**SYSTEMIC**
Vol. 02, No. 01, Agustus 2016, 1-17

# MEMBANGUN HUBUNGAN KERUNUTAN ARTIFAK PADA LINGKUNGAN PENGEMBANGAN CEPAT

**Hengki Suhartoyo[1)], Siti Rochimah[2)]**

[1,2)] Teknik Informatika, Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
e-mail: hengki.suhartoyo@gmail.com [1)], siti@its-sby.edu [2)]

***Abstrak***

*Kerunutan merupakan mekanisme penting untuk mengelola dan mengaudit proses pengembangan perangkat lunak, dapat dikatakan bahwa semua artefak dari pengembangan perangkat lunak diarahkan dan berkaitan dengan kebutuhan perangkat lunak. Kerunutan digunakan untuk berbagai tujuan, termasuk untuk manajemen kebutuhan, manajemen perubahan, dampak analisis, verifikasi, validasi, dan audit. Untuk membantu proses membangun kerunutan hubungan antar artefak perlu dikembangkan alat bantu yang dapat secara otomatis menghubungkan antar artefak didalam lingkungan pengembangan cepat. Pada penelitian ini dikembangkan kakas bantu untuk membentuk hubungan kerunutan antara kedua artefak tersebut yang berbahasa Indonesia. Pada penelitian ini diusulkan metode yang dapat membentuk hubungan kerunutan secara otomatis menggunakan metode pencocokan string antara cerita pengguna dengan kode sumber. Cerita pengguna dan kode sumber diekstraksi menjadi himpunan kata dasar dan dilakukan pencocokan kata antara keduanya menggunakan algoritma trigram. Ekstraksi cerita pengguna menggunakan algoritma Nazief & Adriani. Ekstraksi kode sumber menggunakan pustaka javaparser dan dioptimasi menggunakan pendekatan konvensi penamaan java. Hasil dari pencocokan dioptimalkan dengan memberikan ambang batas jumlah kata yang terkandung dalam kode sumber dibandingkan dengan jumlah kata yang terdapat pada cerita pengguna. Sistem diuji dengan tiga dataset cerita pengguna beserta kode sumbernya yaitu smartPortal terdiri dari 17 cerita pengguna dan 52 file kode sumber, smartAbsensi terdiri dari 47 cerita pengguna dan 161 file kode sumber yang ketiga smartTravel 80 cerita pengguna dan 222 file kode sumber. Hasil otomatisasi hubungan kerunutan oleh sistem dibandingkan dengan hasil kerunutan hubungan oleh pengembang diperoleh nilai presisi smartPortal 0.288, smartAbsensi 0.285 dan smartTravel 0.213. Rata-rata hasil presisi dibawah 0.5 disebabkan dataset masih banyak menggunakan identifier berbahasa inggris dan singkatan-singkatan.*

**Kata Kunci:** *Kerunutan, lingkungan pengembangan cepat, cerita pengguna,pencocokan string*

***Abstract***

*Traceability is an important mechanism to manage and audit the software development process, all of the artifacts of software development aimed and related to the software requierement. Traceability is used for various purposes, including to requirement management, change management, impact analysis, verification, validation, and audit. To help the process of establishing traceability links between artifacts is necessary to develop automatic traceability in the rapid application development environment. Some of the major artifacts in a rapid development environment (agile) are user story and source code. This research proposed a method to establish traceability link between user story and java source code in Indonesian Language. User story and source code are extracted into a set of basic word, then matched both of them using trigram algorithm. User story extraction using Nazief & Adriani algorithm. Extraction of the source code using the library javaparser and optimized by java naming convention approaches. The results of matching words are optimized by providing a threshold number of words contained in the source code as compared to the number of words contained in the story.The automatic traceabillty tool have been made tested using three dataset that containing user story and java source code. First dataset is smartPortal containing 17 user stories and 52 files source code. Second dataset is smartAbsensi containing 47 user stories and 52 files source codes. Third dataset is smartTravel that containing 80 user stories and 222 files souce codes. The results of automatization traceabillity link compared with traceability link by developer. The result of comparation ranked using precision recall with the output are: smartPortal 0.288, smartAbsensi 0.285 and smartTravel 0.213. The average results of the precision of less than 0.5 caused datasheet still using the English language identifier and abbreviations.*

**Keywords:** *traceabillity, agile, user story , string matching.*

# 1. PENDAHULUAN

Kerunutan (Traceability) merupakan mekanisme penting untuk mengelola dan mengaudit proses pengembangan perangkat lunak (P. Lago, 2009), dapat dikatakan bahwa semua artefak dari pengembangan perangkat lunak diarahkan dan berkaitan dengan kebutuhan perangkat lunak. Kerunutan digunakan untuk berbagai tujuan, termasuk untuk manajemen kebutuhan, manajemen perubahan, dampak analisis, verifikasi, validasi, dan audit. Namun dalam lingkungan pengembangan perangkat lunak cepat terdapat berbagai macam artefak yang terdapat dalam satu dokumentasi, hal ini mencerminkan fakta bahwa dalam model pengembangan perangkat lunak cepat setiap artefak dapat digunakan sebagai dokumentasi. (Matthias, 2013) artefak penting dalam melakukan pemodelan perangkat lunak diantaranya yaitu Diagram Kasus Pengguna, Diagram Kelas, Diagram urutan , Diagram Kolaborasi, Ujicoba Kasus, dan kode sumber. Artefak-artefak tersebut saling berhubungan satu dengan yang lainnya. Mengetahui kerunutan antar artefak tersebut sangat dibutuhkan dalam perancangan perangkat lunak. Dengan banyaknya artefak yang akan ditelusuri kerunutan hubungannya maka akan semakin memerlukan waktu dan biaya.

Artefak didalam pengembangan perangkat lunak tangkas (agile software development) selanjutnya disebut Agile berbeda dengan metodologi pengembangan perangkat lunak seperti dengan metodologi maupun Metodologi Berbasis Obyek (Object Oriented Methodologies) (Kenneth , 2011). Agile sebagai metodologi pengembangan system alternative memiliki beberapa jenis diantaranya yang paling popular yaitu: Scrum, eXtreme Programming(XP), Kanban, Feature Driven Development (FDD) dan Agile Unified Process (AUP). Dari berbagai jenis tersebut agile menimbulkan berbagai macam istilah dan artefak yang belum pernah diklasifikasikan sebelumnya, sehingga Matthias GrÖber melakukan penelitian pada 76 paper yang membahas tentang agile, dan menemukan bahwa didalam 73 paper terdapat 314 nama artefak yang berbeda-beda, sehingga artefak pada agile diklasifikasikan menjadi: artefak final, spesifikasi kebutuhan, kasus uji, proses produksi, manajemen proyek dan Backlog Item (Matthias, 2013). Pada agile tingkat perubahan artefak-artefak relative lebih sering terjadi karena antara pengguna dan pengembang terlibat menjadi satu kesatuan tim pengembangan system yang terlibat secara intensif, sehingga antara perubahan RS dan kode sumber dapat dipastikan berubah secara cepat.

Selama pembangunan, artefak perangkat lunak berubah terus-menerus itulah sebabnya metode sederhana untuk pemulihan hubungan menjadi tidak efisien. Suatu pendekatan yang memanfaatkan metode pemulihan kerunutan hubungan untuk menyediakan kerunutan hubungan terus berkembang. Otomatisasi dari kerunutan hubungan sebelumnya menggunakan algoritma Latent Semantic Indexing mengevaluasi kembali dari kerunutan hubungan yang terus mengalami perubahan. Metode yang digunakan menggunakan semua informasi yang tersedia sebelumnya dan hanya mengevaluasi dampak perubahan pada kerunutan tersebut dengan demikian biaya untuk pemulihan kerunutan akan dihindari untuk versi selanjutnya (Rilling, 2007).

Dalam banyak penelitian tentang kerunutan hubungan pengembangan perangkat lunak, sebagian besar menangani masalah pemulihan kerunutan hubungan untuk mengidentifikasi hubungan semantik antara kode sumber dan dokumentasinya pada metode pengembangan tradisional maupun OOM. Pada kesempatan ini penulis mencoba untuk menawarkan otomatisasi kerunutan hubungan didalam lingkungan pengembangan cepat antara cerita pengguna dengan kode sumber menggunakan pencocokan string. Penulis memilih hubungan kerunutan tersebut karena dalam lingkungan pengembangan cepat khususnya scrum perubahan iterasi atau sprint terjadi cukup cepat dari satu versi ke versi berikutnya, sehingga dibutuhkan alat yang dapat mengetahui dengan cepat hubungan kerunutan antara cerita pengguna dengan kode sumber. Dengan string matching, pencarian string yang mirip antara cerita pengguna dengan kode sumber diterapkan sebagai mekanisme penentuan kerunutan hubungan tanpa membatasi kebebasan dalam mengekspresikan kebutuhan dalam Cerita Pengguna. Hubungan diidentifikasi dengan mencari kecocokan antara kata-kata didalam cerita pengguna dengan kode sumber yang telah diekstraksi menjadi nama-nama class, method, atrribut, dan komentar.

# 2. DASAR TEORI

## 2.1 Hubungan Kerunutan

Kerunutan adalah properti dari desain dan pengembangan perangkat lunak yang menghubungkan setiap artefak abstrak dengan artefak teknis maupun sebaliknya (Mens, 2008). Pada proyek perangkat lunak skala besar, kerunutan sangat penting untuk mengetahui hubungan kerunutan antar artefak dalam fase-fase yang berbeda (analisis kebutuhan, desain, dan implementasi). Selain itu, dapat mengetahui hubungan kerunutan antara artefak dan pihak

pengembang yang terlibat.

Proses pembangunan ketelusuran yang dilakukan secara manual akan menghabiskan banyak waktu dan sangat berpotensi terjadi kesalahan sehingga informasi menjadi tidak lengkap. Oleh karena itu, sebuah sistem Kerunutan secara otomatis diperlukan untuk membangun ketelusuran antar artefak Penelitian tentang pendekatan untuk membangan Kerunutan telah dimulai sejak tahun 1984 oleh Dorfman dan Flynn dengan membangun kakas untuk kerunutan dengan nama ART (Automated Requirements Traceabillity) (Dorfman, 1984). Hingga saat ini penelitian Kerunutan masih terbuka dan masih memiliki banyak tantangan, terutama dalam meminimalisasi kesalahan keterhubungan antarartefak (Kannenberg, 2009). Penelitian yang dilakukan oleh Siti Rochimah, dkk pada tahun 2007 mengevaluasi berbagai pendekatan untuk kerunutan dengan menggunakan kerangka kerja taksonomi evolusi perangkat lunak yang diusulkan oleh Buckley (Rochimah, 2007).

