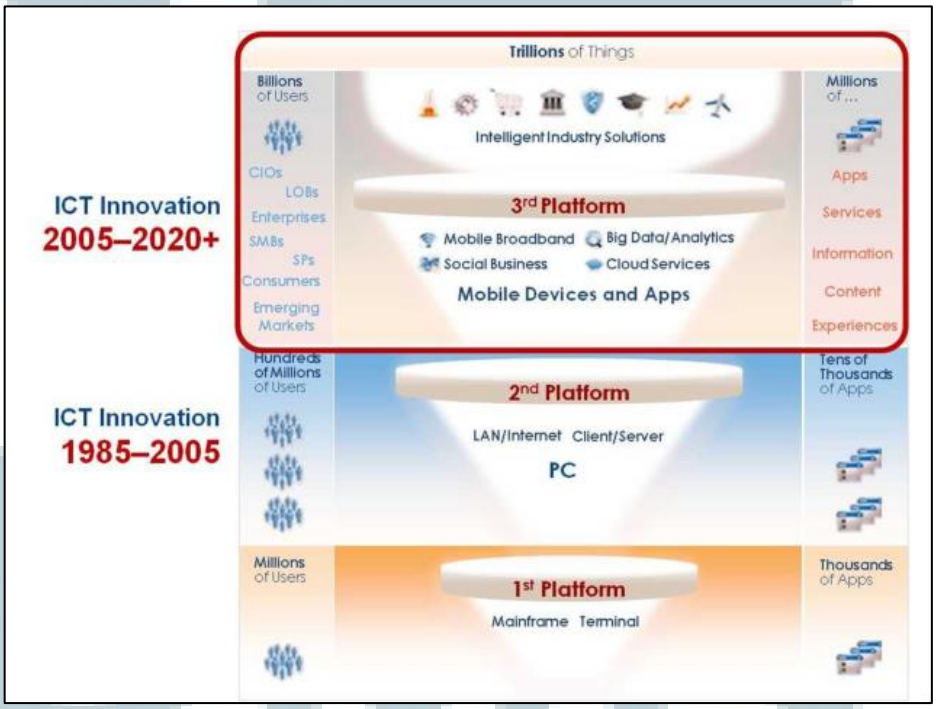
BAB I

PENDAHULUAN

1.1 Latar Belakang

Perkembangan teknologi informasi di dunia saat ini sudah semakin besar. Pengeluaran untuk produk *Information Technology* (IT) diseluruh dunia untuk 2013, diramalkan bisa berkembang mencapai 5,7% (730 miliar dolar Amerika) dari tahun sebelumnya. Dimana 22% dari keseluruhan pengeluaran tersebut berasal dari *3rd Platform technologies* (Gens, 2012) dan diperkirakan pasar akan mulai bertransisi ke arah ini (Reisinger, 2012).



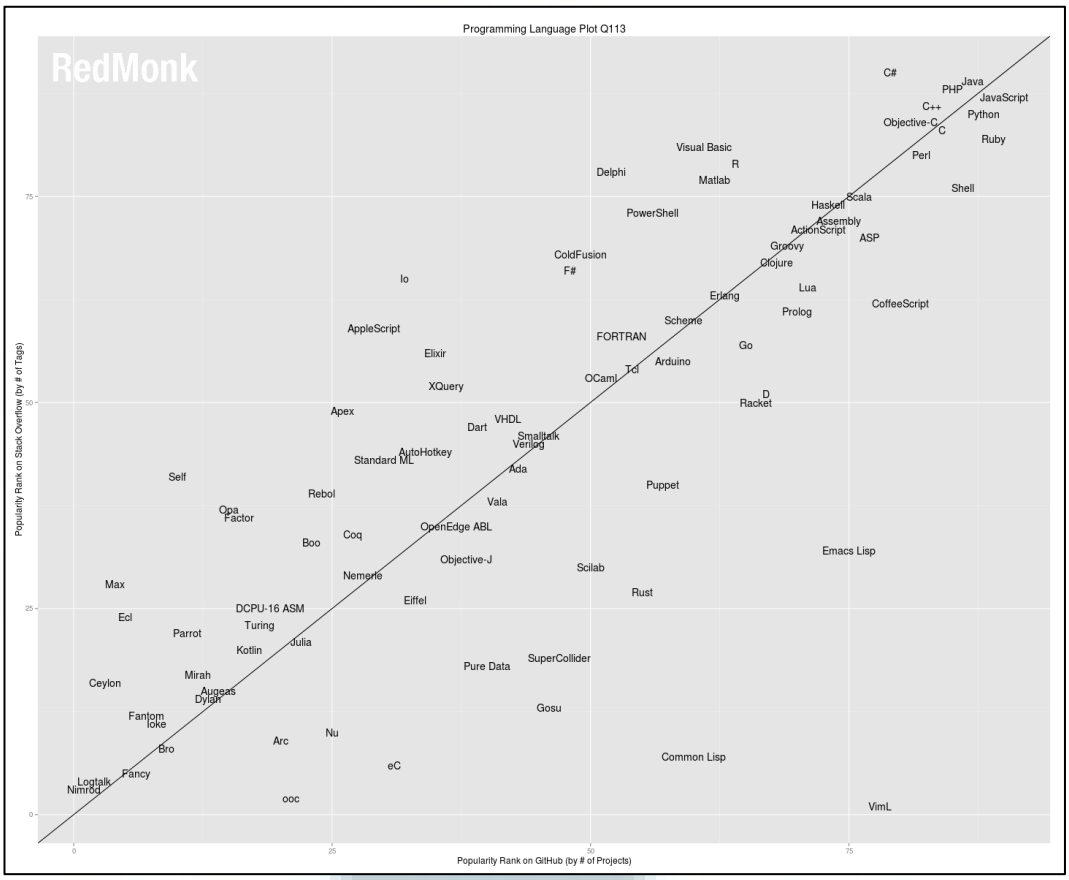
Gambar 1.1 Prediksi perkembangan industri *Information Communication Technology* (ICT) (IDC, 2012).

