Sistem Pencegah Plagiarisme Dalam Tugas Mahasiswa Dengan Snapshotting Dan User Activity Logging

Azzam Syawqi Aziz[1] , Fajar Pradana[2] , Fitra A. Bachtiar [3]

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

Abstrak

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. Penanganan dengan metode *plagiarism detection* memiliki banyak kekurangan, seperti timbulnya paradigma polisi-kriminal, pola pikir nilai *oriented*, dan kelemahan teknis seperti lemah dalam menganalisis dokumen yang diterjemahkan. Penelitian Hellas, Leinonen, dan Ihantola dengan metode *plagiarism prevention* memiliki beberapa kekurangan, yaitu mengabaikan proses pengerjaan sehingga dapat dicurangi siswa dengan membeli tugas, analisis di akhir membuat siswa dapat melakukan kecurangan selama pengerjaan, tidak memiliki *log* sehingga guru tidak dapat menilai usaha siswa, dan *software* yang digunakan terikat dengan *platform* tertentu. Penelitian ini bertujuan untuk menyempurnakan penelitian sebelumnya dengan menambahkan proses *snapshotting* yang merekam perubahan dokumen, *user activity logging* yang merekam aktivitas siswa, dan membangun sistem dengan sifat *agnostic* sehingga terdapat kebebasan memilih bentuk tugas dan *software* yang akan digunakan. Pengujian sistem dilakukan dengan pengujian unit, integrasi, validasi, *automated testing,* dan *compatibility*. Pengujian unit dan integrasi bernilai valid dan melebihi 70% *code coverage*, pengujian validasi pada 34 kasus uji bernilai valid, dan pengujian *compatibility* menunjukkan sistem berjalan baik 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

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. Handling with the plagiarism detection method has many disadvantages, such as the emergence police-criminal paradigm, value-oriented mindset, and technical weakness such as inability in analyzing translated documents. Hellas, Leinonen, and Ihantola's research with the plagiarism prevention method has several shortcomings, namely ignoring the process of work so that students can be cheated by buying assignments, the analysis at the end makes students able to cheat during assignments, no any log produced so the teacher can't assess student effort, and the software is tied to a particular platform. This research aims to perfect previous research by adding a snapshotting process that records document changes, user activity logging which records student activities, and build the systems with agnostic properties so that there is freedom to choose the form of tasks and the software that will be used. The system is tested using unit testing, integration, validation, automated testing, and compatibility. Unit testing and integration are valid and exceed 70% code coverage, validation testing in 32 test cases are valid, and compatibility testing shows the system is running well on six common desktop environments. Valid automated testing that contains no errors can speed up the testing process.

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

1. PENDAHULUAN

Plagiarisme dalam lingkungan akademisi adalah penggunaan kata atau ide dari sebuah sumber tanpa menyertakan pengakuan sebagaimana ditentukan oleh prinsip-prinsip akademik (Meuschke & Gipp, 2013). Plagiarisme menjadi hal yang tak terpisahkan dari lingkungan akademisi. Ketersediaan sumber daya *internet* dan kemudahan mencari informasi berkontribusi pada munculnya plagiarisme oleh siswa (Born, 2003). Plagiarisme di Indonesia tidak hanya dilakukan oleh mahasiswa, melainkan 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, dilema sosial, membuat tingkat plagiarisme di universitas semakin tinggi (Park, 2004). Oleh karena alasan-alasan tersebut diperlukan penanganan plagiarisme

Terdapat dua metode dalam plagiarisme, yaitu *plagiarism detection* dan *plagiarism prevention*. Metode *plagiarism detection* memiliki beberapa kelemahan seperti menyebabkan timbulnya paradigma polisi-kriminal antar guru dan murid (Howard, 2002), pola pikir nilai *oriented (Dweck & Leggett, 1988)*, dan beberapa kelemahan teknis lain seperti lemah dalam menganalisis dokumen yang telah diterjemahkan (Meuschke & Gipp, 2013).

