan Tabel 6.54 menghasilkan nilai valid. Maka dapat dipastikan fungsionalitas "Mencari *Window*" sudah sesuai dengan kebutuhan yang didefinisikan.

6.3.7 Pengujian Validasi Melihat Seluruh Alamat *IP*

Pengujian validasi "Melihat Seluruh Alamat *IP*" dilakukan untuk menguji spesifikasi kebutuhan fungsional LUPV-F-06. Terdapat satu kasus uji pada pengujian ini, yaitu kasus uji pada Tabel 6.55 memastikan bahwa sistem menampilkan seluruh alamat *IP*.

Tabel 6.55 Kasus uji dan hasil uji Melihat Seluruh Alamat *IP*

Kode Kebutuhan	LUPV-F-05
Nama Kasus Uji	Melihat Seluruh Alamat *IP*
Prosedur	1. Mengeklik tombol "*Show all IP addresses*".
Expected Result	Sistem menampilkan seluruh alamat *IP* mahasiswa.
Result	*As expected*
Status	Valid

Hasil pengujian validasi "Melihat Seluruh Alamat *IP*" yang memiliki 1 kasus uji yaitu pada Tabel 6.55 menghasilkan nilai valid. Maka dapat dipastikan fungsionalitas "Melihat Seluruh Alamat *IP*" sudah sesuai dengan kebutuhan yang didefinisikan.

6.3.8 Pengujian Validasi Melihat Grafik *Edit-distance*

Pengujian validasi "Melihat Grafik *Edit-distance*" dilakukan untuk menguji spesifikasi kebutuhan fungsional LUPV-F-06. Terdapat dua kasus uji pada pengujian ini. Kasus uji pertama pada Tabel 6.56 memastikan bahwa sistem menampilkan grafik *edit-distance* dari tugas mahasiswa, kasus uji kedua pada Tabel 6.57 memastikan bahwa sistem menampilkan pesan peringatan bahwa salah satu tombol *combo box* nama mahasiswa dan berkas tugas belum dipilih.

Tabel 6.56 Kasus uji dan hasil uji Melihat Grafik *Edit-distance*

Kode Kebutuhan	LUPV-F-06
Nama Kasus Uji	Melihat Grafik *Edit-distance*
Prosedur	1. Memilih nama mahasiswa dari tombol *combo box* "*Current Student*". 2. Memilih nama berkas dari tombol *combo box* "*Filename*". 3. Mengeklik tombol "*Show Mode*".
Expected Result	Sistem menampilkan grafik *edit-distance* dari tugas mahasiswa.
Result	*As expected*
Status	Valid

Tabel 6.57 Kasus uji dan hasil uji Melihat Grafik *Edit-distance* alternatif 1

Kode Kebutuhan	LUPV-F-06
Nama Kasus Uji	Melihat Grafik *Edit-distance* alternatif 1
Prosedur	1. Mengeklik tombol "*Show Mode*".
Expected Result	Sistem menampilkan pesan peringatan "*Please fill one of the pairs*".
Result	*As expected*
Status	Valid

Hasil pengujian validasi "Melihat Grafik *Edit-distance*" yang memiliki 2 kasus uji yaitu pada Tabel 6.56, dan Tabel 6.57 menghasilkan nilai valid. Maka dapat dipastikan fungsionalitas "Melihat Grafik *Edit-distance*" sudah sesuai dengan kebutuhan yang didefinisikan.

6.3.9 Pengujian Validasi Melihat Grafik *Edit-distance* Tugas Sebelumnya

Pengujian validasi "Melihat Grafik *Edit-distance* Tugas Sebelumnya" dilakukan untuk menguji spesifikasi kebutuhan fungsional LUPV-F-07. Terdapat empat kasus uji pada pengujian ini. Kasus uji pertama pada Tabel 6.58 memastikan bahwa sistem menampilkan grafik *edit-distance* dari tugas sebelumnya, kasus uji kedua pada Tabel 6.59 memastikan bahwa sistem menutup dialog pencarian berkas *edit-distance*, kasus uji ketiga pada Tabel 6.60 memastikan bahwa sistem menampilkan pesan peringatan bahwa berkas *edit-distance* belum dibuka, dan kasus uji keempat pada Tabel 6.61 memastikan bahwa salah satu tombol *combo box* nama mahasiswa atau berkas tugas belum dipilih.