Hal Ini menandakan bahwa, transaksi penjualan aplikasi (dalam bentuk apapun) akan membanjiri pasar di bidang ICT. Apapun aplikasi yang dibuat, umumnya dihasilkan dari *source code* atau bisa disebut dengan kumpulan kode-kode perintah untuk komputer. Semakin besar suatu program dibuat, tentu saja *source code*-nya juga semakin panjang dan semakin sulit untuk diatur dengan baik.

Terkadang karena sulitnya mengatur *source code* yang panjang, seringkali banyak fungsi-fungsi yang tidak dipakai, tetapi juga ikut masuk kedalam *source code programmer*. Hal ini tentu saja akan mempengaruhi ukuran dan performa dari program aplikasi yang ingin dibuat. Biasa hal ini biasa disebut dengan "*Code Bloat*" (DocForge, 2011). Alangkah baiknya jika fungsi-fungsi yang tidak digunakan ini bisa dihapus, sehingga performa dari aplikasi bisa menjadi lebih maksimal. Tetapi mengingat ukuran *source code* yang biasanya tidak sedikit, hal itu akan sulit jika dikerjakan secara manual.

Dalam dunia pemrograman sekarang, terdapat berbagai macam jenis bahasa pemrograman dan JAVA merupakan salah satu bahasa pemrograman yang populer dibahas dalam forum pemrograman Stack Overflow dan GitHub yang bisa dilihat pada Gambar 1.2. JAVA memiliki beberapa fitur seperti *dynamic binding*, *polymorphism* dan *garbage collection* yang banyak membantu dalam proses pemrograman (Java). Akan tetapi, disisi lain juga menimbulkan beberapa pertanyaan mengenai performa program yang dihasilkan. Hal ini membuat para programmer tidak bisa hanya menunggu perkembangan teknologi *compiler*, mereka juga harus memperhatikan ketepatan dari *source code* yang dibuat.

Sehingga dalam kasus ini analisis program diperlukan untuk membuat performa program menjadi lebih optimal.



Gambar 1.2 Grafik popularitas bahasa pemrograman (O'Grady, 2013).

Untuk melakukan analisis *source code* dari sebuah program, *Call Graph* adalah salah satu alat bantu untuk menganalisis relasi pemanggilan fungsi dalam sebuah program (Bhat & Singh, 2012). *Call Graph* menunjukkan alur pemanggilan fungsi dalam sebuah program dalam bentuk diagram. Dengan begitu *programmer* bisa lebih mudah / cepat dalam membaca alur program buatannya, sehingga bisa melakukan optimalisasi seperti memperkecil ukuran program dengan cara mengeliminasi fungsi / *class* yang tidak terpakai dan lain-lain.

