O Problema de Restauração de Cadastros

Alunos: Caio Dias Valentim
Paulo Ivson Netto Santos
Orientador: Eduardo Sany Laber

1. Introdução

O cadastro de endereços de clientes é um ativo muito importante da maioria das grandes empresas da atualidade, especialmente no setor de serviços. Tal cadastro permite a empresa se comunicar com seus clientes e facilita o comércio ao permitir o envio de produtos e cobranças.

Infelizmente, muitos destes cadastros contêm uma quantidade grande erros. Estes ocorrem por diversos motivos: alguns são provenientes da digitação, outros provêm de uma ortografia incorreta. Freqüentemente, o próprio cliente desconhece parte de seu endereço, escrevendo o bairro ou o CEP errado, por exemplo.

Tais erros causam prejuízos grandes a empresas, devido ao custo associado ao envio de correspondências para endereços equivocados.

Como fruto dos esforços anteriores, foi desenvolvido um software capaz de corrigir automaticamente uma base de endereços incorretos de forma eficaz e eficiente. Os resultados obtidos validaram os algoritmos implementados e comprovaram a estratégia de solução desenvolvida.

Dando continuidade aos estudos efetuados, foram observados problemas e limitações no software desenvolvido. Estes por sua vez obstruíam o estudo de novos algoritmos, já que prejudicavam a implementação e conseqüente experimentação de novas propostas.

Objetivos

- Verificar a correção do software desenvolvido anteriormente
- Elaborar uma nova proposta para uma plataforma de desenvolvimento de algoritmos que visam atacar o problema de restauração de cadastros
- Implementar as soluções propostas e validar seus resultados com base nos resultados já comprovados do software anterior

2. Uma Metodologia para o Problema de Restauração de Cadastros

O processo de correção de um endereço baseia-se na premissa de que todo endereço errado - que não existe na realidade - foi escrito com o intuito de representar uma localidade geográfica correta. Dessa forma, a metodologia de solução do problema consiste em identificar, em uma base de endereços corretos - que realmente existem - os endereços que melhor correspondem ao endereço que se pretende corrigir.

Portanto, a restauração de cadastros de endereços ocorre através de um processo de busca e comparação, onde objetiva-se identificar qual o melhor ou os melhores candidatos à correção de um dado endereço incorreto.

O software desenvolvido anteriormente implementava este paradigma através de estruturas de busca e indexação de bases de dados, além de algoritmos auxiliares de recuperação de informação textual e comparação aproximada de frases e palavras.

Seus resultados provaram que esta abordagem era capaz de corrigir com sucesso a grande maioria dos endereços incorretos recebidos. O processo de correção para uma base de quase 20.000 endereços incorretos produzia resultados de confiança mais do que satisfatória para mais de 80% dos endereços, consumindo apenas cerca de 3 minutos.

Identificação do Problema

No início do segundo ano de desenvolvimento do programa, pequenos ajustes eram efetuados para manter o sistema funcionando corretamente. Algumas novas idéias começavam a ser implementadas para teste e validação.

Contudo, esta necessidade de implementar e testar novos algoritmos apontou para um problema que ainda não havia sido constatado: a extensibilidade do programa. Em outras palavras, a maneira com que o software fora construído estava dificultando a implementação de novas técnicas resultantes de nossos estudos.

Estava sendo observado um fenômeno comumente conhecido na Engenharia de Software por *code bloat*. No mesmo, o código de um programa se torna exponencialmente difícil de manter e de se adicionar novas funcionalidades. Vários fatores contribuem para isto, dentre os principais: a linguagem de programação utilizada, o paradigma de programação adotado, a coordenação entre os programadores, a estruturação do programa, a preocupação com o desacoplamento entre seus módulos constituintes, dentre outros.

Tendo em vista o objetivo principal de estudo e pesquisa do projeto, chegou-se à conclusão de que seria necessário uma reestruturação e reimplementação do software. Desse modo, seria possível dar continuidade a futuros estudos sem os empecilhos advindos das dificuldades mencionadas. Iniciou-se, então, um processo de *refactoring* [5].