Tabel 6.58 Kasus uji dan hasil uji Melihat Grafik *Edit-distance* Tugas Sebelumnya

Kode Kebutuhan	LUPV-F-07
Nama Kasus Uji	Melihat Grafik *Edit-distance* Tugas Sebelumnya

Prosedur	1. Mengeklik tombol "*Load Edit-distance File*". 2. Memilih berkas *edit-distance* tugas sebelumnya pada dialog pencarian berkas. 3. Mengeklik tombol "*Open*". 4. Memilih nama mahasiswa dari tombol *combo box* "*Previous Student*". 5. Memilih nama berkas dari tombol *combo box* "*Filename*". 6. Mengeklik tombol "*Show Mode*".
Expected Result	Sistem menampilkan grafik *edit-distance* tugas sebelumnya.
Result	*As expected*
Status	Valid

Tabel 6.59 Kasus uji dan hasil uji Melihat Grafik *Edit-distance* Tugas Sebelumnya alternatif 1

Kode Kebutuhan	LUPV-F-07
Nama Kasus Uji	Melihat Grafik *Edit-distance* Tugas Sebelumnya alternatif 1.
Prosedur	1. Mengeklik tombol "*Load Edit-distance File*". 2. Memilih berkas *edit-distance* tugas sebelumnya pada dialog pencarian berkas. 3. Mengeklik tombol "*Cancel*'.
Expected Result	Sistem menutup dialog pencarian berkas *edit-distance* tugas sebelumnya.
Result	*As expected*
Status	Valid

Tabel 6.60 Kasus uji dan hasil uji Melihat Grafik *Edit-distance* Tugas Sebelumnya alternatif 2

Kode Kebutuhan	LUPV-F-07
Nama Kasus Uji	Melihat Grafik *Edit-distance* Tugas Sebelumnya alternatif 2

Prosedur	1. Memilih nama mahasiswa dari tombol *combo box* "*Previous Student*". 2. Mengeklik tombol "*Show Mode*".
Expected Result	Sistem menampilkan pesan peringatan "*Please load edit-distance file first*".
Result	*As expected*
Status	Valid

Tabel 6.61 Kasus uji dan hasil uji Melihat Grafik *Edit-distance* Tugas Sebelumnya alternatif 3

Kode Kebutuhan	LUPV-F-07
Nama Kasus Uji	Melihat Grafik *Edit-distance* Tugas Sebelumnya alternatif 3.
Prosedur	1. Mengeklik tombol "*Load Edit-distance File*". 2. Memilih berkas *edit-distance* tugas sebelumnya pada dialog pencarian berkas. 3. Mengeklik tombol "*Open*". 4. Mengeklik tombol "*Show Mode*".
Expected Result	Sistem menampilkan pesan peringatan "*Please fill one of the pairs*".
Result	*As expected*
Status	Valid

Hasil pengujian validasi "Melihat Grafik *Edit-distance* Tugas Sebelumnya" yang memiliki 4 kasus uji yaitu pada Tabel 6.58, Tabel 6.59, Tabel 6.60, dan Tabel 6.61 menghasilkan nilai valid. Maka dapat dipastikan fungsionalitas "Melihat Grafik *Edit-distance* Tugas Sebelumnya" sudah sesuai dengan kebutuhan yang didefinisikan.

6.3.10 Pengujian Validasi Menyimpan Grafik *Edit-distance*

Pengujian validasi "Menyimpan Grafik *Edit-distance*" dilakukan untuk menguji spesifikasi kebutuhan fungsional LUPV-F-08. Terdapat satu kasus uji pada pengujian ini, yaitu kasus uji pada Tabel 6.62 memastikan bahwa sistem menyimpan grafik *edit-distance*.

Tabel 6.62 Kasus uji dan hasil uji Menyimpan Grafik *Edit-distance*

Kode Kebutuhan	LUPV-F-08
Nama Kasus Uji	Menyimpan Grafik *Edit-distance*
Prosedur	1. Mengeklik tombol "*Save Graph*".
Expected Result	Sistem menyimpan grafik *edit-distance* dengan nama "«namamahasiswa-nim».png" dan menampilkan pesan informasi "*Graph saved to «lokasi berkas»*" menandakan bahwa berkas berhasil disimpan.
Result	*As expected*
Status	Valid

Hasil pengujian validasi "Menyimpan Grafik *Edit-distance*" yang memiliki 1 kasus uji yaitu pada Tabel 6.62 menghasilkan nilai valid. Maka dapat dipastikan fungsionalitas "Menyimpan Grafik *Edit-distance*" sudah sesuai dengan kebutuhan yang didefinisikan.