Hubungan kerunutan dengan artefak secara tekstual diperlukan untuk menghubungkan cerita pengguna dengan kode sumber menggunakan pencocokan string dengan kemiripan

### 2.2 Similarity

Similarity atau similaritas merupakan tingkat perbandingan persentase kemiripan antar dokumen yang diuji. Similarity ini akan menghasilkan nilai dimana nilai tersebut yang akan dijadikan acuan dalam menentukan persentase tingkat kemiripan sebuah dokumen yang diuji. Besarnya persentase similarity akan dipengaruhi oleh tingkat kemiripan dari dokumen yang diuji semakin besar persentase similarity maka semakin tinggi tingkat kemiripan dokumen
K-Gram adalah rangkaian terms dengan panjang K. Kebanyakan yang digunakan sebagai terms adalah kata. K-Gram merupakan sebuah metode yang diaplikasikan untuk pembangkitan kata atau karakter. Metode K-Gram ini digunakan untuk mengambil potongan-potongan karakter huruf sejumlah k dari sebuah kata yang secara kontinuitas dibaca dari teks sumber hingga akhir dari dokumen. Dalam Markov Model nilai K-Gram yang sering digunakan yaitu, 2-gram (bigram), 3-gram (trigram) dan seterusnya disebut K-Gram (4-gram, 5-gram dan seterusnya). Dalam natural language processing, penggunaan K-Gram (atau lebih dikenal dengan n-gram), proses parsing token (tokenisasi) lebih sering menggunakan 3-gram dan 4-gram, sedangkan 2-gram digunakan dalam parsing sentence, misal dalam part-of-speech (POS). Penggunaan 2-gram dalam tokenisasi akan menyebabkan tingkat perbandingan antar karakter akan semakin besar. Contohnya pada kata “makan” dan “mana” yang merupakan dua kata yang sama sekali berbeda. Dengan menggunakan metode bigram dalam mencari similarity, hasil dari bigram tersebut yaitu kata “makan” akan menghasilkan bigram ma, ak, ka, an serta kata “mana” akan menghasilkan bigram ma, an, na. Dengan demikian, akan terdapat kesamaan dalam pengecekannya similarity. Namun jika menggunakan 3-gram (“makan” = mak, aka, kan dan “mana” = man, ana) atau 4-gram (“makan” = maka, akan, dan “mana” = mana) akan mengecilkan kemungkinan terjadinya kesamaan pada kata yang strukturnya berbeda.

### 2.3 Konversi Penamaan Java

Konvensi penamaan pada java dibuat agar kode program lebih mudah dimengerti dan lebih mudah dibaca. Konvensi ini juga dapat memberikan informasi tentang fungsi dari identifier, apakah itu konstanta, paket, atau kelas. Aturan konvensi penamaan telah ditetapkan oleh Sun Microsystems, Inc. Konvensi Penamaan pada java. (Sun, 1995-1999).

### 2.4 Kode Sumber

Kode sumber adalah suatu rangkaian peryataan atau deklarasi yang ditulis dalam bahasa pemrograman komputer yang terbaca manusia. Kode sumber yang menyusun suatu program biasanya disimpan dalam satu atau lebih berkas teks (Juergen , 2007). Kode sumber biasanya ditransformasikan (di-compile) oleh compiler program dari kode sumber menjadi low level machine code yang dapat dimengerti oleh computer atau menjadi executable file.

### 2.5 Cerita Pengguna

Cerita Pengguna sebagai bagian Spesifikasi Kebutuhan Perangkat Lunak (SKPL). Cerita Pengguna merupakan bentuk utama dari ekspresi kebutuhan *(requirement expression)* yang digunakan dalam pengembangan cepat (*Agile*), SKPL akan menjadi kompilasi dari Cerita Pengguna dan SKPL berkembang sebagai suatu sistem yang berkembang.

Setiap Cerita Pengguna dalam agile merupakan pernyataan singkat yang digunakan untuk menguraikan apa yang sistem perlu lakukan untuk pengguna. Biasanya, setiap cerita pengguna dinyatakan dalam bentuk sebagai berikut:

*As a <role> I can <activity> so that < value>*
Dimana *Role* menggambarkan siapa yang melakukan aksi. *Activity m*enggambarkan aksi yang dilakukan. *Value* menggambarkan nilai yang dihasilkan dari aksi yang dilakukan
Contoh dari penulisan cerita pengguna dalam bahasa Indonesia adalah sebagai berikut:

"Sebagai Kasir Saya dapat Mencetak Laporan Setiap Shift sehingga saya mengetahui berapa uang kas, kartu kredit maupun kartu debit yang saya terima."

Untuk menyatakan bahwa Cerita Pengguna telah sesuai dengan yang diinginkan atau dianggap selesai oleh user maka Cerita Pengguna harus dilengkapi dengan kriteria penerimaan (Acceptance Criteria). Kriteria penerimaan menentukan batasan dari cerita pengguna, serta mengkonfirmasi bahwa cerita pengguna telah lengkap, selesai dan dapat bekerja sebagaimana dimaksud.

Dari contoh diatas dapat dilengkapi dengan kriteria penerimaan sebagai berikut:

- Terdapat pilihan periode shift untuk setiap kasir
- Dapat mencetak rekap seluruh kasir setiap shift
- Laporan dapat secara otomatis menambahkan kolom jenis pembayaran

## 2.6 Sistem Temu Kembali Informasi

Sistem Temu Kembali Informasi atau *Information Retrieval* (IR) merupakan sistem yang berfungsi untuk menemukan informasi yang relevan dengan kebutuhan pemakai. Salah satu hal yang perlu diingat adalah bahwa informasi yang diproses terkandung dalam sebuah dokumen yang bersifat tekstual. Dalam konteks ini, temu kembali informasi berkaitan dengan representasi, penyimpanan, dan akses terhadap dokumen representasi dokumen. Dokumen yang ditemukan tidak dapat dipastikan apakah relevan dengan kebutuhan informasi pengguna yang dinyatakan dalam query. Pengguna Sistem Temu Kembali informasi sangat bervariasi dengan kebutuhan informasi yang berbeda-beda. Tahapan yang akan digunakan dalam IR secara umum adalah sebagai *Indexing, Searching*, Perengkingan relevansi *keyword query.*

Konsep dasar dalam IR System terdiri dari *Indexing, Searching* dan perengkingan relevansi *keyword query*. Dimana proses indexing dilakukan untuk membentuk database index terhadap koleksi dokumen yang dimasukkan, atau dengan kata lain, indexing merupakan proses persiapan yang dilakukan terhadap dokumen sehingga dokumen siap untuk retrive. Proses indexing sendiri meliputi 2 proses, yaitu dokumen indexing dan term indexing. Dari term indexing akan dihasilkan koleksi kata yang akan digunakan untuk meningkatkan performansi pencarian pada tahap selanjutnya. Tahap-tahap dalam proses indexing ialah *Word Token / Parsing, Stopword Removal / filtering, Stemming, TF/IDF (Term Frequency – Inversed Document Frequency.*

### 2.6.1 Word Token/Parsing

Untuk pemrosesan, dokumen dipilih menjadi unit-unit yang lebih kecil contohnya berupa kata, frasa atau kalimat. Unit hasil pemrosesan disebut sebagai token. Dalam proses ini biasanya juga digunakan sebuah daftar kata yang tidak digunakan (stoplist) karena tidak signifikan dalam membedakan dokumen atau kueri, misalnya kata-kata tugas seperti yang, hingga, dan dengan. Proses parsing akan menghasilkan daftar istilah beserta informasi tambahan seperti frekuensi dan posisi yang akan digunakan dalam proses selanjutnya (Mark , 2007).

Dalam bidang linguistik dan tata bahasa, parsing merupakan suatu proses yang dilakukan untuk memilah dokumen menjadi unit-unit yang lebih kecil berupa kata atau frasa . Unit hasil pemrosesan disebut sebagai token. Dalam proses ini biasanya juga digunakan sebuah daftar kata yang tidak digunakan (stoplist) karena tidak signifikan dalam membedakan dokumen atau kueri, misalnya kata-kata tugas seperti yang, hingga, dan dengan. Proses parsing akan menghasilkan daftar istilah beserta informasi tambahan seperti frekuensi dan posisi yang akan digunakan dalam proses selanjutnya.

### 2.6.2 Stemming

*Stemming* adalah proses penghilangan prefiks dan sufiks dari kueri dan istilah-istilah dokumen. Stemming dilakukan atas dasar asumsi bahwa kata-kata yang sama memiliki makna yang serupa. Dalam hal keefektifan stemming dapat meningkatkan recall dengan mengurangi bentuk-bentuk kata ke bentuk kata dasarnya. Selain itu proses stemming juga dapat mengurangi ruang penyimpanan indeks. (Mark , 2007)

Stemming adalah salah satu cara yang digunakan untuk meningkatkan kemampuan IR dengan cara mentransformasi kata-kata dalam kalimat atau dokumen teks menjadi kata dasar atau kata akarnya. Algoritma Stemming yang banyak digunakan ditujuka untuk bahasa Inggris namun bahasa Inggris memiliki morfologi yang berbeda dengan Bahasa Indonesia sehingga algoritma stemming untuk kedua bahasa tersebut juga berbeda. Proses stemming pada teks berbahasa Indonesia lebih rumit/kompleks karena terdapat variasi imbuhan yang harus dibuang untuk mendapatkan kata dasar dari sebuah kata. Beberapa algoritma stemming Bahasa Indonesia telah dikembangkan sebelumnya. Penggunaan algoritma stemming yang sesuai mempengaruhi performa sistem IR. Dalam penelitian Jelita telah membandingkan lima algoritma stemming dengan

hasil perbandingan: Nazief yang terbaik dengan tingkat kebenaran 93% dari seluruh kata yang diuji dan 92% dari kata-kata yang unik, pada peringkat berikutnya adalah Ahmad dengan nilai 88.8%, Idris 87.9%, Arifin 87.7%, dan Vega 66.3%. Vega memiliki nilai terendah karena satu-satunya yang tidak menggunakan kamus. (jelita, 2005)

Algoritma-algoritma stemming memiliki kelebihan dan kekurangannya masing-masing. Efektifitas algoritma stemming dapat diukur berdasarkan beberapa parameter, seperti kecepatan proses, keakuratan, dan kesalahan. Perbandingan efektifitas algoritma Nazief dan Adriani dengan algoritma Porter untuk proses stemming pada teks berBahasa Indonesia. Kesimpulan dari peneletian menyebutkan bahwa:
Proses stemming dokumen teks berBahasa Indonesia menggunakan Algoritma Porter membutuhkan waktu yang lebih singkat dibandingkan dengan stemming menggunakan Algoritma Nazief & Adriani.

Proses stemming dokumen teks berBahasa Indonesia menggunakan Algoritma Porter memiliki prosentase keakuratan (presisi) lebih kecil dibandingkan dengan stemming menggunakan Algoritma Nazief & Adriani.
Pada proses stemming menggunakan Algoritma Nazief & Adriani, kamus yang digunakan sangat mempengaruhi hasil stemming. Semakin lengkap kamus yang digunakan maka semakin akurat pula hasil stemming.

Kamus yang digunakan mempengaruhi perhitungan presisi. Semakin lengkap kamus yang digunakan maka semakin akurat pula nilai presisinya (jelita, 2005). Dari kesimpulan tersebut maka penulis akan menggunakan Algoritma Nazief & Adriani karena penulis membutuhkan akurasi daripada kecepatan. Selain itu penulis mencoba untuk menggunakan kamus resmi bahasa Indonesia dalam Kamus Besar Bahasa Indonesia (KBBI) untuk menghasilkan kata dasar yang lebih akurat.