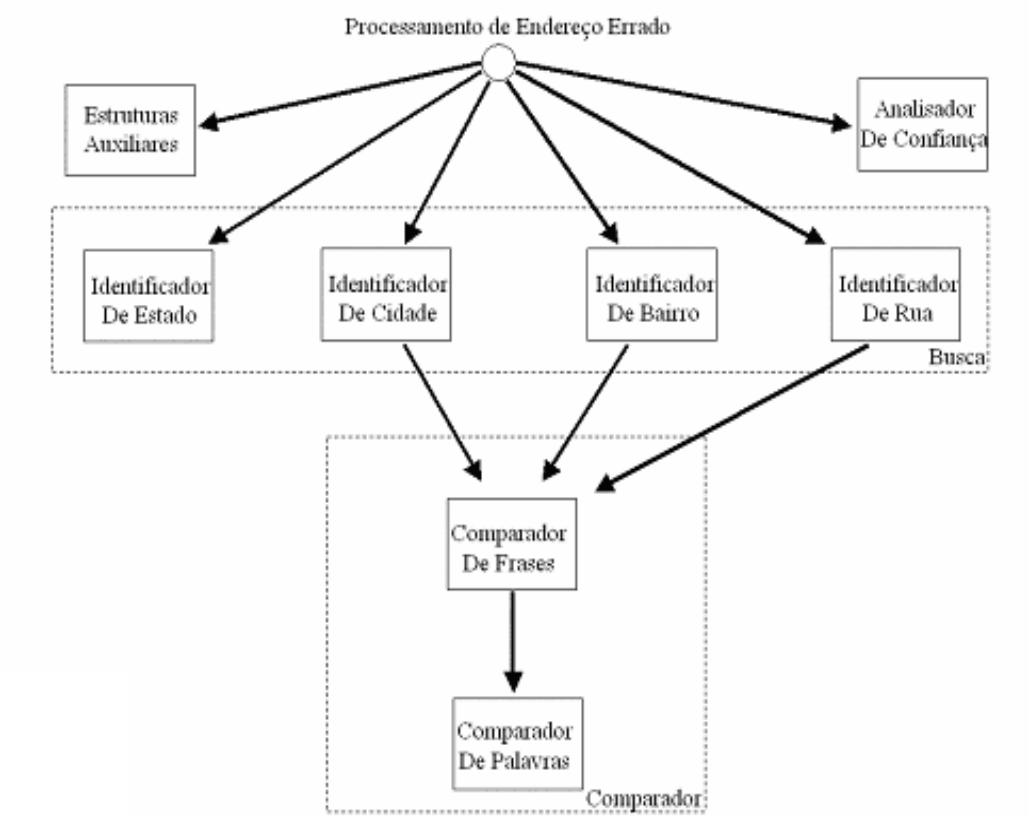
Análise do Software Desenvolvido

Inicialmente, foram efetuadas análises dos resultados obtidos até o momento, identificando tanto seus pontos fortes quanto lugares onde pudessem ser efetuadas melhorias. Foram efetuados testes extensivos do programa, verificando se seus resultados correspondiam com os esperados. As análises obtidas a partir dos testes foram compiladas para posterior referência, já que serviriam como base de validação para resultados futuros.

Em paralelo, foram analisados os principais algoritmos implementados no programa com o mesmo intuito de identificar suas vantagens e limitações. Juntamente com a engenharia reversa do sistema e de suas estruturas, foi elaborada uma documentação do software desenvolvido.

O software fora desenvolvido em C, em uma estrutura modular que visava definir escopos de soluções. Infelizmente esta linguagem contribuía para uma quantidade considerável de vazamentos de memória detectados através de ferramentas especializadas, além de uma dificuldade de isolamento de diferentes estruturas e funcionalidades ao longo do código.

A seguir pode-se observar um diagrama simplificado do programa desenvolvido com base na metodologia proposta.



Após a etapa de testes, ficou constatado que os principais problemas encontrados eram referentes às estruturas de busca e armazenamento utilizadas. Além disso, a falta de encapsulamento apropriado entre os módulos tornava-os dependentes entre si e difíceis de manter. Uma pequena alteração em uma parte do programa causava um impacto significativo em outra parte, tornando-a muitas vezes inoperante.

Concluiu-se que a estrutura do programa, embora eficaz para o problema proposto, estava excessivamente limitada e específica à solução adotada, deixando pouco espaço para a adição de novas funcionalidades e para o teste de novas propostas que vinham sendo estudadas.