6.3.11 Pengujian Validasi Mengekspor Semua Nilai *Edit-distance*

Pengujian validasi "Mengekspor Semua Nilai *Edit-distance*" dilakukan untuk menguji spesifikasi kebutuhan fungsional LUPV-F-09. Terdapat dua kasus uji pada pengujian ini. Kasus uji pertama pada Tabel 6.63 memastikan bahwa sistem mengekspor semua nilai *edit-distance*, dan kasus uji kedua pada Tabel 6.64 memastikan bahwa sistem menutup dialog pemilihan berkas dan nilai *edit-distance* tidak diekspor.

Tabel 6.63 Kasus uji dan hasil uji Mengekspor Semua Nilai *Edit-distance*

Kode Kebutuhan	LUPV-F-09
Nama Kasus Uji	Mengekspor Semua Nilai *Edit-distance*
Prosedur	1. Mengeklik tombol "*Export Edit-distance*". 2. Mengeklik tombol "*Export*".
Expected Result	Sistem mengekspor semua nilai *edit-distance* pada berkas dengan nama "«namatugas»-editdistance.lup", dan menampilkan pesan informasi "*Edit-distance exported to «lokasi berkas»*".
Result	*As expected*
Status	Valid

Tabel 6.64 Kasus uji dan hasil uji Mengekspor Semua Nilai *Edit-distance* alternatif 1

Kode Kebutuhan	LUPV-F-09
Nama Kasus Uji	Mengekspor Semua Nilai *Edit-distance* alternatif 1
Prosedur	1. Mengeklik tombol "*Export Edit-distance*". 2. Mengeklik tombol "*Cancel*".
Expected Result	Sistem menutup dialog pemilihan berkas dan nilai *edit-distance* tidak diekspor.
Result	*As expected*
Status	Valid

Hasil pengujian validasi "Mengekspor Semua Nilai *Edit-distance*" yang memiliki 2 kasus uji yaitu pada Tabel 6.63, dan Tabel 6.64 menghasilkan nilai valid. Maka dapat dipastikan fungsionalitas "Mengekspor Semua Nilai *Edit-distance*" sudah sesuai dengan kebutuhan yang didefinisikan.

6.3.12 Pengujian Validasi Mengalihkan Format Waktu

Pengujian validasi "Mengalihkan Format Waktu" dilakukan untuk menguji spesifikasi kebutuhan fungsional LUPV-F-10. Terdapat satu kasus uji pada pengujian ini, yaitu kasus uji pada Tabel 6.65 memastikan bahwa sistem mengalihkan format waktu.

Tabel 6.65 Kasus uji dan hasil uji Mengalihkan Format Waktu

Kode Kebutuhan	LUPV-F-10
Nama Kasus Uji	Mengalihkan Format Waktu
Prosedur	1. Mengeklik tombol "*Real DateTime*" dan "*Relative DateTime*".
Expected Result	Sistem mengalihkan format waktu.
Result	*As expected*
Status	Valid

Hasil pengujian validasi "Mengalihkan Format Waktu" yang memiliki 1 kasus uji yaitu pada Tabel 6.65 menghasilkan nilai valid. Maka dapat dipastikan fungsionalitas "Mengalihkan Format Waktu" sudah sesuai dengan kebutuhan yang didefinisikan.