### 2.7 Sistem Temu Kembali Informasi

Dalam pengenalan pola (*pattern recognition*) dan temu kembali informasi (*information retrieval*) dengan klasifikasi biner, precision/presisi (juga disebut nilai prediktif positif) adalah fraksi contoh diambil yang relevan atau tingkat ketepatan antara informasi yang diminta oleh pengguna dengan jawaban yang diberikan oleh system. Sementara *Recall* (juga dikenal sebagai sensitivitas) adalah fraksi contoh yang relevan yang diambil atau tingkat keberhasilan sistem dalam menemukan kembali sebuah informasi. Sedangkan akurasi didefinisikan sebagai tingkat kedekatan antara nilai prediksi dengan nilai aktual. Untuk memudahkan pemahaman pada presisi, recall dan akurasi dapat dilihat pada Tabel 1.

Tabel 1.
Rumusan precision recall dan accuracy

| | | Nilai Sebenarnya | | Total |
|---|---|---|---|---|
| | | True | False | |
| Nilai Prediksi | True | TP (true positive) relevan/ hasil benar | FP (false positive) tidak relevan/ hasil tidak diharapkan | TP + FP |
| | False | FN (false negative) relevan, hasil yang hilang | TN (true negative) tidak relevan, hasil yang tidak benar | FN+TN |
| | Total | TP+FN | FP+TN | TP+FP+FN+TN |

$$Precision = \frac{TP}{TP + FP} \quad (1)$$

$$Recall = \frac{TP}{TP + FN} \quad (2)$$

$$Accuracy = \frac{TP + TN}{\mathrm{TP} + \mathrm{FP} + \mathrm{FN} + \mathrm{TN}} \quad (3)$$

## 3. MODEL HUBUNGAN KERUNTUNAN

### 3.1 Automatic Traceability Recovery : An Ontological Approch

Hubungan kerunutan membantu pembuat perangkat lunak memahami hubungan dan saling ketergantungan antar perangkat lunak artefak. Pada kenyataanya, artefak perangkat lunak yang dibuat sebagai bagian dari proses-proses ini akhirnya akan terputus dari satu sama lain. Dalam penelitian ini menyajikan pendekatan baru untuk masalah ketertelusuran (traceability) dengan menciptakan representasi formal ontological dari dokumentasi perangkat lunak dengan kode sumber.

Representasi yang dihasilkan kemudian disesuaikan untuk membangun link ketertelusuran (traceability link) pada tingkat semantik. Juergen Rilling dkk menggunakan Text Mining (TM) untuk menganalisis dokumentasi perangkat lunak dan parsing kode sumber untuk menganalisis kode sumber. Sebagai gambaran umum dari pendekatan metodologi ini ditunjukkan pada Gambar 1. Yang pertama, ontologi yang sudah ada secara otomatis diisi dari kode sumber dan dokumentasi artefak. Pada tahap kedua, dasar pengetahuan yang dihasilkan dieksplorasi untuk membangun hubungan ketertelusuran guna memberikan dukungan bagi pengelola.

### 3.2 Recovering and Using Use Case Diagram to Source Code Traceability (Mark Grechanik et all)

Pada paper ini terdapat dua kontribusi yaitu yang pertama, diwarkan pendekatan baru untuk mengotomatisasi bagian dari proses *traceability links* (TLs) antara jenis dan variabel dalam program Java dan unsur-unsur UCDs. *Prototipe* yang dihasilkan dievaluasi pada perangkat lunak kode terbuka dan komersial, dan hasilnya menunjukkan bahwa pendekatan yang digunakan dapat memulihkan TLs dengan otomatisasi tingkat tinggi dan presisi. Kedua, dikembangkan sebuah plugin Eclipse yang memungkinkan program untuk melacak jenis program dan variabel untuk unsur UCDs dan menggunakan TLs. Studi kasus yang dilakukan menunjukkan bahwa *developer* mengembangkan perangkat lunak lebih efisien dengan plugin yang dikembangkan. Developer mengembangkan TLs secara bersama-sama untuk mengurangi beban developer.

Otomatisasi dalam proses *traceability links* (TLs) untuk parameter jenis dan variabel dalam program java pada Use Case Diagram (UCDs). Pengujian dilakukan pada perangkat lunak kode terbuka dan komersial. Teknik yang digunakan dengan *LEarning and ANAlyzing Requirements Trace-ability* (LeanArt) sesuai Gambar 2.. Penerapan sistem ini di pengujian pada *Vehicle Maintenance Tracker* (VMT) project. Dengan menghubungkan kode sumber dengan *Use Case Diagram* maka menghasilkan pendekatan yang dilakukan dapat *Recovering traceability links* dengan sangat sesuai. *(Mark , 2007)*

### 3.3 Traceability Management Architectures Supporting Total Traceability In The Context of Software Engginering (Hector Garcia, et all)

Tujuan utama dalam *traceability* perangkat lunak adalah untuk melacak semua elemen yang dapat dianggap cukup relevan bagi organisasi dalam proyek tertentu atau produk software. Beberapa contoh klasik dari unsur-unsur persyaratan, desain, kode sumber atau tes. Namun ada sejumlah informasi yang dianggap tidak hati-hati dalam literatur saat ini. Email yang dikirim oleh para pemangku kepentingan, risalah rapat, proposal proyek atau analisis biaya manfaat adalah dokumen yang juga merupakan bagian penting dari produk perangkat lunak, mempertahankan sejumlah besar pengetahuan yang diperlukan oleh organisasi (misalnya untuk mengelola perbaikan proses dan penentuan kemampuan) *(Héctor , 2007)*

Pada Gambar 3 menggambarkan konsep traceability total. Sebuah proyek perangkat lunak tidak mulai dalam tahap rekayasa persyaratan, seperti yang ditunjukkan oleh banyak penulis. Selain itu, kita tidak harus mempertimbangkan persyaratan sebagai inti dalam traceability, bahkan ketika perspektif yang lebih luas, termasuk informasi yang tidak secara langsung berkaitan dengan proses rekayasa seperti yang dijelaskan dalam.

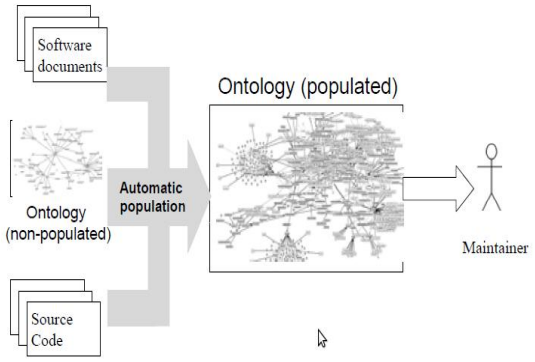
*Hector dkk* mengklaim bahwa *software traceability* harus difokuskan dalam membangun hubungan antara unsur-unsur yang membentuk produk, terlepas dari jenis atau tahap di mana pertama kali muncul. Gambar 3 menunjukkan kasus hipotetik sederhana,sehingga kita dapat dengan mudah menemukan di dunia nyata. Siklus hidup produk dimulai ketika pengguna mengirimkan email yang meminta untuk beberapa jenis pertemuan dalam kebutuhan perangkat lunak.

### 3.4 Perbandingan Hubungan Kerunutan Kode Sumber

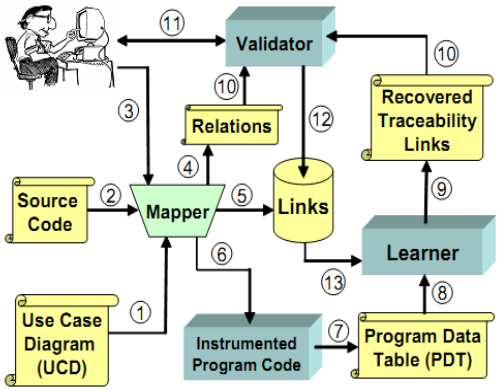
Dalam penelitian yang dilakukan oleh Juergen Rilling dkk, menyajikan pendekatan baru untuk masalah kerunutan dengan menciptakan representasi formal ontological dari software documents dengan source code. Representasi yang dihasilkan kemudian disesuaikan untuk membangun link kerunutan (Kerunutan link) pada tingkat semantik. TLs, pada penelitian yang dilakukan Juergen Rilling dkk, menghubungkan antara software documents yang berisi mulai dari Software Requierement Specification (SRS), Software Design Documentation (SDD) dihubungkan langsung dengan Source Code, sedangkan menurut Héctor García dkk terlihat di gambar 2., TLs tidak terhubung langsung pada SRS tetapi melalui Diagram Kelas. Sedangkan menurut Mark Grechanik dkk TLs menghubungkan dari Diagram Kasus Pengguna dengan Source Code. Pada rencana penelitian yang ditawarkan, akan menghubungkan Cerita pengguna (US) dengan Java Kode sumber (SC) dengan mencari kemiripan kata yang ada pada US dan SC menggunakan String Matching yang telah tersedia dalam PostgreSQL, PostgreSQL dalam kakas terpisah menyediakan fungsi similaritas untuk mencocokkan dua kata yang berbeda dengan algoritma trigram.

Tabel 2
Perbandingan penelitia sebelumnya

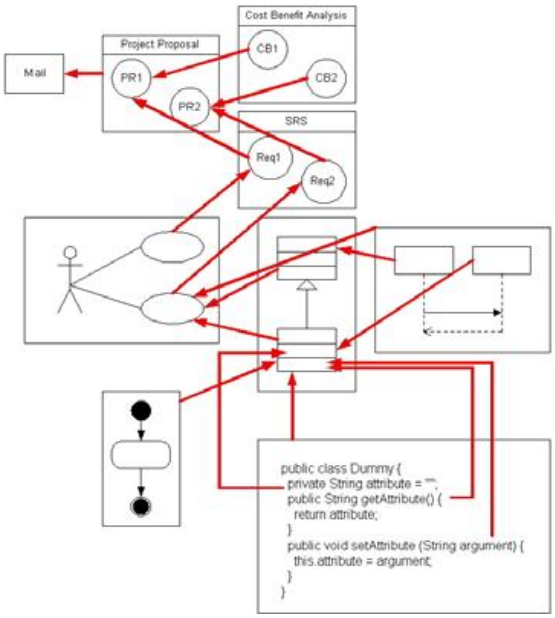
| Penulis | SC TLs | Metode |
|---|---|---|
| **Juergen Rilling** | Dok. PL dan Kode Sumber | Ontologi |
| **Mark Grechanik** | Diagram Kasus Pengguna dan Kode Sumber | Ontologi |
| **Héctor García** | Diagram Kelas dan Kode Sumber | Kecocokan |




**Gambar 1.** Model *Traceability link* dengan pendekatan ontologi




**Gambar 2.** *Model . LeanART architecture (Mark,2007)*



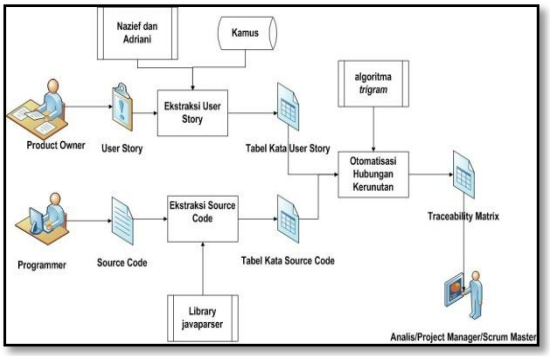

**Gambar 3** *Basic example on total traceability (Héctor , 2007)*

## 4. MODEL HUBUNGAN KERUNUTAN

### 4.1 Rancangan Sistem Otomatisasi Hubungan Kerunutan

Dalam membangun hubungan kerunutan yang menghasilkan matrix hubungan kerunutan. Sistem memiliki dua masukan yaitu Cerita Pengguna dari pengguna dan kode sumber dari programmer, sedangkan luaran yang dihasilkan adalah matrik kerunutan.