Plagiarism Detection Software (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). Hellas, Leinonen, & Ihantola (2017) telah melakukan penelitian dengan menggunakan metode *plagiarism prevention*. Metode tersebut merekam waktu mulai, waktu selesai, dan alamat *IP*. Pada akhir proses berkas tugas dianalisis menggunakan algoritme *edit-distance*. Tetapi metode pada penelitian Hellas, Leinonen, & Ihantola (2017) memiliki beberapa kelemahan, yaitu mengabaikan proses pengerjaan tugas sehingga dapat dicurangi siswa dengan membeli tugas (Leung & Cheng, 2017), analisis hanya dilakukan di akhir sehingga siswa dapat berkolaborasi melakukan kecurangan selama tugas dikerjakan, tidak memiliki *log* sehingga guru tidak dapat menilai usaha siswa (Hellas, Leinonen, & Ihantola, 2017), dan perangkat lunak yang digunakan terikat dengan *platform* tertentu.

Kelemahan pada metode yang dilakukan Hellas, Leinonen, & Ihantola (2017) akan disempurnakan dengan mengimplementasikan proses *snapshotting* yang merekam seluruh perubahan dokumen 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 yang disukai untuk mengerjakan tugas.

Penelitian ini akan mengimplementasikan metode *snapshotting*, *user activity logging,* dan algoritme *edit-distance* dari penelitian sebelumnya. Metode *snapshotting* akan dibangun di atas *distributed version control* untuk merekam seluruh perubahan dokumen tugas. Metode *activity logging* digunakan untuk merekam segala aktivitas siswa selama pengerjaan tugas. Maka seluruh usaha, keputusan, keterlibatan dan aktivitas siswa dapat dianalisis oleh guru.

2. LANDASAN KEPUSTAKAAN

2.1 Snapshotting

Snapshot merupakan proses untuk merekam suatu keadaan media penyimpanan dalam suatu waktu tertentu. *Snapshot* yang dihasilkan dapat dikembalikan kapan pun pengguna butuhkan. *Version Control Software* (*VCS*) merupakan salah satu kakas bantu yang dapat digunakan untuk membuat *snapshot*. *Git* merupakan salah satu *version control software* yang digunakan secara umum. Hal ini disebabkan karena *git* yang memiliki sifat *distributed* sehingga memiliki kelebihan yang tidak dimiliki *version control* terpusat, salah satunya adalah kemampuan untuk melakukan seluruh operasi secara lokal (Ruparelia, 2010). Di dalam *git, snapshot* merupakan sebuah keadaan berkas dalam waktu tertentu. Pembuatan *snapshot* dapat dilakukan dengan menggunakan perintah "*add*" dan "*commit*". Berbeda dengan *version control* lain *git* tidak menyimpan *diff* antar berkas, melainkan menyimpan seluruh keadaan berkas pada waktu tertentu. Semua *snapshot* tersebut memiliki integritas yang berupa 40 karakter *hexadecimal* dihasilkan dari kalkulasi berkas pada suatu

snapshot dengan algoritme *Secure Hash Algorithm 1* atau biasa dikenal dengan *SHA-1* (Chacon & Straub, 2014). Pada penelitian ini *git* digunakan untuk menyimpan *snapshot* dari perubahan berkas tugas dari waktu ke waktu.

2.2 User Activity Logging

Log merupakan sebuah berkas yang memuat rekaman aktivitas yang terjadi pada perangkat lunak. *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).

2.3 Algoritme *Edit-distance*

Algoritme *edit-distance* merupakan sebuah algoritme untuk menghitung penghapusan, sisipan, maupun substitusi antara dua buah kata. Semua karakter 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 algoritme *edit-distance* dapat ditulis dengan persamaan (1):

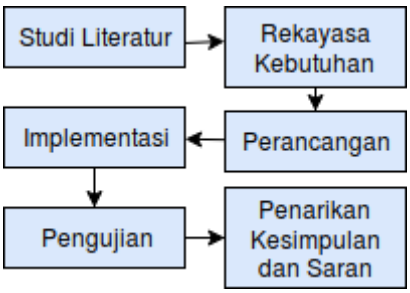
$$^{w}ins^{(x)}, ^{w}del^{(x)}, ^{w}\mathrm{sub}^{(x,y)} \quad (1)$$