Sejak tahun 1990-an algoritma untuk membuat *Call Graph* sudah mulai diteliti dan menghasilkan berbagai macam algoritma (Tip & Palsberg, 2000). Diantaranya yang paling umum digunakan adalah *Rapid Type Analysis* (RTA) dan 0-CFA. Algoritma RTA dikenal memiliki skalabilitas yang tinggi. Namun, ketepatan algoritma ini dalam meng-generate call graph masih dipertanyakan. Sebaliknya, skalabilitas algoritma 0-CFA yang memiliki ketepatan tinggi, masih diragukan. Pada tahun 2000, Frank Tip dan Jens Palsberg merancang sebuah algoritma yang bisa menyeimbangkan unsur kecepatan dan ketepatan yang didapat dari RTA dan 0-CFA, yaitu Separate Type Analysis (XTA) (Tip & Palsberg, 2000). Sehingga, diantara algoritma-algoritma konstruksi *call graph* yang lain, XTA dinilai cocok dalam kasus penggunaan pada umumnya, karena memiliki keseimbangan antara skalabilitas dan kecepatan.

1.2 Identifikasi Masalah

Berdasarkan latar belakang, dapat disimpulkan bahwa penulisan *source code* oleh *programmer* pada umumnya bisa terdapat kekurangan yang sebenarnya bisa dianalisis dan diperbaiki untuk meningkatkan performa dari program yang dihasilkan.

1.3 Rumusan Masalah

Masalah yang dapat dirumuskan pada penelitian ini adalah bagaimana mengimplementasikan algoritma *Separate Type Analysis* untuk membuat sebuah

aplikasi yang mampu mengkonstruksikan *call graph* dari sebuah *source code* berbahasa pemrograman JAVA.

1.4 Batasan Masalah

Dalam penelitian yang dikerjakan, hasil aplikasi yang dibuat hanya akan menganalisis *source code* berbahasa pemrograman JAVA (JAVA *Project*) yang bisa terbaca (tidak terenkripsi), sudah bisa ter-*compile* dan *call graph* yang dibuat adalah *static call graph*, dengan algoritma *separate type analysis* (XTA).

1.5 Tujuan Penelitian

Tujuan dari penelitian ini yaitu mengimplementasikan algoritma XTA pada sebuah *plug in* IDE Eclipse yang bisa men-*generate* sebuah *static call graph* dari *source code* berbahasa pemrograman JAVA dan dapat menunjukkan relasi antar fungsi, sehingga dapat memberikan manfaat bagi penggunanya.

1.6 Manfaat Penelitian

Manfaat dari penelitian ini adalah untuk membantu para *programmer* bahasa pemrogaman JAVA dalam memberikan kemudahan dalam menganalisis *source code* buatannya, untuk kemudian bisa dibuat lebih efektif, sehingga aplikasi yang akan dihasilkan pun akan lebih baik performanya.

1.7 Sistematika Penulisan

Sistematika penulisan skripsi ini dijelaskan sebagai berikut

1. Bab I : Pendahuluan

   Bab ini berisi gambaran umum dari penelitian ini. Bab ini berisi latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat dan sistematika penelitian.

2. Bab II : Tinjauan Pustaka

   Bab ini berisikan istilah-istilah yang menjadi dasar teori yang digunakan dalam penelitian ini. Istilah yang dijelaskan dalam penelitian ini adalah Eclipse, bahasa pemrograman JAVA, *call graph*, RA, XTA, abstract syntax tree (AST), metode pengembangan *prototyping* dan Ecplise *plug in*.

3. Bab III : Analisis dan Perancangan Aplikasi

   Bab ini berisi spesifikasi umum dan perancangan aplikasi (*plug in*). Bab ini berisikan gambaran metodologi penelitian, rancangan cara kerja dan desain rancangan antarmuka aplikasi.

4. Bab IV : Implementasi dan Uji Coba

   Bab ini berisi penjelasan mengenai implementasi dan hasil uji coba aplikasi (*plug in*). Bab ini akan membahas bagaimana perancangan yang telah dilakukan, diterapkan secara nyata. *Output*, dari aplikasi (*plug in*) ini akan diuji dan diamati untuk menentukan apakah sesuai dengan tujuan penelitian.

5. Bab V : Simpulan dan Saran

   Bab ini berisi simpulan dan saran dari penelitian ini. Simpulan berbicara tentang berhasil atau tidaknya penelitian dalam menyelesaikan permasalahan yang ada dan kesesuaian tujuan dengan hasil penelitian.

Saran akan berbicara tentang kekurangan dari penelitian, yang mungkin bisa dikembangkan lagi dalam penelitian selanjutnya.