Mekanisme untuk menghasilkan matrik kerunutan hubungan diperlukan tiga proses utama yaitu proses pertama adalah ekstraksi kata dari cerita pengguna yang ditulis secara bebas berdasarkan format penulisan cerita pengguna. Proses kedua adalah ekstraksi kata dari kode sumber yang telah dibuat oleh programmer. Proses ketiga adalah mencocokan kata yang ada pada table cerita pengguna dengan daftar kata yang ada pada kode sumber apabila terdapat kesamaan kata pada tingkat tertentu maka ditentukan terjadi hubungan kerunutan antara cerita pengguna dengan kode sumber tersebut, selanjutnya cerita pengguna dengan kode sumber yang terhubung tersebut ditandai dalam matrik hubungan kerunutan. Gambaran umum dari perancangan sistem digambarkan pada Gambar 4.




**Gambar 4**. Sistem Otomatisasi Hubungan Kerunutan

Dari gambar 4 dapat dijelaskan peran dari masing-masing blok diagram diatas sebagai berikut:

#### *4.1.1 Product Owner*

Product Owner bertanggung-jawab untuk memaksimalkan nilai produk dan hasil kerja Tim Pengembang. Cara pelaksanaannya sangat bervariasi antar organisasi, Tim Scrum dan individu. Product Owner merupakan satu-satunya orang bertanggung-jawab untuk mengelola Product Backlog. Product owner bersama stakeholder yang lain menyampaikan kebutuhan system kepada tim yang selanjutnya tim scrum membantu menuliskan kebutuhan system tersebut

dalam bentuk Cerita Pengguna yang dilengkapi dengan kriteria penerimaan (acceptance criteria).

### *4.1.2 Cerita Pengguna*

Pada tahun 1999, *Kent Beck* mulai memperkenalkan istilah Cerita Pengguna untuk fitur produk perangkat lunak. Cerita Pengguna dituliskan dari perspektif pengguna mengenai apa yang pengguna inginkan pada system serta apa kelebihan yang sistem bisa lakukan untuknya. Dengan demikian, pandangan benar-benar berubah dari produk ke pengguna dan Cerita Pengguna menjadi *de facto* standar untuk kebutuhan pada *Agile framework*. Pada penelitian ini Cerita Pengguna menjadi obyek utama penelitian yang akan diekstraksi menjadi potongan-potongan kata dasar yang nantinya akan dihubungkan dengan kode sumber, Cerita Pengguna dinyatakan dalam bentuk sebagai berikut seperti pada bagian II-F.

### *4.1.3* Kode sumber

Kode sumber adalah suatu rangkaian peryataan atau deklarasi yang ditulis dalam bahasa pemrograman komputer yang terbaca manusia. Kode sumber yang menyusun suatu program biasanya disimpan dalam satu atau lebih berkas teks. Selain cerita pengguna, kode sumber juga merupakan bagian inti dari penelitian ini karena hasil ekstraksi kode sumber akan dihubungkan dengan hasil ekstraksi dari Cerita Pengguna yang hasilnya berupa matrik Kerunutan.

### 4.1.4 Ekstraksi Cerita Pengguna

Ekstraksi Cerita Pengguna adalah proses merubah kalimat bebas yang terdapat dalam Cerita Pengguna menjadi daftar kata dasar dalam table basis data. Proses ini dimulai dari mengkodekan masing-masing cerita pengguna, parsing, stop word removal, stemming menggunakan Algoritma Nazief dan Andriani hingga terbentuk kata dasar yang selanjutnya disimpan didalam table daftar kata.

### 4.1.5 Ekstraksi Source Code

Ekstraksi Kode Sumber adalah proses parsing dari kode sumber menjadi daftar nama class, method dan attribute dari setiap class dalam kode sumber java menggunakan library javaparser

Tabel Kata Cerita Pengguna

Setiap kata hasil stemming akan disimpan dalam table daftar kata yang sudah berupa kata dasar, yang terdiri dari bagian role, activity, value dan acceptance criteria untuk masing-masing Cerita pengguna.

### 4.1.6 Tabel Kata Source Code

Hasil dari ekstraksi kode sumber berupa Nama Class, Nama Method dan Atribut akan disimpan kedalam Tabel Kode Sumber.

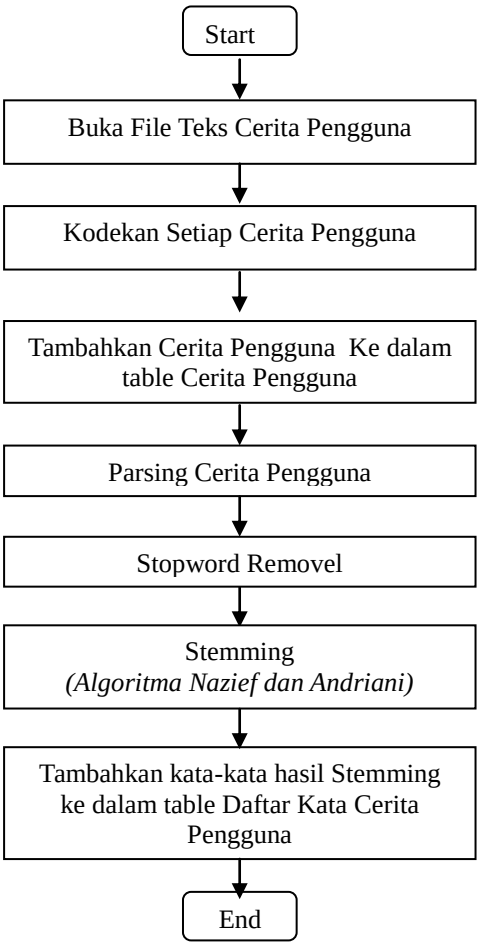
### 4.1.7 Otomatisasi Hubungan kerunutan

Otomatisasi hubungan kerututan ini akan dibangun dengan cara menghubungkan daftar kata dalam Cerita Pengguna dengan kode sumber menggunakan string matching dengan algotirma.

## 4.2 Ekstraksi Cerita Pengguna

Ekstraksi Cerita Pengguna adalah proses merubah kalimat bebas dalam Cerita Penggunamenjadi daftar kata dasar dalam table basis data seperti tampak pada gambar gambar 5, dengan langkah-langkah sebagai berikut:

1. Membuka File Teks Cerita Pengguna (US), file Cerita Penggunadisimpan dalam teks file dibuka dan selanjutnya dipilah per Cerita Pengguna .
2. Setiap US yang telah dipilah diberi kode unik sebagai identitas masing-masing US.
3. US yang telah diberi kode disimpan kedalam table database.
4. US di dalam table selanjutnya diparsing menjadi potongan-potongan kata dan membuang semua tanda baca.




**Gambar 5** . *Ekstraksi Cerita Pengguna*

5. Proses stopword removel membersihkan kata-kata hasil parsing dengan membandingkan setiap kata dengan daftar kata yang tidak digunakan (stoplist). Jika kata-kata ditemukan dalam stoplist maka kata-kata tersebut dihilangkan dari daftar kata yang akan digunakan.
6. Hasil dari proses 5 selanjutnya dilakukan stemming setiap pada kata dengan algoritma

Nazief dan Andriani yang ditambahkan dengan daftar kata dalam kamus bahasa Indonesia (KBBI).

7. Setiap kata hasil stemming akan disimpan dalam table daftar kata yang sudah berupa kata dasar, yang terdiri dari bagian role, activity, value dan acceptance criteria untuk masing-masing Cerita pengguna.

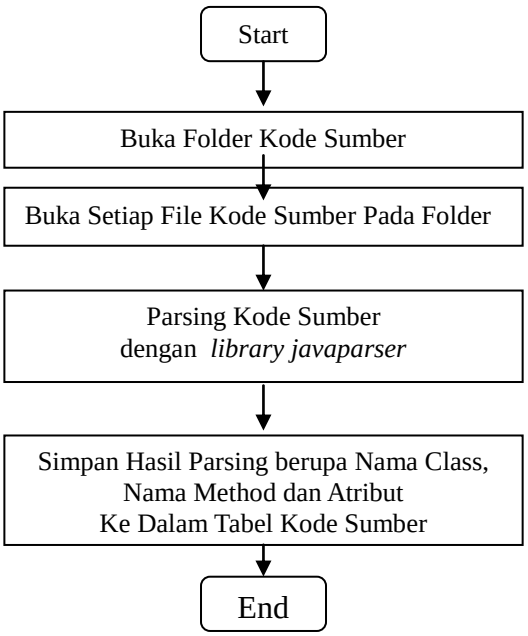
### 4.3 Ekstraksi Kode Sumber

Ekstraksi Kode Sumber adalah proses parsing dari kode sumber menjadi daftar nama class, method dan attribute dari setiap class seperti tampak pada Gambar 6, dengan langkah-langkah sebagai berikut:

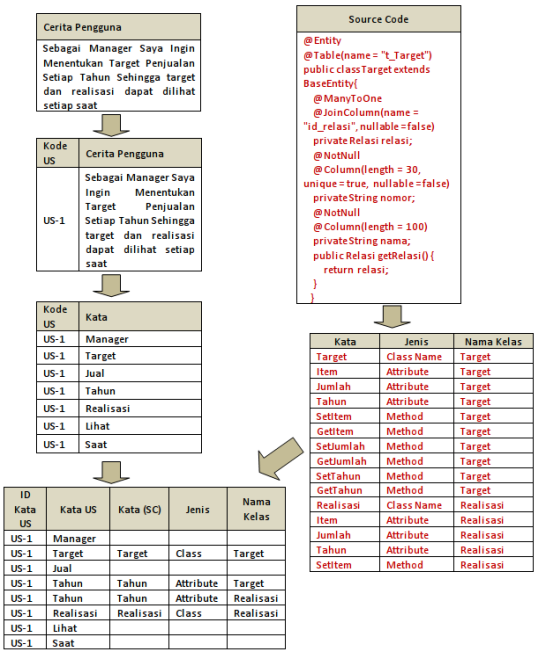
1. Membuka Folder Project untuk Kode Sumber (KS)
2. Buka setiap file KS didalam folder.
3. Parsing file KS menggunakan Parsing library javaparser
4. Simpan Hasil Parsing berupa Nama Class, Nama Method dan Atribut ke Dalam Tabel Kode SumberMembangun

### 4.4 Hubungan Kerunutan

Membangun Kerunutan Hubungan antara tabel Cerita Pengguna dan tabel kode sumber menggunakan string matching dengan algoritma trigram yang telah diintegrasikan dengan postgreSQL (pg_trgrm). Indentifikasi adanya hubungan antara US dan SC menggunakan query ini akan didapat kesimpulan apakah US dan SC telah terhubung atau belum, seperti tampak pada Gambar 7, dengan langkah-langkah seperti pada gambar tersebut.