Melhorando o Processo de Desenvolvimento

O primeiro passo para a correta reestruturação do programa foi, na verdade, a reestruturação do processo de desenvolvimento do grupo de pesquisa. Após estudos, decidiu-se adotar uma metodologia similar ao processo de desenvolvimento de software conhecido por *XP*, ou *Extreme Programming* [2].

Seus enfoques principais envolvem um maior envolvimento do cliente e uma maior coordenação entre a equipe de desenvolvimento. Além disso, o método defende 5 valores fundamentais: *communication*, *simplicity*, *feedback*, *courage*, *respect*. Estes, por sua vez,

determinam uma série de procedimentos a serem seguidos de forma a melhorar, como um todo, o processo de desenvolvimento de um software.

A comunicação de requisitos foi reforçada através de diversas reuniões com todos os integrantes do projeto. Com base na documentação gerada, foram discutidos os pontos fortes e fracos observados no software e atentou-se para as opiniões e sugestões de cada um a respeito de cada observação efetuada. Além disso, tais discussões deram origem a novas propostas de solução dos problemas encontrados, além da extensão de funcionalidades do programa.

Atentando à necessidade de coordenação entre os desenvolvedores do software, foram adotadas metodologias de programação que visavam garantir o correto funcionamento de diferentes partes integrantes do sistema, que eram desenvolvidas concomitantemente. Uma delas foi a utilização de um software para controle de versão do código. Adicionalmente, foi dada atenção para testes unitários e contínuos de cada componente e do sistema como um todo, de modo a verificar e validar todas e quaisquer alterações efetuadas no software.

O contato freqüente com o orientador da pesquisa foi importante para a constante revisão dos requisitos do sistema e para a demonstração de resultados que seriam responsáveis pela validação das alterações e pela identificação de falhas a serem corrigidas.

Procurou-se adotar um paradigma de *pair programming* aliado à adoção de padrões de código comuns, de modo a melhorar a coordenação entre os desenvolvedores e melhorar a usabilidade e interoperabilidade de módulos desenvolvidos separadamente.

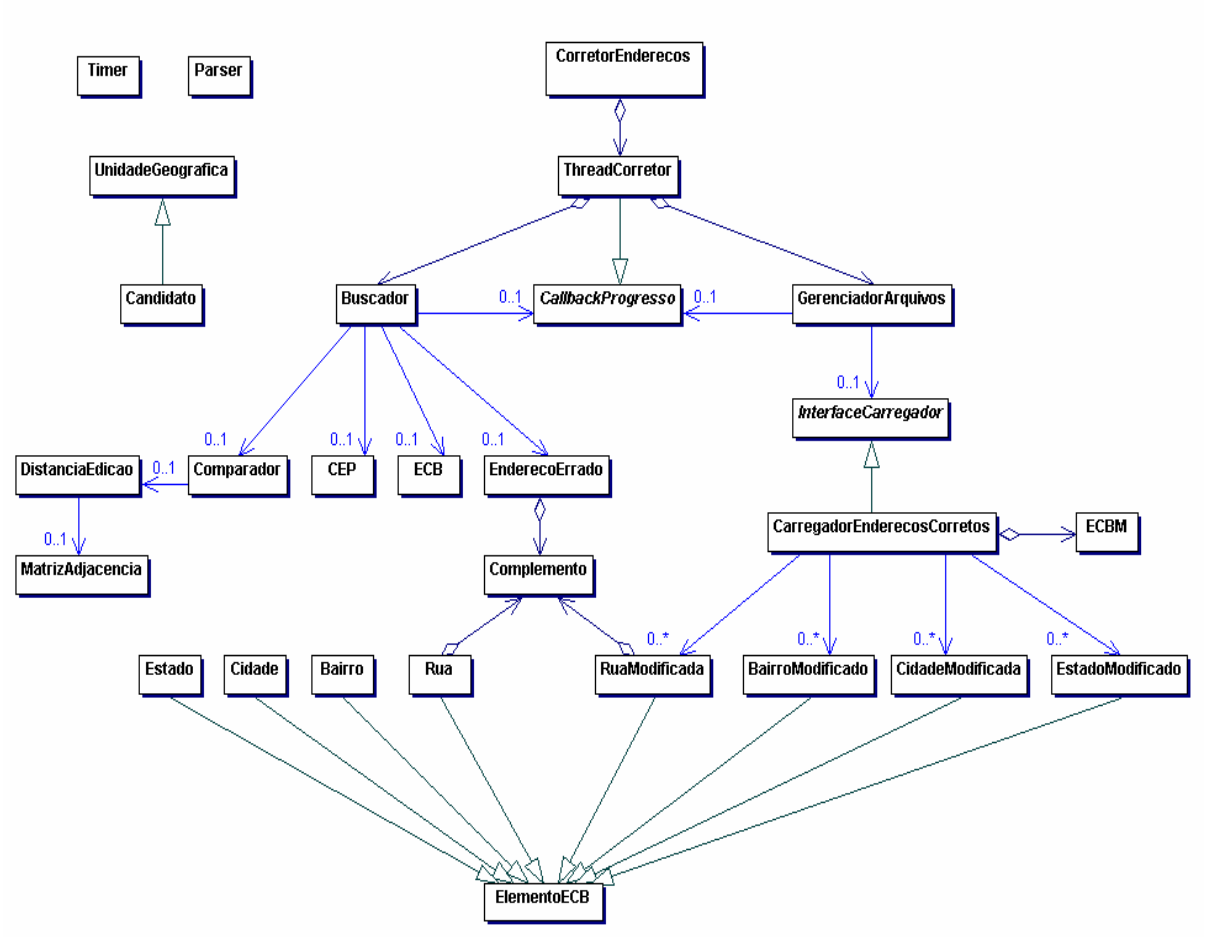
Uma Plataforma de Desenvolvimento de Algoritmos