6.4 *Automated Testing*

Automated testing digunakan untuk mempercepat proses pengujian. Pada tahapan ini dibangun *script* yang menjalankan kasus uji secara otomatis. Kasus uji didapatkan dari pengujian unit dan pengujian integrasi yang telah dilakukan pada tahapan sebelumnya. *Script* dibangun menggunakan kakas bantu *Pytest*, dan dijalankan secara otomatis dengan kakas bantu *Gitlab-CI*. Terdapat tiga sampel yang dipaparkan, yaitu *script automated testing* untuk *function get_all_windows* pada *class Controller*, *function populate_log* pada *class LogControleIr*, dan *function construct_ed_graph_path* pada *class SearchController*. Gambar 6.33 menunjukkan seluruh hasil *automated testing* yang berjalan dengan nilai "*PASSED*" tanpa ada galat.

```
tests/test_search_controller.py::TestSearchController::test_student_directories_iterator PASSED
tests/test_search_controller.py::TestSearchController::test_record_iterator PASSED
tests/test_search_controller.py::TestSearchController::test_analyze_suspects PASSED
tests/test_search_controller.py::TestSearchController::test_group_by_name PASSED
tests/test_search_controller.py::TestSearchController::test_get_suspects PASSED
tests/test_search_controller.py::TestSearchController::test_get_student_ips PASSED
tests/test_search_controller.py::TestSearchController::test_group_by_ip PASSED
tests/test_search_controller.py::TestSearchController::test_multigroup_child PASSED
tests/test_search_controller.py::TestSearchController::test_get_student_ip_groups PASSED
tests/test_search_controller.py::TestSearchController::test_idx_of_substring PASSED
tests/test_search_controller.py::TestSearchController::test_collect_student_windows PASSED
tests/test_search_controller.py::TestSearchController::test_get_student_windows PASSED
tests/test_search_controller.py::TestSearchController::test_populate_student_dirs PASSED
tests/test_search_controller.py::TestSearchController::test_load_prev_editdistances PASSED
tests/test_search_controller.py::TestSearchController::test_get_prev_student_names PASSED
tests/test_search_controller.py::TestSearchController::test_get_prev_filename_sample PASSED
tests/test_search_controller.py::TestSearchController::test_calc_prev_editdistances PASSED
tests/test_search_controller.py::TestSearchController::test__get_student_records PASSED
tests/test_search_controller.py::TestSearchController::test_calc_editdistances PASSED
tests/test_search_controller.py::TestSearchController::test_create_lupvnotes_dir PASSED
tests/test_search_controller.py::TestSearchController::test_construct_ed_graph_path PASSED
tests/test_search_controller.py::TestSearchController::test_construct_editistance_path PASSED
tests/test_search_controller.py::TestSearchController::test_export_editdistance PASSED

========================================== 91 passed in 8.62 seconds ===========================
```

Gambar 6.33 Seluruh hasil *automated testing*

6.4.1 *Test Script Automated Testing Class Controller Function get_all_windows*

Tabel Kode 6.15 memaparkan *script* yang digunakan untuk *automated testing* pada *function get_all_windows*. Pengujian dilakukan secara terisolasi sehingga dilakukan *stub* pada proses *Popen*. *Driver* kemudian memanggil *function get_all_windows* dan melakukan *assert* pada nilai yang dikembalikan.

No	get_all_windows_with_fake
1	`def test_get_all_windows_with_fake(self, ctrl, monkeypatch):`
2	`"""Test get_all_windows with Fake Object."""`

```
3     window_title =
      ↪ b"0x006000ab  0 machine-name foo_window_title"
4
5     def fake_communicate(input=None, timeout=None):
6         return window_title, "err"
7
8     Lupr.controllers.controller.Popen = FakePopen
9     Lupr.controllers.controller.Popen.communicate =
      ↪ fake_communicate
10
11    output = ctrl.get_all_windows()
12    assert output == "foo_window_title\n"
```

Tabel Kode 6.15 *Test script function* get_all_windows_with_fake

Hasil pengujian *function get_all_windows* dengan menggunakan *script automated testing* pada Tabel Kode 6.15 menghasilkan nilai "*PASSED*" seperti yang terlihat pada Gambar 6.33. Maka dapat dipastikan penggunaan *script automated testing* dapat mempercepat proses pengujian.

6.4.2 *Test Script Automated Testing Class LogController Function populate_logs*

Tabel Kode 6.16 memaparkan *script* yang digunakan untuk *automated testing* pada *function populate_logs*. Pengujian dilakukan secara terisolasi sehingga dilakukan *stub* pada *function is_exist* dan *function student_records*. *Driver* kemudian memanggil *function populate_logs* dan melakukan *assert* pada nilai yang dikembalikan.

No	populate_logs
1	`def test_populate_logs_no_records(self, log_ctrl_model):`
2	`  """Test populating student logs using dummy data."""`
3	`  log_ctrl, log_model = log_ctrl_model`
4	`  log_model.student_records = []`
5	`  log_model.is_exists = cf.fake_is_exists`
6	
7	`  ani_logs = list(`
8	`    log_ctrl.populate_logs(`
9	`      selected_file="tugas-tif.txt"`

```
10      ))
11
12    assert ani_logs == []
```

Tabel Kode 6.16 *Test script function* populate_logs

Hasil pengujian *function populate_logs* dengan menggunakan *script automated testing* pada Tabel Kode 6.16 menghasilkan nilai "*PASSED*" seperti yang terlihat pada Gambar 6.33. Maka dapat dipastikan penggunaan *script automated testing* dapat mempercepat proses pengujian.