**Gambar 7**. Ekstraksi Kode Sumber




**Gambar 7**. Ilustrasi pembentukan matri hubungan kerunutan

## 5. HASIL UJI COBA

### 5.1 Dataset

Dataset yang digunakan adalah tiga project pembangunan perangkat lunak pada berbagai instansi pemerintah dan swasta. Dataset pertama diimplementasikan pada Dinas Pendapatan Pengelolaan Keuangan dan Aset Daerah pada salah satu kabupaten di Jawa Tengah. Aplikasi ini digunakan untuk memungut retribusi truk pengangkut galian C. Aplikasi ini dibangun dengan bahasa pemrograman java desktop dengan swing yang selanjutnya disebut smartPortal.

Dataset yang kedua adalah aplikasi absensi online yang diimplementasikan pada salah satu Pemerintah Kota di Jawa Timur. Aplikasi ini digunakan untuk mengintegrasikan mesin-mesin absensi baik yang menggunakan fingerprint maupun face recognition. Data dari masing-masing mesin absensi setiap SKPD atau untit kerja secara otomatis terekam pada server database di Badan Kepegawaian Daerah yang selanjutnya disebut smartAbsensi.

Dataset yang ketiga adalah aplikasi tour dan travel yang diimplementasikan pada salah satu perusahaan tour travel di wilayah Jawa Timur. Aplikasi ini digunakan untuk melakukan resevasi dan pembelian tiket pesawat, paket tour, hotel, dan penyewaan mobil, yang selanjutnya disebut smartTravel.

Identitier kode sumber pada semua dataset masih menggunakan bahasa campuran yaitu bahasa Inggris dan Indonesia, serta masih banyak terdapat singkatan-singkatan, sehingga hasil

identifikasi system menjadi tidak optimal. Pada masa mendatang diharapkan terdapat dataset dengan penulisan kode sumber yang benar dan disertai cerita pengguna.

Tabel 3. Daftar Dataset

| No | Nama Aplikasi | Ukuran | Jumlah Cerita Pengguna | Jumlah File Progrm Java |
|---|---|---|---|---|
| **1.** | smartPortal | Kecil | 17 | 52 |
| **2.** | smartAbsensi | Sedang | 47 | 161 |
| **3.** | smartTravel | Besar | 80 | 222 |

## 5.2 Implementasi

Proses identifikasi kerunutan hubungan dilakukan secara otomatis. Proses otomatis dilakukan dengan menggunakan kakas bantu. Kakas bantu dibangun dengan bahasa pemrograman Java, sistem basis data PostgreSQL versi 9.4, proses ekstraksi kode sumber dibantu dengan javaparser library dan menerapkan teori yang sudah dipaparkan pada bab sebelumnya. Proses otomatis menyelesaikan tiga proses pada metodologi yang diusulkan dalam penelitian ini, yaitu proses ekstraksi cerita pengguna, proses ekstraksi kode sumber, dan proses otomatisasi hubungan kerunutan antara cerita pengguna dengan kode sumber.

## 5.3 Ekstraksi Cerita Pengguna

Proses ekstraksi cerita pengguna seperti yang telah dijelaskan pada bagian metodologi dimulai dari proses kodefikasi cerita pengguna, parsing, stopword removel, stemming menggunakan algoritma Nazief dan Andrian dan dibantu dengan KBBI (Kamus Besar Bahasa Indonesia), hasil dari ekstraksi ini disimpan kedalam table database user_story dengan struktur table seperti pada Tabel 4.

Tabel 4
Tabel user_story

| Nama Kolom | Tipe data | Keterangan |
|---|---|---|
| **id_project** | character varying(10) | Kode proyek |
| **id_user_story** | character varying(10) | Kode cerita pengguna |
| **user_story** | Text | Isi cerita pengguna |

Sebagai contoh setelah system membaca seluruh file text cerita pengguna maka setiap serita pengguna disimpan pada tabel diatas, misalnya terdapat cerita pengguna sebagai berikut:

```
Sebagai pengguna saya ingin memiliki halaman login  sehingga saya dapat masuk kedalam sistem menggunakan user dan password.
```

Gambar 8. Contoh cerita pengguna

Cerita pengguna pada gambar 8 akan disimpan kedalam tabel user_story seperti pada Tabel 5.

Tabel 8.
Contoh isi tabel user_story

| id_project | id_user_story | user_story |
|---|---|---|
| **P001** | P001-001 | Sebagai pengguna saya ingin memiliki halaman login sehingga saya dapat masuk kedalam sistem menggunakan user dan password. |

Proses selanjutnya adalah dari setiap baris tabel user_story akan dilakukan proses parsing menjadi potongan-potongan kata dan membuang semua tanda baca. Setelah proses parsing dilakukan proses membersihkan kata-kata hasil parsing yang tidak diperlukan dengan membandingkan setiap kata dengan daftar kata yang tidak digunakan (stoplist). Jika kata-kata ditemukan dalam stoplist maka kata-kata tersebut dihilangkan dari daftar kata yang akan digunakan. Kata-kata yang ditemukan selanjutnya dilakukan stemming setiap pada kata dengan algoritma Nazief dan Andriani yang ditambahkan dengan daftar kata dalam kamus besar bahasa Indonesia (KBBI). Hasil dari proses ini berupa kata dasar yang selanjutnya disiimpan kedalam tabel daftar kata yang sudah berupa kata dasar kedalam tabel user_story_kata (Tabel 9).

Tabel 9.
Tabel user_story_kata

| Nama Kolom | Tipe data | Keterangan |
|---|---|---|
| **id_user_story** | character varying(10) | Kode cerita pengguna |
| **id_kata_user_story** | Serial | Identitas unik dari setiap kata |
| **kata_user_story** | character varying(100) | Kata dari user story |

Hasil dari proses ekstraksi kata dalam cerita pengguna ini akan disimpan kedalam tabel *user_story_kata* sehingga hasil yang diperoleh dari cerita pengguna pada tabel 8 akan diproses dan hasilnya berupa kumpulan kata-kata seperti tampak pada tabel 10.

Tabel 10.
Contoh isi *tabel user_story_kata*

| id_user_story | id_user_story_kata | kata_user_story |
|---|---|---|
| **P001-001** | 1 | sebagai |
| **P001-001** | 2 | pengguna |
| **P001-001** | 3 | ingin |
| **P001-001** | 4 | milik |
| **P001-001** | 5 | halaman |
| **P001-001** | 6 | login |
| **P001-001** | 7 | dapat |
| **P001-001** | 8 | masuk |
| **P001-001** | 9 | dalam |
| **P001-001** | 10 | sistem |
| **P001-001** | 11 | user |
| **P001-001** | 12 | password |

## 5.4 Ekstraksi Kode Sumber

Proses ekstraksi kode sumber seperti telah dijelaskan pada gambar 6, proses dimulai dengan memilih lokasi atau folder dimana file kode sumber disimpan. Selanjutnya system akan mengumpulkan file yang berekstensi .java kedalam tabel source_code dengan struktur tabel seperti pada tabel 11.

Tabel 11
Tabel *source_code*

| Nama Kolom | Tipe data | Keterangan |
|---|---|---|
| **id_project** | character varying(10) | Kode proyek |
| **id_source_code** | character varying(10) | Kode Kode sumber |
| **nama_file** | Text | Path dan nama file kode sumber |

Selanjutnya sistem akan membaca seluruh isi folder yang dipilih dan menyimpan pada tabel source_code seperti contoh pada tabel 12. Contoh isi tabel source_code

Tabel 12.
Contoh isi tabel *source_code*

| id_project | id_source_code | nama_file |
|---|---|---|
| **P001** | P001-001 | ReportParam.java |
| **P001** | P001-002 | FileEncryption.java |
| **P001** | P001-003 | CipherExample.java |
| **P001** | P001-004 | CryptoUtilsTest.java |
| **P001** | P001-005 | CryptoException.java |
| **P001** | P001-006 | CryptoUtils.java |
| **P001** | P001-007 | HariLiburController.jav |
| **P001** | P001-008 | SKPDController.java |
| **P001** | P001-009 | ServiceController.java |
| **P001** | P001-010 | RoleController.java |

Proses pengkodean kode sumber seperti pada tabel 12, berfungsi untuk mempermudah proses otomatisasi kerunutan serta menyingkat nama file program menggunakan identitas kode sumber. Pada proses selanjutnya kode sumber yang tersinpan pada tabel tersebut akan diparsing satu persatu dengan bantuan *javaparser* library versi 1.0.11 (*javaparser-1.0.11.jar*). Pustaka *javaparser* sebagai kakas yang digunakan untuk melakukan parsing pada kelas, metode, komentar dan attribute sehingga dapat diidentifikasi nama kelas, nama konstruktor, nama metode, isi komentar dan nama attribute.

Pada proses parsing kode sumber melakukan parsing pada setiap file kode sumber, setiap file akan dibaca dan diamati dengan memanggil masing masing metode yang bernama visit. Didalam setiap metode visit kata yang diterima baik nama kelas, metode, atrribut dan konstruktor akan diparsing kembali sesuai dengan aturan atau konvensi penamaan java.

Untuk metode *parsing comment* memiliki isi metode yang berbeda dari yang lainnya, karena komentar biasanya berisi kalimat, sehingga masih diperlukan proses parsing, stemming sampai ditemukan bentuk kata dasarnya selanjutnya kata tersebut disimpan dalam tabel *source_code_kata* dengan struktur tabel seperti pada tabel 13.

Tabel 13.
Tabel *source_code*

| Nama Kolom | Tipe data | Keterangan |
|---|---|---|
| **id_source_code** | character varying(10) | Indentitas Kode sumber |
| **id_source_code_kata** | Serial | Identitas unik kata kode sumber |
| **nama_class** | character varying(255) | Nama kelas |
| **Jenis** | character varying(255) | Jenis Kode Sumber |
| **Kata** | character varying(255) | Kata dalam kode sumber |

Hasil dari parsing kode sumber berupa daftar kata-kata dasar yang disimpan pada tabel *source_code*, *id_source_code* akan diisi dengan kode dari setiap kode sumber yang berelasi dengan tabel source_code. *id_source_code_kata* diisi dengan angka yang secara otomatis akan dibuat oleh postgreSQL. Jenis akan diisi jenis dari kode sumber dapat berisi method, class, constructor, attribute, atau comment. Kata akan diisi dengan kata-kata dasar yang terkandung dari setiap identifier seperti contoh pada tabel 14. contoh isi tabel source_code.