Voltando nossa atenção para a necessidade de suportar extensões do programa para futuras pesquisas, decidiu-se então adotar uma linguagem de programação orientada a objetos, onde o encapsulamento seria reforçado pelos próprios padrões de código a serem adotados.

Estes fatores levaram à escolha do C++ como a linguagem a ser utilizada na nova versão do software [1]. Outros fatores incluem maximizar o reuso do código anterior – escrito em C, preservar seu desempenho e tempo de processamento, além de reestruturar o programa para suportar futuras extensões.

Durante o ciclo de reuniões, foram elaborados pseudo-códigos das partes principais do programa. Estes eram revistos e aprovados por todos e incluíam os algoritmos de busca e recuperação de informação textual e os algoritmos de comparação de frases e palavras.

De posse destes pseudo-códigos, juntamente com as conclusões discutidas nas reuniões, foi possível elaborar uma nova proposta para o programa de correção automática de endereços, como pode ser vista na página seguinte.



Com esta nova estrutura, foi possível isolar diferentes funcionalidades do programa, bem como encapsular suas estruturas para melhor manutenção e extensão futura.

Adicionalmente, cada classe pôde ser testada de forma isolada durante o processo de desenvolvimento, garantindo assim sua correta funcionalidade antes de ser incorporada ao resto do sistema. As estruturas procuraram seguir modelos e padrões para melhor utilização, compreensão e manutenção, como descritos em [3], [4] e [6]. O resultado final foi uma plataforma de desenvolvimento de algoritmos capaz de ser adaptada e estendida com maior facilidade que a versão anterior do programa.

De posse desta segunda versão, iniciou-se o processo de verificação e validação de seus resultados. Até o momento, foi possível comprovar seus algoritmos de comparação de palavras e frases, bem como sua estruturação para indexação e recuperação de informação.

3. Comparação com o sistema anterior

Como dito, um dos principais estágios do trabalho desenvolvido ao longo do ano foi o de revisão dos códigos antigos. Neste estágio, houve o cuidado de analisar cada módulo do software antigo e identificar seus pontos fortes e suas falhas quanto a sua estrutura lógica. Diversas reuniões foram realizadas para discutir o antigo código e gerar novos pseudo-códigos mais eficientes e organizados. Este fato, teve efeito direto sobre a eficiência e eficácia da nova ferramenta.

Nesta seção, a análise de eficiência foi dividida em três grandes áreas: Consumo de memória, Tempo para Correção de endereços e Qualidade dos resultados.