6.4.3 *Test Script Automated Testing Class SearchController Function construct_ed_graph_path*

Tabel Kode 6.17 memaparkan *script* yang digunakan untuk *automated testing* pada *function construct_ed_graph_path*. *Function construct_ed_graph_path* tidak memanggil *function* lain sehingga tidak diperlukan adanya *stub*. *Driver* kemudian memanggil *function construct_ed_graph_path* dan melakukan *assert* pada nilai yang dikembalikan.

No	construct_ed_graph_path
1	`def test_construct_ed_graph_path(self, search_ctrl):`
2	`  """Test constructing editdistance graph path."""`
3	`  graph_path = search_ctrl.construct_ed_graph_path(`
4	`    "ani-1111")`
5	`  graph_path_2 = search_ctrl.construct_ed_graph_path(`
6	`   "ani-1111", "budi-2222")`
7	
8	`  assert "/lupv-notes/ani-1111.png" in graph_path`
9	`  assert "/lupv-notes/ani-1111_budi-2222.png" in`
	`  ↪  graph_path_2`

Tabel Kode 6.17 *Test script function* construct_ed_graph_path

Hasil pengujian *function construct_ed_graph_path* dengan menggunakan *script automated testing* pada Tabel Kode 6.17 menghasilkan nilai "*PASSED*" seperti yang terlihat pada Gambar 6.33. Maka dapat dipastikan penggunaan *script automated testing* dapat mempercepat proses pengujian.

6.5 Pengujian *Compatibility*

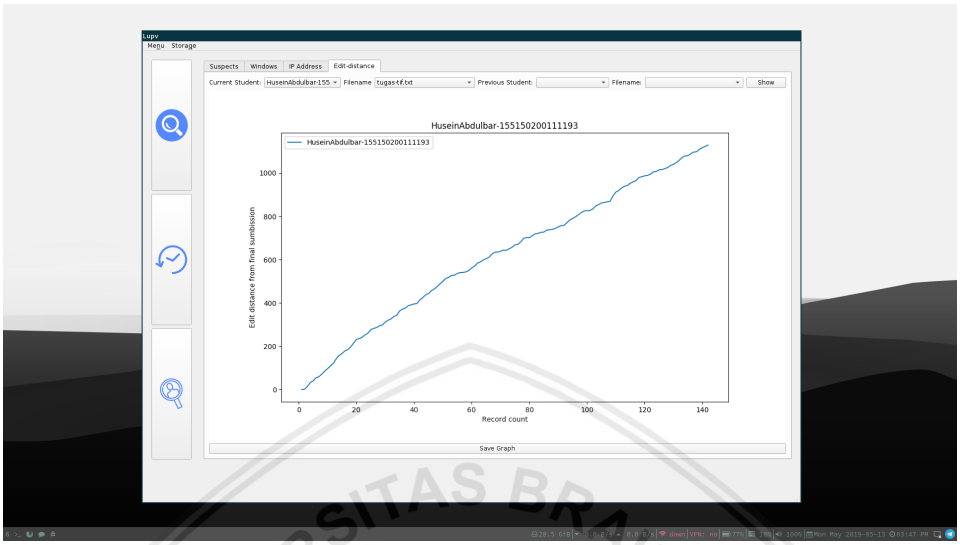
Pengujian *compatibility* dilakukan untuk memastikan aplikasi yang dibangun dapat berjalan dengan baik dan dapat menangani perbedaan format nama *window* pada enam *desktop environment* utama yang ada pada sistem operasi *GNU/Linux*. Enam *desktop environment* utama tersebut adalah *GNOME, KDE, XFCE, Unity, Cinnamon* dan *Phanteon*. Pengujian *compatibility* pada sistem *Lup Recorder* dilakukan dengan menjalankan sistem pada seluruh *desktop environment* yang tertera dalam Tabel 6.66, pengujian dilakukan oleh penguji yang bersangkutan dalam tabel yang sama. Terlihat pada Tabel 6.66 sistem juga diujikan terhadap dua *desktop environment* tambahan yaitu *OpenBox* dan *i3wm*. Gambar 6.34 menunjukkan seluruh hasil rekaman sistem *Lup Recorder* yang dapat dibaca oleh sistem *Lup Viewer*. Hal ini menandakan bahwa sistem *Lup Viewer* dapat berjalan dengan baik pada seluruh *desktop environment* dalam Tabel 6.66. Pengujian sistem *Lup Viewer* dilakukan dengan menjalankannya pada *desktop environment* yang berbeda-beda. Gambar 6.35 menunjukkan sistem *Lup Viewer* dapat berjalan dengan baik pada *desktop environment i3wm* dan Gambar 6.36 menunjukkan sistem *Lup Viewer* berjalan dengan baik pada *desktop environment XFCE*.