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

1) tekkom → fekkom (penggantian dari "t" ke "f)
2) fekkom → fikkom (penggantian dari "e" ke "i")
3) fikkom → filkom (penggantian dari "k" ke "l")

3. METODOLOGI PENELITIAN

Terdapat beberapa langkah-langkah dalam penelitian ini. 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* pada pengembangan sistem dalam penelitian ini disebabkan kebutuhan sistem yang sudah dapat didefinisikan dengan baik pada tahap awal pengembangan sistem. Langkah terakhir penelitian ditutup dengan penarikan kesimpulan dan saran. Langkah-langkah penelitian dapat dilihat pada Gambar 1.



Gambar 1. Diagram Alir Metodologi Penelitian

Studi literatur dibutuhkan untuk mendalami teori plagiarisme dan pengembangan perangkat lunak. Sumber dari literatur tersebut berasal dari jurnal, buku, situs resmi dan dokumentasi perangkat lunak yang digunakan.

Rekayasa kebutuhan merupakan kumpulan tugas dan teknik untuk memahami kebutuhan (Pressman, 2010). Terdapat empat tahapan dalam rekayasa kebutuhan, yaitu tahap analisis kebutuhan, spesifikasi kebutuhan, manajemen kebutuhan, dan pemodelan kebutuhan. Analisis kebutuhan dilakukan untuk menggali seluruh kebutuhan yang diperlukan untuk membangun sistem. 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. Sedangkan kebutuhan non-fungsional sistem didapatkan dari batasan sistem yang hanya berjalan pada sistem operasi *GNU/Linux* sehingga sistem

harus *compatible* dengan berbagai *desktop environment* yang marak digunakan. Setelah kebutuhan didapatkan dari proses analisis kebutuhan, dilakukan proses spesifikasi kebutuhan yang meningkatkan pernyataan kebutuhan menjadi lebih detail, jelas dan tidak ambigu. Langkah selanjutnya adalah manajemen kebutuhan yang di dalamnya dilakukan kodefikasi terhadap kebutuhan sehingga kebutuhan dapat dengan mudah diidentifikasi, dikontrol dan dilacak. Langkah terakhir adalah pemodelan kebutuhan yang dilakukan dengan bantuan *use case diagram* dan *use case scenario*. *Use-case diagram* digunakan untuk menggambarkan fungsionalitas dan batasan sistem secara visual. Sedangkan *use case scenario* digunakan untuk menjelaskan skenario interaksi antar aktor dan sistem secara mendetail.

Perancangan dilakukan dengan merancang arsitektur, komponen dan antarmuka sistem. Rancangan ini berdasar pada hasil analisis kebutuhan sebelumnya. Pada tahapan penancangan arsitektur, sistem dibangun menggunakan arsitektur *Model-View-Controller (MVC)*. Arsitektur *MVC* dipilih agar komponen fungsionalitas sistem terpisah dari komponen antarmuka sistem atau *separation of concern*. Untuk menggambarkan interaksi yang terjadi antar komponen pada sistem, penelitian ini menggunakan alat bantu *sequence digaram*, dan untuk menggambarkan struktur sistem dan relasi antar komponen pada sistem, digunakan alat bantu *class diagram*. Perancangan komponen dilakukan menggunakan *pseudocode* dan perancangan antarmuka dilakukan dengan menggambar dasar antarmuka dan tata letak komponen pada antarmuka sistem.

Tahapan selanjutnya yaitu implementasi. Tahapan ini mengimplementasikan seluruh hasil rancangan sebelumnya. Terdapat dua tahapan yaitu implementasi kode program dan implementasi antarmuka. Pada implementasi kode program, hasil rancangan yang berupa *pseudocode* diimplementasikan menjadi *working code* dengan bahasa pemrograman *Python.* Pada implementasi antarmuka hasil rancangan antarmuka diimplementasikan dengan bantuan *widget toolkit Qt*.