Tabel 14.
Contoh isi tabel *source_code*

| id_source_code | id_source_code_kata | nama_class | jenis | kata |
|---|---|---|---|---|
| **P001-001** | 1 | Report Param | Class | report |
| **P001-002** | 2 | Report Param | Class | Param |
| **P001-003** | 3 | Report Param | Method | Bulan |
| **P001-004** | 4 | Report Param | Method | Tahun |
| **P001-005** | 5 | Report Param | Method | Tanggal1 |
| **P001-006** | 6 | Report Param | Method | Tanggal2 |
| **P001-007** | 7 | Report Param | Comment | @author |
| **P001-008** | 8 | Report Param | Comment | faheem |

## 5.5 Membangun Hubungan Kerunutan

Hubungan kerunutan antara tabel cerita pengguna dan tabel kode sumber menggunakan string matching dengan algoritma trigram yang telah diintegrasikan dengan postgreSQL (pg_trgrm). Modul pg_trgm menyediakan fungsi dan operator untuk menentukan kesamaan teks alfanumerik berdasarkan pencocokan trigram, serta kelas Operator indeks yang mendukung pencarian string yang sama secara cepat. Pada pg_trgm

karakter non-kata (non-alphanumerics) diabaikan ketika mengekstrak trigram dari string. Setiap kata dianggap memiliki dua ruang diawali dan satu ruang akhiran ketika menentukan set trigram yang terkandung dalam string. Misalnya, himpunan trigram dalam string "cat" adalah "c", "ca", "cat", dan "at". Himpunan trigram dalam string "foo | bar" adalah "f", "fo", "foo", "oo", "b", "ba", "bar", dan "ar". *(The PostgreSQL , 1996-2016)*

Pg_trgm memiliki beberapa fungsi dan operator yang dapat digunakan seperti similarity(text, text), show_trgm(text), show_limit() dan set_limit(real). Pada penelitian ini yang digunakan adalah fungsi similarity(text, text) yang akan digunakan untuk membandingkan dua string atau kata dan diambil nilai keluarannya berupa bilangan real, sebagai gambaran akan diberikan contoh pada tabel 15. Contoh query penggunaan fungsi similarity(text, text).

Tabel 15.
Contoh eksekusi fungsi *similarity*

| Query | Output |
|---|---|
| **SELECT similarity('tanggal','tanggal');** | 1 |
| **SELECT similarity('tanggal','tanggal1');** | 0,7 |
| **SELECT similarity('tanggal','tgl');** | 0,0909091 |
| **SELECT similarity('tanggal','date');** | 0 |

Dari tabel 15. dapat dilihat antara “tanggal” dan “tanggal” memiliki nilai kemiripan 1, “tanggal” dan “tanggal1” memiliki nilai kemiripan 0.7, “tanggal” dan “tgl” memiliki nilai kemiripan 0.0909091 serta “tanggal” dan “date” memiliki nilai kemiripan 0 yang artinya tidak ada satu karakterpun yang mirip antara keduanya.

Indentifikasi adanya hubungan antara cerita pengguna dengan kode sumber menggunakan query yang nantinya dapat disesuaikan ambang batasnya sebagai bahan percobaa, sehingga dapat ditentukan nilai kemiripan antara kata yang terkandung dalam cerita pengguna dengan kata yang terkandung dalam kode sumber. Hasil dari identifikasi dapat dilihat pada gambar 9.

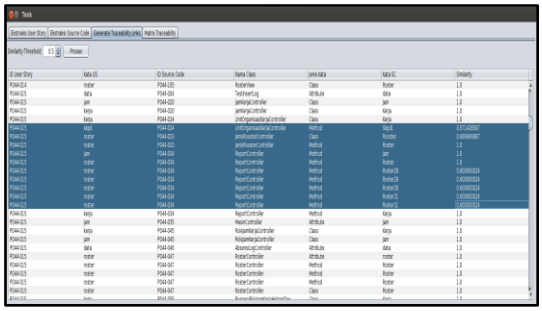
Indentifikasi adanya hubungan antara cerita pengguna dengan kode sumber menggunakan query yang dapat disesuaikan ambang batasnya sebagai bahan percobaan, sehingga dapat ditentukan nilai kemiripan antara kata yang terkandung dalam cerita pengguna dengan kata yang terkandung dalam kode sumber.

### 5.6 Membentuk Matrik Kerunutan Hubungan

Matrik hubungan kerunutan antara tabel cerita pengguna dan tabel kode sumber dibentuk berdasarkan hasil pencarian kemiripan kata antara keduanya. Apabila terdapat kata-kata yang memiliki kemiripan dengan ambang batas yang telah ditentukan maka dianggap memiliki hubungan.kerunutan. Agar lebih optimal ditambahkan perbandingan jumlah kata yang ditemukan pada kode sumber dengan cerita pengguna sebagai ambang batasnya.

Sistem ini menggunakan query untuk membentu matrik kerunutan hubungan dibentuk seperti tampak pada gambar 10. Query Pembentukan Matrik Kerunutan. Pada queri tersebut tampak terdapat 4 parameter yaitu:

- *p_kode_SC*
- *Parameter ini diisi dengan kode dari kode sumber misalnya: “P001-023”*
- *p_kode_US*
- *Parameter ini diisi dengan kode dari cerita pengguna misalnya: “P001-001”*
- *p_similarity*
- *Parameter ini diisi dengan besaran nilai ambang batas kemiripan kata misalnya: 0.5*
- *p_prosen_kata*
- *Parameter ini diisi dengan prosentase jumlah kata yang terkandung dalam kode sumber dibandingkan dengan jumlah kata yang terkandung dalam cerita pengguna.*



Gambar 9. Hasil kemiripan kata cerita pengguna dengan kode sumber

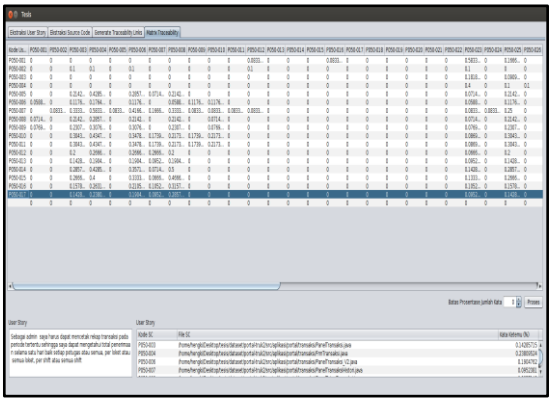
```
QUERY =#
SELECT sl.id_user_story,
   id_source_code,
   count(distinct kata_source_code)
jml_kata_sc
   count(distinct us.kata_user_story) as
jml_kata_us,
   count(distinct
kata_source_code)::real/count(distinct
   us.kata_user_story)::real as
persen_kata
   FROM (SELECT
id_source_code,nama_class,jenis,kata as
        kata_source_code, id_user_story,
kata_user_story,
   similarity(kata,kata_user_story)
        FROM source_code_kata,
user_story_kata
            WHERE id_source_code =
p_kode_SC
                and id_user_story =
P_kode_US
                and
similarity(kata,kata_user_story) >=
                P_Similarity
            ORDER BY id_user_story,
id_source_code
```

```
   ) as sl
INNER JOIN user_story_kata us on
   sl.id_user_story = us.id_user_story
GROUP BY sl.id_user_story,id_source_code
HAVING count(distinct

    kata_source_code)::real/count(distinct

    us.kata_user_story)::real >
P_Prosen_Kata
```

| Hasil Query | |
|---|---|
| *id_user_story* | *"P001-001"* |
| *id_source_code* | *"P001-023"* |
| *jml_kata_sc* | *7* |
| *jml_kata_us* | *12* |
| *persen_kata* | *0.583333* |

**Gambar 10.** Query matrik hubungan kerunutan dan hasilnya

Hasil dari query ini seperti tampak pada gambar 10 yang menunjukkan nilai 0.583333 artinya jumlah kata yang mirip dari kode sumber yang terkandung dalam cerita pengguna sebesar 58% lebih. Apabila query tersebut diulang untuk semua kata pada kode sumber dan cerita pengguna maka dihasilkan matrik kerunutan hubungan seperti tambak pada Gambar 11.



Gambar 11. Matrik kerunutan hubungan

## 5.7 Skenario Uji Coba

Pengujian terhadap metode yang ditawarkan pada penelitian ini dilakukan dengan mengimplementasikan alur kerja yang sudah diusulkan dan dijelaskan pada metodologi. Alur kerja tersebut diimplementasikan pada dataset yang akan menjadi bahan uji coba. Dataset yang digunakan untuk uji coba dibagi menjadi tiga kategori yaitu ukuran kecil, sedang dan besar.

Pengujian dilakukan dengan membandingkan hasil otomatisasi identifikasi kerunutuan dengan hasil dari identifikasi yang dilakukan oleh pengembang masing-masing dataset. Pengembang yang dipilih untuk melakukan identifikasi adalah orang-orang yang terlibat mulai dari pendefinisian kebutuhan sampai dengan programmer yang terlibat pembuatan kode sumber dari masing-masing dataset.

Proses identifikasi oleh sistem dilakukan secara otomatis dengan mencoba memberikan beberapa nilai threshold yang berbeda, hasil dari masing-masing ujicoba akan disimpan dalam database. Hasil identifikasi system tersebut selanjutnya dianalisis dengan cara membandingkan dengan hasil identifikasi oleh tim pengembang. Metode pembandingan antara hasil identifikasi oleh system dengan hasil identifikasi pengembang dengan mengukur nilai precission, recall dan accuracy dari masing-masing dataset.

***Pengujian smartPortal***

Dataset SmartPortal terdiri dari 17 cerita pengguna dan 52 file kode sumber. Percobaan pada smartPortal dengan masing-masing similarity threshold 0.4, 0.5, 0.6, 0.7 dipilih threshold tersebut karena semakin kecil threshold semakin banyak hasil identifikasi system yang salah, sedangkan semakin tinggi semakin sedikit yang teridentifikasi. Selain similarity threshold ditambahkan pula prosentase jumlah kata yang terdapat pada kode sumber dibandingkan dengan jumlah kata yang terdapat pada cerita pengguna dengan threshold 0%, 5%, 10%, 15%, 20%, 25%, dan 30%. Dipilih prosentase tersebut dengan alasan yang sama dengan similarity threshold.

Berdasarkan pemilihan threshold tersebut maka dihasilkan tabel 14 Tabel rata-rata precission dan recall smartPortal. Dari tabel ini dibentuk grafik precission, recall serta trend grafik precission dan recall. Pada tabel 14 nilai recall tertinggi terdapat pada similarity threshold 0.4 dan 0.5 serta pada kandungan kata 0% dan 5%, hal ini disebabkan terdapat 22 kerunutan yang berhasil teridentifikasi dari 47 yang seharusnya, tetapi system mendeteksi lebih dari 179 kerunutan sehingga nilai presisinya rendah. Nilai presisi tertinggi terletak pada similarity threshold 0.7 dan prosentase kandungan kata 25%, hal ini disebabkan system berhasil mengidentifikasi 9 kerunutan yang tepat dari 47 yang seharusnya ditemukan, dan terdapat 27 kesalahan yang seharusnya bukan merupakan kerunutan.