Todos os testes foram realizados em um sistema Microsoft Windows XP Professional - Versão 2002 - Service Pack 2, com processador Intel Pentium 4 - 3.00GHz e com 480Mb de RAM.

Consumo de Memória

Em termos de memória, a principal característica do novo software é sua estabilidade. Bem entendido, ao longo da execução o programa não aumenta significativamente o total de memória consumida. Este problema foi evidenciado nas primeiras versões do antigo software, tendo sido foco de atenção e correções ao longo de meses. Na versão de comparação utilizada o problema já havia sido resolvido através da substituição e reestruturação de boa parte das estruturas de armazenamento previamente utilizadas.

Na tabela a seguir, comparamos o total de memória principal consumida pelos programas ao iniciar a correção e ao fim de uma correção de 10.000 endereços. De fato, a memória consumida pela nova ferramenta é maior que a consumida pela antiga. Mas, isso é justificável uma vez que na nova versão, além da base de busca, a base dos endereços errados também está carregada na memória principal.

Programa mais a base de busca carregada	**Antigo Projeto**	**Novo Projeto**
Antes a correção	117.124 Kb	159.508 Kb
Após a correção	122.444 Kb	165.475 Kb

Tempo para correção de endereços

Outro importante fator é o tempo médio necessário para correção de um endereço, que não somente se encontra na mesma ordem de grandeza da versão anterior do programa, mas se revelou ainda menor do que o necessário anteriormente. Mesmo o atual software sendo feito em uma linguagem de programação por essência menos eficiente, quanto a tempo de processamento, que a linguagem original.

Fatores que justificam tal desempenho são as melhores técnicas de programação adotadas, o uso de estruturas mais inteligentes para auxiliar os processos de busca, a revisão e melhoria dos algoritmos utilizados, dentre outros.

Na tabela a seguir analisamos os tempos médios de correção para lotes de K endereços em ambos os softwares.

Tamanho da Entrada	**Projeto Antigo**	**Novo Projeto**
K = 100	3 seg	0.800762 seg
K = 1.000	22 seg	9.3791 seg
K = 10.000	184 seg	89.8421 seg

Qualidade dos resultados

Nenhum dos dados acima sobre eficiência de memória e de tempo para processamento seriam relevantes se o novo software não cumprisse sua função básica de apresentar possíveis correções para um endereço errado. Alem disso, tais possíveis correções devem ter qualidade, ou seja, devem ter, a olhos humanos, uma proximidade com o endereço errado em questão.

Para garantir tal qualidade, estão sendo efetuados testes para comparar seus resultados para a correção não somente de um endereço, mas da base original com dezenas de milhares de endereços incorretos. Atualmente, estes testes evidenciam que os novos algoritmos de busca funcionam corretamente, bem como o restante do processo de correção de um endereço.

O exemplo abaixo mostra a correção de alguns endereços errados pelo novo software.

Exemplo 1:

Endereços	**Cep**	**Tipo-Logradouro**	**Logradouro**	**Bairro**	**Cidade**	**Estado**
Errado	20540280	R	BORDA DO CMPO	GRAJAU	RIO DE JANEIRO	RJ
Correção 1	20561200	RUA	BORDA DO MATO	GRAJAU	RIO DE JANEIRO	RJ
Errado	22040040	R	RAIMUNDO CORREA	COPACABNA	RIO DO OESTE	RJ
Correção 1	22040040	RUA	RAIMUNDO CORREIA	COPACABANA	RIO DE JANEIRO	RJ
Errado	21645180	RUA	DEOCLECIANO	AMCHIETA	RIO DO OESTE	RJ
Correção 1	21645180	RUA	DEOCLECIANO RAMOS	ANCHIETA	RIO DE JANEIRO	RJ
Errado	21770190	RUA	LEOCADIO	REALENGU	RIO DE JANEIRO	RJ
Correção 1	21770190	RUA	LEOCADIA	REALENGO	RIO DE JANEIRO	RJ
Errado	25223360	AVENIDA	MARQUES DE BAEPENDI	JARDIM PRIMAVERA	DUQUE DE CAXIAS	RJ
Correção 1	25223360	AVENIDA	MARQUES DE BAEPENDI	PARQUE CHUNO	DUQUE DE CAXIAS	RJ
Correção 2	25223365	AVENIDA	MARQUES DE BAEPENDI	PARQUE MODERNO	DUQUE DE CAXIAS	RJ