Tabel 6.66 Daftar *desktop environment* yang diuji

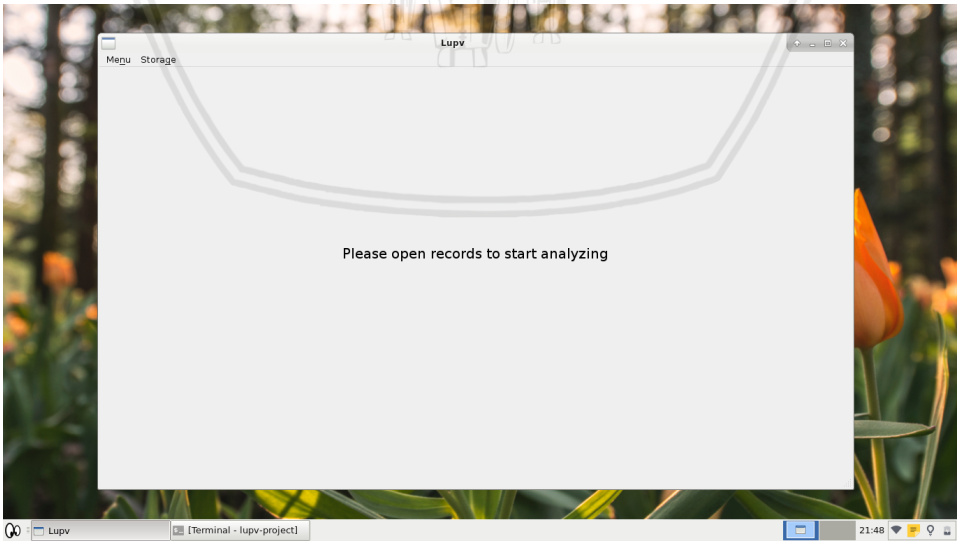
No	Nama Penguji	*Desktop* Environment	Versi
1	Bintang Dimas PAR	*GNOME*	3.28.2
2	Muhammad Iqbal K	*GNOME*	3.28.2
3	Laode Muhamad Fauzan	*KDE*	5.9.15
4	Husein Abdulbar	*KDE*	5.15.5
5	Rofy Firmansyah R	*KDE*	5.15.5
6	Satria Adhi Kharisma	*XFCE*	4.12
7	Dese Narfa Firmansyah	*XFCE*	4.12
8	Rizaldy Firmansyah Y	*Unity*	7
9	Insan Nurzaman	*Cinnamon*	3.6.6
10	Jerna Ferda Kusuma	*Cinnamon*	4.0.10
11	Rahmat Guntur Husodo	*Cinnamon*	4.0.10

No	Nama Penguji	*Desktop* Environment	Versi
12	Febristya AF	*Pantheon*	5.0
13	Achmad Rizki Aditama	*OpenBox*	3.6.1
14	Mohammad Anton RS	*i3wm*	4.16.1

Gambar 6.34 Sistem *Lup Recorder* berhasil merekam berkas tugas pada *desktop environment* yang berbeda-beda



Gambar 6.35 *Lup Viewer* pada *desktop environment i3wm*



Gambar 6.36 *Lup Viewer* pada *desktop environment XFCE*

BAB 7 PENUTUP

7.1 Kesimpulan

Kesimpulan dari hasil penelitian yang dilakukan adalah sebagai berikut:

1. Pada tahapan rekayasa kebutuhan didapatkan 2 aktor utama, 2 kebutuhan fungsional untuk sistem *Lup Recoder* dan 10 kebutuhan fungsional untuk sistem *Lup Viewer*. Dua aktor utama yang terlibat di dalam sistem adalah dosen dan mahasiswa. Kebutuhan utama sistem adalah merekam berkas tugas, merekam aktivitas siswa, dan memiliki sifat *platform agnostic*. Kedua sistem memiliki satu kebutuhan non-fungsional yaitu *compatibility*. Kedua sistem harus *compatible* dengan enam *desktop environment* utama pada sistem operasi *GNU/Linux*.