Gambar 12 menunjukkan kurva interpolasi Precission dan Recall smartPortal, Perpotongan antara precission dan recall terletak pada titik antara prosentase jumlah kata 20% dan 25% pada ambang batas kemiripan 0,7 merupakan titik terbaik dari presisi dan recall, dengan nilai presisi antara 0,288 dengan nilai recall antara 0,556 sampai 0,226.

Tabel 16.
Rata-rata precission recall smartPortal

| No | ST | PJK | ID | | JP | | | | AVG | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | Dev | Sys | TP | FP | FN | TN | Prec | Rec |
| **1** | 0.4 | 0% | 47 | 196 | 22 | 174 | 25 | 663 | 0.12 | 0.51 |
| **2** | 0.4 | 5% | 47 | 185 | 22 | 163 | 25 | 674 | 0.12 | 0.51 |
| **3** | 0.4 | 10% | 47 | 140 | 19 | 121 | 28 | 716 | 0.14 | 0.45 |
| **4** | 0.4 | 15% | 47 | 104 | 17 | 87 | 30 | 750 | 0.17 | 0.42 |
| **5** | 0.4 | 20% | 47 | 80 | 14 | 66 | 33 | 771 | 0.19 | 0.32 |
| **6** | 0.4 | 25% | 47 | 56 | 11 | 45 | 36 | 792 | 0.19 | 0.26 |
| **7** | 0.4 | 30% | 47 | 38 | 9 | 29 | 38 | 808 | 0.25 | 0.23 |
| **8** | 0.5 | 0% | 47 | 187 | 22 | 165 | 25 | 672 | 0.12 | 0.51 |
| **9** | 0.5 | 5% | 47 | 179 | 22 | 157 | 25 | 680 | 0.13 | 0.51 |
| **10** | 0.5 | 10% | 47 | 123 | 18 | 105 | 29 | 732 | 0.16 | 0.44 |
| **11** | 0.5 | 15% | 47 | 90 | 17 | 73 | 30 | 764 | 0.21 | 0.42 |
| **12** | 0.5 | 20% | 47 | 67 | 13 | 54 | 34 | 783 | 0.27 | 0.30 |
| **13** | 0.5 | 25% | 47 | 42 | 10 | 32 | 37 | 805 | 0.26 | 0.24 |
| **14** | 0.5 | 30% | 47 | 28 | 7 | 21 | 40 | 816 | 0.20 | 0.16 |
| **15** | 0.6 | 0% | 47 | 174 | 21 | 153 | 26 | 684 | 0.13 | 0.50 |
| **16** | 0.6 | 5% | 47 | 166 | 21 | 145 | 26 | 692 | 0.13 | 0.50 |
| **17** | 0.6 | 10% | 47 | 117 | 17 | 100 | 30 | 737 | 0.19 | 0.42 |
| **18** | 0.6 | 15% | 47 | 79 | 13 | 66 | 34 | 771 | 0.21 | 0.30 |
| **19** | 0.6 | 20% | 47 | 63 | 12 | 51 | 35 | 786 | 0.26 | 0.28 |
| **20** | 0.6 | 25% | 47 | 37 | 9 | 28 | 38 | 809 | 0.26 | 0.23 |
| **21** | 0.6 | 30% | 47 | 24 | 7 | 17 | 40 | 820 | 0.22 | 0.16 |
| **22** | 0.7 | 0% | 47 | 174 | 21 | 153 | 26 | 684 | 0.13 | 0.50 |
| **23** | 0.7 | 5% | 47 | 166 | 21 | 145 | 26 | 692 | 0.13 | 0.50 |
| **24** | 0.7 | 10% | 47 | 115 | 15 | 100 | 32 | 737 | 0.18 | 0.34 |
| **25** | 0.7 | 15% | 47 | 77 | 12 | 65 | 35 | 772 | 0.20 | 0.29 |
| **26** | 0.7 | 20% | 47 | 61 | 11 | 50 | 36 | 787 | 0.25 | 0.27 |
| **27** | 0.7 | 25% | 47 | 36 | 9 | 27 | 38 | 810 | 0.29 | 0.23 |
| **28** | 0.7 | 30% | 47 | 21 | 7 | 14 | 40 | 823 | 0.25 | 0.16 |

Keterangan

| | | | |
|---|---|---|---|
| ST | Similarity Treshhold | TP | True Positive |
| PJK | Prosentase Jumlah Kata | FP | False Positive |
| ID | Identifikasi | FN | False Negative |
| Dev | Developer | TN | True Negative |
| Sys | System | AVG | Rata-Rata |
| JP | Jumlah Perbandingan | Prec | Precission |
| | | Rec | Recall |

## *Pengujian smartAbsensi*

Dataset ini terdiri dari terdiri dari 47 cerita pengguna dan 161 file kode sumber. Berdasarkan tabel cerita pengguna dan tabel kode sumber, pengembang malakukan identifikasi identifikasi hubungan kerunutan sehingga diidentifikasi terdapat 200 hubungan kerunutan, seperti terlihat pada tabel 14. Tabel rata-rata precission recall smartAbsensi pada kolom ketiga.

Pada tabel tersebut juga dapat dilihat semakin kecil similarity threshold semakin banyak kerunutan yang teridentifikasi oleh system pada ambang batas 0.4 dan prosentase kata 0% system mengidentifikasi sebanyak 2092 kerunutan, dan yang benar sesuai dengan identifikasi pengembang hanya 129 dan yang 1963 sistem salah mengidentifikasi sehingga nilai presisi menjadi kecil. Dengan menambahkan prosentase jumlah kata yang terkandung dalam kode program dapat memperbaiki nilai presisi pada percobaan dapat dilihat pada tabel 14 dengan menambahkan prosentase antara 15% sampai 20% nilai presisi menjadi lebih baik.

Hampir sama dengan smartPortal, pemberian jumlah kandungan kata diatas 20% akan mengurangi jumlah kerunutan yang teridentifikasi, sehingga nilai presisipun menurun. Pada smartAbsensi terdapat sedikit perbedaan karakteristik data yaitu nilai presisi mulai menurun pada saat prosentase jumlah kata sebesar 25% dan mengalami puncaknya pada prosentase 20% dengan nilai presisi sebesar 0.285.

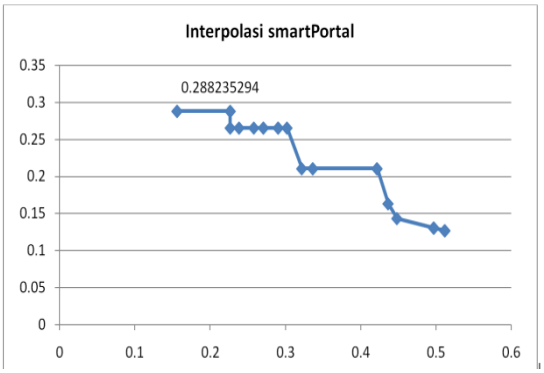
## *Pengujian smartTravel*

Dataset ini terdiri dari 80 cerita pengguna dan 222 file kode sumber. Berdasarkan tabel cerita

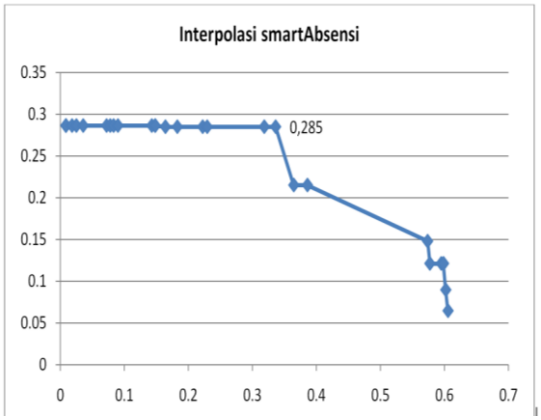
pengguna dan tabel kode sumber, pengembang malakukan identifikasi identifikasi hubungan kerunutan sehingga diidentifikasi terdapat 334 hubungan kerunutan, seperti terlihat pada tabel 15, Tabel hasil evaluasi precision recall *smartTravel* pada kolom ketiga.

Pada tabel tersebut juga dapat dilihat semakin kecil similarity threshold semakin banyak kerunutan yang teridentifikasi oleh system pada ambang batas 0.4 dan prosentase kata 0% system mengidentifikasi sebanyak 5379 kerunutan, dan yang benar sesuai dengan identifikasi pengembang hanya 279 dan terdapat 5100 sistem salah mengidentifikasi sehingga nilai presisi menjadi kecil. Dengan menambahkan prosentase jumlah kata yang terkandung dalam kode program dapat memperbaiki nilai presisi pada percobaan dapat dilihat pada tabel 4.24 dengan menambahkan prosentase antara 15% sampai 20% nilai presisi menjadi lebih baik.

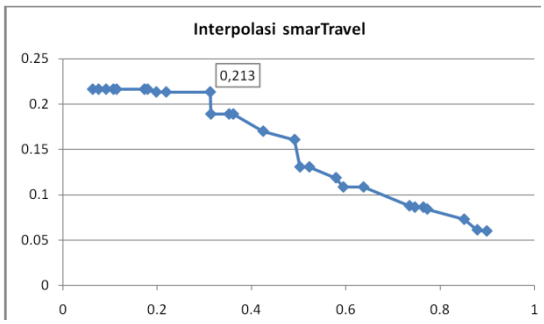
Hampir sama dengan data sebelumnya, pemberian jumlah kandungan kata diatas 25% akan mengurangi jumlah kerunutan yang teridentifikasi, sehingga nilai presisipun menurun. Pada smartTravel nilai precision tertinggi berada pada threshold 0.5 dan jumlah kandungan kata 25% dengan nilai 0,217.




Gambar 12. Kurva interpolasi Precission dan Recall smartPortal.




Gambar 13. Kurva interpolasi Precission dan Recall smartAbsensi




Gambar 14. Kurva interpolasi Precission dan Recall smartTravel

## 6. KESIMPULAN

Beberapa kesimpulan yang dapat ditarik dari hasil pengerjaan penelitian ini adalah sebagai berikut ini.

1. Dalam penelitian ini dikembangkan suatu metode untuk mengidentifikasi hubungan kerunutan antara cerita pengguna dengan kode sumber yang berbahasa Indonesia. Metode yang digunakan adalah menghubungkan kata-kata hasil ekstraksi dari cerita pengguna dan kode sumber menggunakan kemiripan kata dengan algoritma trigram yang terdapat dalam plugin postgreSQL. Hasil dari hubungan keduanya dioptimasi dengan menambahkan ambang batas perbandingan jumlah kata dalam kode sumber dengan jumlah kata dalam cerita pengguna.
2. Optimasi parsing kode sumber menggunakan aturan yang digunakan dalam penamaan kode sumber java atau konvensi penamaan java.
3. Hasil akhir uji coba oleh sistem dibandingkan dengan hasil identifikasi oleh pengembang adalah sebagai berikut:
   - Pada kasus pengujian smartPortal diperoleh nilai presisi 0,288 dengan nilai recall antara 0,556 sampai 0,226.
   - Pada kasus pengujian smartAbsensi diperoleh nilai presisi 0,285 berada pada nilai recall 0,008 sampai 0,336.
   - Pada kasus pengujian smartTravel diperoleh nilai presisi 0,213 berada pada nilai recall 0,006 sampai 0,312.
   - Berdasarkan hasil dari ketiga percobaan tersebut maka nilai presisi tergolong kecil karena berada dibawah 0,5 hal ini disebabkan tidak ditemukannya kemiripan kata antara cerita pengguna dengan kode sumber yang masih menggunakan bahasa Inggris atau singkatan-singkatan.