Errado	20510060	RUA	URUGUAI	TIJUCA	RIO DE JANEIRO	RJ
Correção 1	20510060	RUA	URUGUAI	ANDARAI	RIO DE JANEIRO	RJ

Errado	20010000	R	DA LIBERDADE	CAJU	RIO DE JANEIRO	RJ
Correção 1	20910070	RUA	DA LIBERDADE	SAO CRISTOVAO	RIO DE JANEIRO	RJ

4. Trabalhos futuros

Através do software desenvolvido podemos obter tanto uma única possível correção para determinado endereço como uma lista de possíveis correções (quatto endereço do Exemplo 1). No segundo caso é necessário que existam métricas para avaliar a qualidade individual de cada possível correção apresentada. Este problema era tratado no software antigo através de classificação empírica e sem muitos cuidados quanto à qualidade.

Neste contexto, aplicaremos e avaliaremos métodos de *aprendizado por máquina* [7] para classificar a qualidade de uma determinada correção. Tais métodos utilizarão uma base de endereços corrigidos e suas respectivas classificações atribuídas manualmente. A partir desta base, os algoritmos poderão "aprender" o que determina uma boa correção e assim estabelecer um modelo de classificação.

5. Conclusões

A experiência de desenvolvimento de um software de médio porte trouxe consigo inúmeras lições. Dentre elas, foi observada a importância da coordenação entre os programadores envolvidos, de sua disciplina de programação, da seleção da linguagem de programação adequada às suas necessidades, bem como a preocupação com a extensibilidade e manutenibilidade do software desenvolvido. Afinal, o mesmo servirá como plataforma de desenvolvimento e pesquisa futuros não somente dos pesquisadores atuais mas de seus sucessores.

Através da documentação obtida a partir da análise de resultados anteriores e da engenharia reversa do software que havia sido desenvolvido, foi-se capaz de identificar suas principais vantagens e limitações. O resultado desta análise serviu como base para o desenvolvimento de um segundo protótipo do programa de correção automática de endereços.

Uma revisão na metodologia de desenvolvimento utilizada, bem como a adoção de novos paradigmas de programação, atentando para o maior envolvimento do cliente e uma maior coordenação entre os desenvolvedores, foram os grandes responsáveis por boa parte das melhorias conseguidas na nova versão do sistema.

Este, por sua vez, visa não somente solucionar o problema de restauração de cadastros, mas fazê-lo de forma estruturada e, assim, garantir seu uso como plataforma para estudos futuros. Atualmente em fase final de testes para validação de seus resultados, o novo software comprova a cada dia sua evolução sob o ponto de vista de um projeto de pesquisa.

O paradigma de correção automática de endereços abre novas portas para pesquisa futura na área de aprendizado por máquina.

Por fim, pode-se concluir que a maneira como foi desenvolvida esta nova ferramenta a torna importante aliada no estudo e pesquisa não somente de novas soluções para o problema de restauração de cadastros, mas também para os paradigmas a ele associados. Com a verificação final do novo software, poderão ser incorporadas novas melhorias e propostas futuras que venham a ser estudadas com maior eficiência e facilidade.

6. Referências

1 - STROUSTRUP, Bjarne**: The C++ Programming Language (Special Edition)**. Addison Wesley, 2000

2 - BECK, Kent; ANDRES, Cynthia. **Extreme Programming Explained : Embrace Change (2nd Edition)**. Addison Wesley, 2004

3 - MEYERS, Scott**. Effective C++ (3rd Edition)**. Addison Wesley, 2005

4 - MEYERS, Scott**. More Effective C++**. Addison Wesley, 2005

5 - FOWLER, Martin; BECK, Kent; BRANT, John; OPDYKE, William; ROBERTS, Don. **Refactoring: Improving the Design of Existing Code.** Addison Wesley, 1999

6 - GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. **Design Patterns: Elements of Reusable Object-Oriented Software.** Addison Wesley, 1995

7 – MITCHELL, Tom; HILL, McGraw. **Machine Learning**. 1997