2. Pada tahapan perancangan didapatkan pemodelan *class diagram* yang terdiri dari 6 *class* pada sistem *Lup Recorder* dan 12 *class* pada sistem *Lup Viewer*, serta tiga sampel pemodelan *sequence diagram*. Hasil dari tahapan perancangan diimplementasikan dalam bentuk *working code* dengan bahasa pemrograman *Python*, dan antarmuka sistem diimplementasikan dengan bantuan *widget toolkit Qt*.

3. Terdapat lima pengujian yang dilakukan, yaitu pengujian unit, pengujian integrasi, pengujian validasi, *automated testing*, dan pengujian *compatibility*. Pengujian unit dilakukan dengan metode *white-box testing* dan teknik *basis path testing*, pengujian integrasi dilakukan dengan pendekatan *use-based testing* menggunakan metode *white-box testing* dan teknik *basis path testing*, pengujian validasi dilakukan dengan metode *black-box testing* dengan pemilihan nilai *input* menggunakan teknik *boundary value analysis* dan *equivalence partitioning*, *automated testing* dilakukan dengan menggunakan bantuan kakas bantu *Pytest* dan *Gitlab-CI*, dan pengujian *compatibility* dilakukan dengan menjalankan sistem pada enam *desktop environment* pada sistem operasi *GNU/Linux*. Pengujian unit dan integrasi yang dilakukan telah mencapai lebih dari 70% *code coverage* sehingga bisa dipastikan semua *path* utama selesai diuji. Pengujian validasi dilakukan terhadap seluruh *use case scenario* sehingga bisa dipastikan bahwa sistem yang dibangun memenuhi seluruh kebutuhan yang didefinisikan. *Automated testing* dijalankan tanpa menghasilkan galat sehingga pengujian dapat dilakukan dengan otomatis dan me-

mangkas waktu pengujian. Pengujian *compatibility* telah memastikan sistem berjalan dengan baik pada enam *desktop environment* berbeda.

7.2 Saran

Saran untuk penelitian selanjutnya adalah sebagai berikut:

1. Aplikasi pencegah plagiarisme dengan menggunakan metode *snapshotting* dan *user activity logging* menyimpan semua data rekaman secara lokal. Maka terdapat kemungkinan adanya manipulasi data rekaman oleh mahasiswa yang memahami struktur data rekaman sistem. Saran untuk penelitian selanjutnya adalah dengan tidak menyimpan data secara lokal, melainkan dikirim dan disimpan langsung pada penyimpanan *server*.
2. Sistem operasi yang didukung dapat ditambah dengan melakukan *porting* modul perekam pada sistem ke sistem operasi lain seperti sistem operasi *Microsoft Windows* dan *Apple Macintosh*.

DAFTAR PUSTAKA

About draw.io 2019. Available at: <https://about.draw.io/about-us/> [Accessed July 3, 2019].

Agustina, R. & Raharjo, P., 2017. Exploring Plagiarism into Perspectives of Indonesian Academics and Students. *Journal of Education and Learning*, 11(3), pp. 262–272.

Bell, D., 2000. *Software Engineering for Students*. Pearson Education.

Born, A. D., 2003. Teaching tip: How to reduce plagiarism. *Journal of Information Systems Education*, 14(3), p. 223.

Brooks Jr, F. P., 1995. *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition, 2/E*. Pearson Education India.

Burnstein, I., 2006. *Practical software testing: a process-oriented approach*. Springer Science & Business Media.

Capretz, L. F., 2003. A Brief History of the Object-Oriented Approach. *Software Engineering Notes*, 28(2).

Chacon, S. & Straub, B., 2014. *Pro git*. Apress.

DeLaRosa, A., 2018. *Log Monitoring: not the ugly sister*. Available at: <https://blog.pandorafms.org/log-monitoring/> [Accessed Aug. 21, 2018].

Dijkstra, E. W., 1970. *Notes on structured programming*.

Dweck, C. S. & Leggett, E. L., 1988. A social-cognitive approach to motivation and personality. *Psychological review*, 95(2), p. 256.

Fowler, M., 2018. *UnitTest*. Available at: <https://martinfowler.com/bliki/UnitTest.html> [Accessed Sept. 23, 2018].

Garimella, N., 2018. *Understanding and exploiting snapshot technology for data protection, Part 1: Snapshot technology overview*. Available at: <https://www.ibm.com/developerworks/tivoli/library/t-snaptsm1/> [Accessed Aug. 21, 2018].