Tabel 17.
Rata-rata precission recall smartAbsensi

| No | ST | PJK | ID | | JP | | | | AVG | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | Dev | Sys | TP | FP | FN | TN | Prec | Rec |
| 1 | 0.4 | 0% | 334 | 5379 | 279 | 5100 | 55 | 12326 | 0.06 | 0.90 |
| 2 | 0.4 | 5% | 334 | 5135 | 273 | 4862 | 61 | 12564 | 0.06 | 0.88 |
| 3 | 0.4 | 10% | 334 | 2411 | 197 | 2214 | 137 | 15212 | 0.11 | 0.64 |
| 4 | 0.4 | 15% | 334 | 1196 | 157 | 1039 | 177 | 16387 | 0.16 | 0.49 |
| 5 | 0.4 | 20% | 334 | 663 | 114 | 549 | 220 | 16877 | 0.19 | 0.36 |
| 6 | 0.4 | 25% | 334 | 301 | 67 | 234 | 267 | 17192 | 0.20 | 0.22 |
| 7 | 0.4 | 30% | 334 | 203 | 53 | 150 | 281 | 17276 | 0.18 | 0.17 |
| 8 | 0.5 | 0% | 334 | 4705 | 251 | 4454 | 83 | 12972 | 0.07 | 0.85 |
| 9 | 0.5 | 5% | 334 | 1508 | 127 | 1381 | 73 | 5986 | 0.09 | 0.59 |
| 10 | 0.5 | 10% | 334 | 1868 | 170 | 1698 | 164 | 15728 | 0.12 | 0.58 |
| 11 | 0.5 | 15% | 334 | 889 | 130 | 759 | 204 | 16667 | 0.17 | 0.42 |
| 12 | 0.5 | 20% | 334 | 424 | 87 | 337 | 247 | 17089 | 0.21 | 0.31 |
| 13 | 0.5 | 25% | 334 | 182 | 53 | 129 | 281 | 17297 | 0.22 | 0.18 |
| 14 | 0.5 | 30% | 334 | 111 | 33 | 78 | 301 | 17348 | 0.18 | 0.11 |
| 15 | 0.6 | 0% | 334 | 3770 | 232 | 3538 | 102 | 13888 | 0.08 | 0.77 |
| 16 | 0.6 | 5% | 334 | 3583 | 225 | 3358 | 109 | 14068 | 0.09 | 0.75 |
| 17 | 0.6 | 10% | 334 | 1522 | 156 | 1366 | 178 | 16060 | 0.13 | 0.52 |
| 18 | 0.6 | 15% | 334 | 702 | 111 | 591 | 223 | 16835 | 0.16 | 0.35 |
| 19 | 0.6 | 20% | 334 | 310 | 58 | 252 | 276 | 17174 | 0.17 | 0.20 |
| 20 | 0.6 | 25% | 334 | 124 | 33 | 91 | 301 | 17335 | 0.16 | 0.11 |
| 21 | 0.6 | 30% | 334 | 80 | 23 | 57 | 311 | 17369 | 0.13 | 0.08 |
| 22 | 0.7 | 0% | 334 | 3625 | 230 | 3395 | 104 | 14031 | 0.09 | 0.76 |
| 23 | 0.7 | 5% | 334 | 3442 | 222 | 3220 | 112 | 14206 | 0.09 | 0.73 |
| 24 | 0.7 | 10% | 334 | 1434 | 152 | 1282 | 182 | 16144 | 0.13 | 0.50 |
| 25 | 0.7 | 15% | 334 | 642 | 103 | 539 | 231 | 16887 | 0.16 | 0.31 |
| 26 | 0.7 | 20% | 334 | 260 | 53 | 207 | 281 | 17219 | 0.18 | 0.17 |
| 27 | 0.7 | 25% | 334 | 106 | 30 | 76 | 304 | 17350 | 0.17 | 0.09 |
| 28 | 0.7 | 30% | 334 | 59 | 20 | 39 | 314 | 17387 | 0.15 | 0.06 |

Tabel 18.
Rata-rata precission recall smartTravel

| No | ST | PJK | ID | | JP | | | | AVG | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | Dev | Sys | TP | FP | FN | TN | Prec | Rec |
| 1 | 0.4 | 0% | 334 | 5379 | 279 | 5100 | 55 | 12326 | 0.06 | 0.90 |
| 2 | 0.4 | 5% | 334 | 5135 | 273 | 4862 | 61 | 12564 | 0.06 | 0.88 |
| 3 | 0.4 | 10% | 334 | 2411 | 197 | 2214 | 137 | 15212 | 0.11 | 0.64 |
| 4 | 0.4 | 15% | 334 | 1196 | 157 | 1039 | 177 | 16387 | 0.16 | 0.49 |
| 5 | 0.4 | 20% | 334 | 663 | 114 | 549 | 220 | 16877 | 0.19 | 0.36 |
| 6 | 0.4 | 25% | 334 | 301 | 67 | 234 | 267 | 17192 | 0.20 | 0.22 |
| 7 | 0.4 | 30% | 334 | 203 | 53 | 150 | 281 | 17276 | 0.18 | 0.17 |
| 8 | 0.5 | 0% | 334 | 4705 | 251 | 4454 | 83 | 12972 | 0.07 | 0.85 |
| 9 | 0.5 | 5% | 334 | 1508 | 127 | 1381 | 73 | 5986 | 0.09 | 0.59 |
| 10 | 0.5 | 10% | 334 | 1868 | 170 | 1698 | 164 | 15728 | 0.12 | 0.58 |
| 11 | 0.5 | 15% | 334 | 889 | 130 | 759 | 204 | 16667 | 0.17 | 0.42 |
| 12 | 0.5 | 20% | 334 | 424 | 87 | 337 | 247 | 17089 | 0.21 | 0.31 |
| 13 | 0.5 | 25% | 334 | 182 | 53 | 129 | 281 | 17297 | 0.22 | 0.18 |
| 14 | 0.5 | 30% | 334 | 111 | 33 | 78 | 301 | 17348 | 0.18 | 0.11 |
| 15 | 0.6 | 0% | 334 | 3770 | 232 | 3538 | 102 | 13888 | 0.08 | 0.77 |
| 16 | 0.6 | 5% | 334 | 3583 | 225 | 3358 | 109 | 14068 | 0.09 | 0.75 |
| 17 | 0.6 | 10% | 334 | 1522 | 156 | 1366 | 178 | 16060 | 0.13 | 0.52 |
| 18 | 0.6 | 15% | 334 | 702 | 111 | 591 | 223 | 16835 | 0.16 | 0.35 |
| 19 | 0.6 | 20% | 334 | 310 | 58 | 252 | 276 | 17174 | 0.17 | 0.20 |
| 20 | 0.6 | 25% | 334 | 124 | 33 | 91 | 301 | 17335 | 0.16 | 0.11 |
| 21 | 0.6 | 30% | 334 | 80 | 23 | 57 | 311 | 17369 | 0.13 | 0.08 |
| 22 | 0.7 | 0% | 334 | 3625 | 230 | 3395 | 104 | 14031 | 0.09 | 0.76 |
| 23 | 0.7 | 5% | 334 | 3442 | 222 | 3220 | 112 | 14206 | 0.09 | 0.73 |
| 24 | 0.7 | 10% | 334 | 1434 | 152 | 1282 | 182 | 16144 | 0.13 | 0.50 |
| 25 | 0.7 | 15% | 334 | 642 | 103 | 539 | 231 | 16887 | 0.16 | 0.31 |
| 26 | 0.7 | 20% | 334 | 260 | 53 | 207 | 281 | 17219 | 0.18 | 0.17 |
| 27 | 0.7 | 25% | 334 | 106 | 30 | 76 | 304 | 17350 | 0.17 | 0.09 |
| 28 | 0.7 | 30% | 334 | 59 | 20 | 39 | 314 | 17387 | 0.15 | 0.06 |

## DAFTAR PUSTAKA


Buckley, Jim, et al., (2003). "Towards a Taxonomy of Software Change." *Journal of Software Maintenance and Evolution: Research and Practice, Vol. 17, hal. 309-332.*

Dorfman, Merlin dan Flynn, Richard F., (1984). "Arts-An Automated Requirements Traceabillity System." *Journal of Systems and Software, Vol. 4, hal. 63-74.*

Héctor García, Eugenio Santos, Bruno Windels. (2008), "Traceabillity Management Architectures Supporting Total Traceabillity", *Varna, Bulgaria : i.Tech.*

Jelita Asian, Hugh E., Williams S.M.M. Tahaghoghi, (2005), "Stemming Indonesian", *School of Computer Science and Information Technology RMIT University, GPO Box 2476V, Melbourne 3001, Australia.*

Juergen Rilling, René Witte,Yonggang Zhang, (2007), "Automatic Traceabillity Recovery: An Ontological Approach", *Lexington, KY, USA. : s.n., , Vols. March 22-23. ACM ISBN 1-59593-6017/03/07.*

Kannenberg, Andy dan Saiedian, Hossein, (2009). "Why Software Requirements Traceabillity Remains a Challenge." *The Journal of Defense Software Engineering, Juli, Vol. 22, hal. 14-19.*

Kenneth E. Kendall & Julie E. Kendall, (2011). "System Analysis and Design*", Prentice Hall*

Mark Grechanik, Kathryn S. McKinley, Dewayne E. Perry, (2007), "Recovering And Using Use-Case-Diagram-To-Source-Code Traceabillity Links",*, Cavtat near Dubrovnik, Croatia : s.n., 2007, Vols. 3-7. ACM 978-1-59593-811-4/07/0009.*

Mens, Tom dan Demeyer, Serge, (2008), "Software Evolution." *Berlin : Springer.*

P. Lago, H. Muccini, and H. van Vliet, (2009). "A Scoped Approach To Traceabillity Management*", Journal of Systems and Software, 82(1):168 - 182, Special Issue: Software Performance - Modeling and Analysis.*

Rilling, dkk.( 2007), "Automatic Traceabillity Recovery An Ontological Approach" *Dept. of Computer science and SE Montreal, QC H3G1M8, Canada.*

Rochimah, Siti, Wan Kadir, W. M. N. dan Abdullah, Abdul H, (2007). "An Evaluation of Kerunutan Approaches to Support Software Evolution." *International Conference on Software Engineering Advances, 2007.*

Sun Microsystem, (1995-1999) , "9 - Naming Conventions", http://www.oracle.com/technetwork/java/javase/documentation/codeconventions-135099.html

The PostgreSQL Global Development Group, (1996-2016), "PostgreSQL 9.4.8 *Documentation", https://www.postgresql.org/docs/9.4/static/pgtrgm.html*