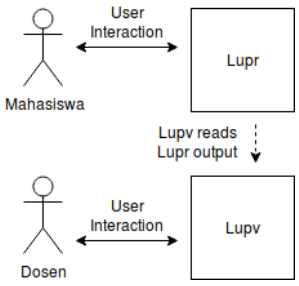
Pengujian sistem dilakukan dengan pengujian unit, integrasi, validasi, dan *compatibility*. pengujian unit dilakukan untuk memastikan hasil implementasi kode program sesuai dengan hasil perancangan komponen. Pengujian unit dilakukan pada *class Controller* dan *Model* hingga mencapai 100% *coverage*, karena semua inti program terdapat pada kedua *class* tersebut. Pengujian unit menggunakan metode *white-box testing* dan teknik *basis path testing*. Pengujian setiap unit dilakukan secara terisolasi sehingga dilakukan pembuatan *stub*. Pengujian integrasi dilakukan untuk memastikan integrasi antar unit berjalan dengan baik. Maka pada pengujian integrasi tidak ada pembuatan *stub* atau *fake.* Pengujian integrasi dilakukan dengan pendekatan *use-based testing.* Metode yang digunakan adalah *white-box testing* dan teknik yang digunakan adalah *basis path testing.* Pengujian validasi menguji apakah sistem yang dibangun sudah sesuai dengan kebutuhan yang didefinisikan. Pengujian validasi dilakukan dengan metode metode *black-box testing* dengan teknik *equivalence partitioning* dan *boundary value analysis* untuk memilih nilai masukan pada pengujian. *Automated testing* dilakukan untuk mempercepat waktu pengujian. Pada tahapan *automated testing* dibangun *script* yang menjalankan kasus uji secara otomatis. Kasus uji didapatkan dari pengujian unit dan pengujian integrasi yang telah dilakukan pada tahapan sebelumnya. Pada tahapan ini, kakas bantu *Pytest* digunakan untuk membangun *script*, dan kakas bantu *Gitlab-CI digunakan* untuk menjalankan *script* secara otomatis. Pengujian *compatibility* dilakukan dengan menjalankan sistem pada enam *desktop environment* umum. Keenam *desktop environment* tersebut adalah *GNOME, KDE, XFCE, Unity, Cinnamon* dan *Phanteon.*

4. REKAYASA KEBUTUHAN

Sistem dibagi menjadi dua bagian. Bagian pertama yaitu *Lup Recorder* yang dijalankan pada komputer mahasiswa untuk merekam pengerjaan tugas, dan bagian kedua yaitu *Lup Viewer* yang dijalankan pada komputer dosen untuk melihat hasil rekaman. Gambaran umum sistem terdapat dalam Gambar 2.

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). Analisis kebutuhan pada sistem ini didapatkan dari beberapa sumber. Sumber pertama adalah metode penelitian yang telah dilakukan oleh Hellas, Leinonen, & Ihantola (2017). Sumber

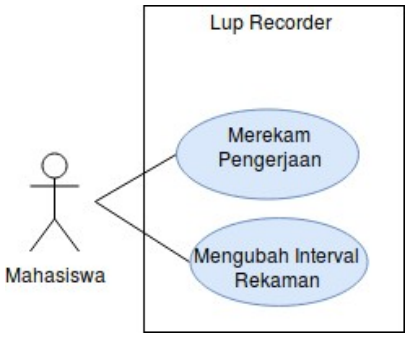
kedua didapatkan dari "future work" penelitian Hellas, Leinonen, & Ihantola (2017) dan penelitian Leung & Cheng (2017). Sumber kebutuhan ketiga adalah kelemahan dari penelitian Hellas, Leinonen, & Ihantola (2017) yang akan disempurnakan. Ketiga sumber tersebut menghasilkan kebutuhan-kebutuhan pokok sistem 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. Maka dari ketiga sumber tersebut didapatkan seluruh hasil kebutuhan yang berupa 2 aktor yang terlibat, yaitu mahasiswa dan dosen, 2 kebutuhan fungsional untuk sistem *Lup Recorder,* 10 kebutuhan fungsional untuk sistem *Lup Viewer*, dan 1 kebutuhan non-fungsional untuk kedua sistem.