Garner, H., Pulverer, B., Marusić, A., Petrovechi, M., Loadsman, J., Zhang, Y., McIntosh, I., Titus, S., Roig, M., & Anderson, M., 2012. How to stop plagiarism. *Nature*, 481(7382), pp. 21–23.

Gregory, L., 2007. *Path Testing*.

Hellas, A., Leinonen, J., & Ihantola, P., 2017. Plagiarism in Take-home Exams: Help-seeking, Collaboration, and Systematic Cheating. In: *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*. ACM, pp. 238–243.

Howard, R. M., 2002. Don't police plagiarism: just teach! *The education digest*, 67(5), p. 46.

IEEE, 1990. *IEEE Standard Glossary of Software Engineering Terminology*.

Kiss, A. K., 2013. Loopholes of plagiarism detection software. *Procedia-Social and Behavioral Sciences*, 106, pp. 1796–1803.

Leung, C. H. & Cheng, S. C. L., 2017. An Instructional Approach to Practical Solutions for Plagiarism. *Universal Journal of Educational Research*, 5(9), pp. 1646–1652.

Macbeth, S. W., Fernandez, R. L., Meyers, B. R., Tan, D. S., Robertson, G. G., Oliver, N. M., Murillo, O. E., Pedersen, E. R., Czerwinski, M. P., Spence, J. E., et al., Dec. 2007. *Logging user actions within activity context*. US Patent App. 11/426,846.

Martins, V. T., Fonte, D., Henriques, P. R., & Cruz, D. da, 2014. Plagiarism Detection: A Tool Survey and Comparison. In: *3rd Symposium on Languages, Applications and Technologies*. Vol. 38. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp. 143–158.

McLafferty, C. L. & Foust, K. M., 2004. Electronic plagiarism as a college instructor's nightmare—Prevention and detection. *Journal of Education for Business*, 79(3), pp. 186–190.

Meuschke, N. & Gipp, B., 2013. State-of-the-art in detecting academic plagiarism. *International Journal for Educational Integrity*, 9(1).

Modiba, P., Pieterse, V., & Haskins, B., 2016. Evaluating plagiarism detection software for introductory programming assignments. In: *Proceedings of the Computer Science Education Research Conference 2016*. ACM, pp. 37–46.

Myers, G. J., Sandler, C., & Badgett, T., 2011. *The art of software testing*. John Wiley & Sons.

Navarro, G., 2001. A guided tour to approximate string matching. *ACM computing surveys (CSUR)*, 33(1), pp. 31–88.

Neill, C. J. & Shanmuganthan, G., 2004. A Web-enabled plagiarism detection tool. *IT professional*, 6(5), pp. 19–23.

Nshimiyiman, M., 2018. *Travis-ci - Send notification with custom message*. Available at: <http://www.marcellin.me/blog/travis-ci-notification-custom-message/> [Accessed Sept. 23, 2018].

Osherove, R., 2015. *The art of unit testing*. MITP-Verlags GmbH & Co. KG.

Park, C., 2004. Rebels without a clause: Towards an institutional framework for dealing with plagiarism by students. *Journal of further and Higher Education*, 28(3), pp. 291–306.

Presman, R. S., 2010. *Software Enginering a Practioner's Aproach*. New York: McGraw-Hill Companies Inc.

Pressman, R. S., 2010. *Software Engineering–A Practitioner's Approach*. Mc Graw-Hill.

Qt About 2018. Available at: <https://wiki.qt.io/About_Qt> [Accessed Dec. 21, 2018].

Qt History 2018. Available at: <https://wiki.qt.io/Qt_History> [Accessed Dec. 21, 2018].

Rumbaugh, J., Jacobson, I., & Booch, G., 2004. *Unified modeling language reference manual, the*. Pearson Higher Education.

Ruparelia, N. B., 2010. The history of version control. *ACM SIGSOFT Software Engineering Notes*, 35(1), pp. 5–9.

Sanner, M. F., et al., 1999. Python: a programming language for software integration and development. *J Mol Graph Model*, 17(1), pp. 57–61.

Savenkov, R., 2008. *How to become a software tester*. Roman Savenkov.

Sommerville, I., 2014. *Software Engineering*. Pearson Education.

Torvalds, L., 2018. *git mailing list*. Available at: <https://marc.info/?l=linux-kernel&m=111314792424707> [Accessed Aug. 21, 2018].

Whittaker, J. A., Arbon, J., & Carollo, J., 2012. *How Google tests software*. Addison-Wesley.