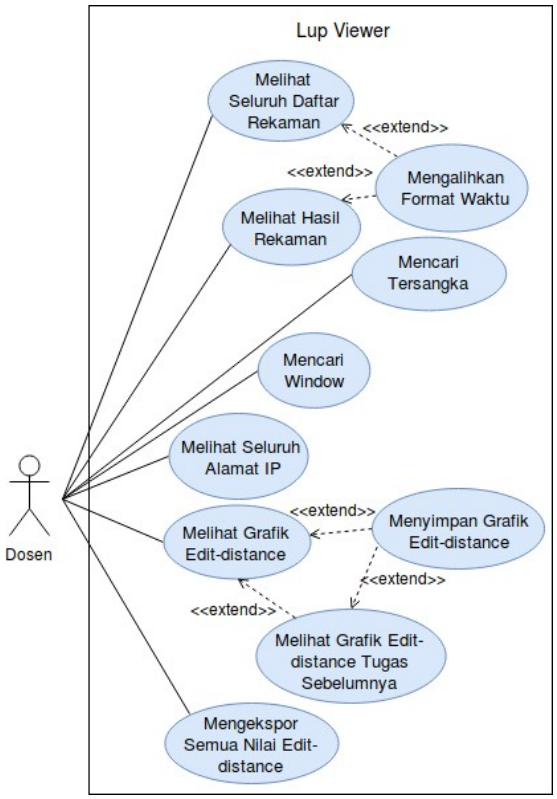
Gambar 2. Gambaran umum sistem

Kebutuhan non-fungsional merupakan batasan (*constraint*) dari sebuah sistem (Sommerville, 2014). Kebutuhan non-fungsional sistem adalah *compatibility* dengan enam *desktop environment* yaitu *GNOME, KDE, XFCE, Unity, Cinnamon dan Phanteon*. Kebutuhan non-fungsional ini didapatkan dari batasan sistem yang hanya berjalan pada sistem operasi *GNU/Linux* sehingga sistem harus dapat berjalan pada *desktop environment* yang marak digunakan.

Setelah kebutuhan sistem didapatkan dari proses analisis kebutuhan, pernyataan kebutuhan ditingkatkan dalam proses spesifikasi kebutuhan. Spesifikasi kebutuhan merupakan proses untuk meningkatkan pernyataan kebutuhan menjadi lebih detail, jelas, tidak ambigu, mudah dipahami, dan konsisten. Spesifikasi tersebut kemudian dikodefikasi pada proses manajemen kebutuhan untuk memudahkan identifikasi, kontrol dan pelacakan kebutuhan. Kebutuhan sistem kemudian dimodelkan dengan bantuan *use case diagram* dan *use case scenario*. *Use case diagram* menggambarkan fungsionalitas dan batasan sistem secara visual, sedangkan *use case scenario* menjelaskan skenario interaksi antar aktor dan sistem secara mendetail. *Use case diagram* untuk sistem *Lup Recorder* terdapat dalam Gambar 3, dan *use case diagram* untuk sistem *Lup Viewer* terdapat dalam Gambar 4.



Gambar 3. *Use case diagram* sistem *Lup Recoder*



Gambar 4. *Use case diagram* sistem *Lup Viewer*

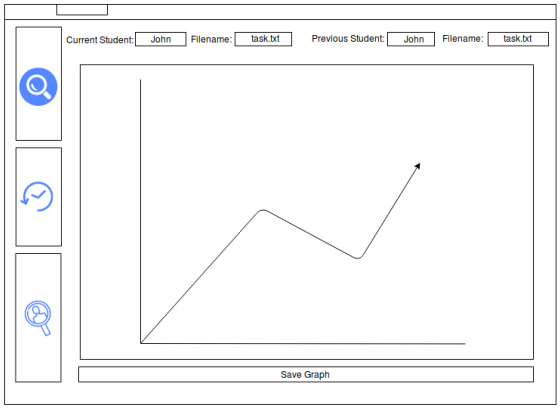
5. PERANCANGAN

Pada tahap perancangan hasil dari tahap analisis kebutuhan dirancang dalam tiga

tahapan, tahap-tahap tersebut adalah perancangan arsitektur, perancangan komponen dan perancangan antarmuka.

Pada 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 sistem dirancang dengan bantuan *sequence diagram* dan *class diagram*. *Sequence diagram* digunakan untuk menggambarkan interaksi yang terjadi antar komponen pada sistem. *Class diagram* digunakan untuk menggambarkan struktur sistem dan relasi antar komponen pada sistem (Pressman, 2010).

Perancangan komponen menghasilkan *pseudocode* yang nantinya menjadi landasan dan acuan pada tahapan implementasi komponen. Perancangan antarmuka menghasilkan gambaran dasar antarmuka sistem. Salah satu hasil perancangan antarmuka, yaitu hasil perancangan antarmuka "Tampilan *Edit-distance* Tugas Sebelumnya" dapat dilihat pada Gambar 5.



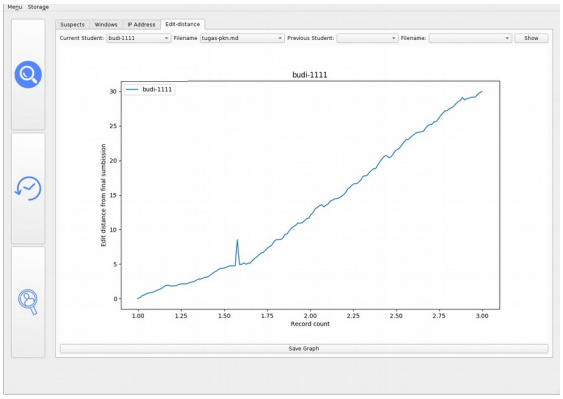
Gambar 5. Hasil perancangan antarmuka "Tampilan *Edit-distance* Tugas Sebelumnya"

6. IMPLEMENTASI

Pada tahapan implementasi terdapat dua tahapan, yaitu tahap implementasi kode program, dan tahap implementasi antarmuka. Pada tahap implementasi kode program, hasil rancangan pada tahapan perancangan sistem diimplementasikan. Hasil rancangan komponen yang berbentuk *pseudocode* akan diimplementasikan menjadi *working code* dengan bahasa pemrograman *Python*. Pada tahapan implementasi antamuka, hasil rancangan antarmuka akan diimplementasikan dengan bantuan *widget toolkit Qt*. Terlihat hasil implementasi antarmuka "Tampilan *Edit-distance* Tugas Sebelumnya" pada Gambar 6.

7. PENGUJIAN

Terdapat beberapa pengujian yang dilakukan pada sistem, yaitu pengujian unit, integrasi, validasi, *automated testing,* dan *compatibility testing*. Pengujian unit dilakukan untuk memastikan hasil implementasi kode program sesuai dengan hasil perancangan komponen. Pengujian unit dilakukan pada *class Controller* dan *Model* hingga mencapai 100% *code coverage*, karena semua inti program terdapat pada kedua *class* tersebut. Pengujian unit dilakukan pada komponen terkecil sistem, yaitu *class*. *Testable* unit dari sebuah *class* adalah *method* atau *function* (Pressman, 2010). Oleh karena itu pengujian unit dilakukan pada *function-function* yang ada menggunakan metode *white-box testing* dan teknik *basis path testing*. Pengujian unit dilakukan secara terisolasi sehingga dilakukan pembuatan *stub*.



Gambar 6. Hasil implementasi antarmuka "Tampilan *Edit-distance* Tugas Sebelumnya"

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. Pengujian integrasi menggunakan pendekatan *use-based testing*, yaitu menguji kelas yang paling tidak memiliki *dependent-class*. Maka pengujian integrasi dimulai pada *class Model*, kemudian *Controller* dan akhirnya *class View*. Pengujian integrasi menggunakan metode *white-box testing* dan teknik *basis path testing*.

Pengujian validasi dilakukan untuk menguji apakah sistem yang dibangun sudah sesuai dengan kebutuhan yang didefinisikan. Pengujian validasi fokus terhadap *user-visible acton* dan *user-recognizable output* dari sistem (Pressman, 2010). Maka pada tahapan ini seluruh skenario kebutuhan diuji. Pengujian validasi dilakukan dengan metode *black-box testing*. Pemilihan nilai masukan (*input*) untuk pengujian dilakukan dengan teknik *equivalence partitioning* dan *boundary value analysis* untuk menghindari *rigorous testing* atau mencoba semua kemungkinan nilai masukan. Salah satu kasus uji dan hasil uji pengujian validasi "Melihat Grafik *Edit-distance* Tugas Sebelumnya" dipaparkan pada Tabel 1.

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. Kakas bantu *Pytest* digunakan untuk membangun *script,* dan kakas bantu *Gitlab-CI* digunakan untuk menjalankan *script* secara otomatis.

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. 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.*

Tabel 1. Kasus uji dan hasil uji pengujian validasi "Melihat Grafik *Edit-distance* Tugas Sebelumnya" kondisi berhasil.

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"
Expected Result	Sistem menampilkan grafik edit-distance tugas sebelumnya
Result	*As expected*
Status	Valid

8. KESIMPULAN

Rekayasa kebutuhan menghasilkan dua aktor utama, yaitu dosen dan mahasiswa. Didapatkan juga 2 kebutuhan untuk sistem *Lup Recorder* dan 10 kebutuhan untuk sistem *Lup Viewer*. Satu kebutuhan non-fungsional untuk kedua sistem yaitu *compatibility*.

Perancangan sistem menghasilkan pemodelan *class diagram* dan se*quence diagram*. Terdapat 6 *class* pada sistem *Lup Recoder* dan 12 *class* pada sistem *Lup Viewer*. Hasil rancangan komponen diimplementasikan pada tahapan implementasi kedalam bentuk *working-code* dengan bahasa pemrograman *Python* dan hasil rancangan antarmuka diimplementasikan dengan *widget toolkit Qt*.

Pengujian yang dilakukan adalah pengujian unit, integrasi, 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 teknik *boundary value analysis* dan *equivalence partitioning* untuk pemilihan *input. Automated testing* dilakukan dengan menggunakan bantuan kakas bantu *Pytest* dan *Gitlab-CI.* 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% c*ode coverage* sehingga bisa dipastikan semua *path* utama selesai diuji. Pengujian validasi dilakukan terhadap seluruh *case scenario* sehingga dapat dipastikan bahwa sistem yang dibangun memenuhi seluruh kebutuhan yang diperlukan. *Automated testing* dijalankan tanpa menghasilkan galat sehingga pengujian dapat dilakukan dengan otomatis dan memangkas waktu pengujian. Pengujian *compatibility* telah

memastikan sistem berjalan dengan baik pada enam *desktop environment* berbeda.

Saran untuk penelitian selanjutnya yaitu tidak menyimpan data rekaman secara lokal, melainkan mengirimkan dan menyimpannya langsung pada penyimpanan *server*. Hal ini dapat menghindari adanya kemungkinan manipulasi data oleh mahasiswa yang memahami struktur rekaman sistem. Saran kedua yaitu menambahkan dukungan sistem operasi dengan melakukan *porting* modul perekam ke sistem operasi lain seperti sistem operasi *Microsoft Window*s dan *Apple Macintosh.*

9. DAFTAR PUSTAKA

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

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

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

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

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.

Hellas, A., Leinonen, J., & Ihantola, P., 2017. Plagiarism in Take-home Exams: Helpseeking, 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.

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 ac vity context*. US Patent App. 11/426,846.

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

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.

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

Pressman, R. S., 2010. *Software Engineering A Practitioner's Approach*. Mc GrawHill.

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.

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

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