# Reestruturação de Software Dirigida por Conectividade para Redução de Custo de Manutenção

Kecia Aline M. Ferreira [1]
Mariza A. S. Bigonha [1]
Roberto S. Bigonha [1]

**Resumo:** Grande parte do custo de software é dedicado à manutenção. Nos últimos anos, tem havido muito interesse pela definição de instrumentos de estimativa de custo e predição de esforço para a manutenção de software. Nossa tese é que a conectividade é o fator principal no custo de manutenção. Apresentamos neste artigo o Modelo de Avaliação de Conectividade em Sistemas Orientados por Objetos (MACSOO), que é um modelo de reestruturação de software baseado em conectividade cujo objetivo é a redução do custo de manutenção. Relatamos experimentos que exemplificam o uso do modelo proposto e evidenciam a relação entre conectividade e manutenibilidade.

**Abstract:** Most of the software cost is due to maintenance. In the last years, there has been a great deal of interest in developing cost estimation and effort prediction instruments for software maintenance. This work proposes that module connectivity is a key factor to predict maintenance cost and uses this thesis as the basis to develop a Connectivity Evaluation Model in OO Systems (MACSOO), which is a refactoring model based on connectivity whose aim is to minimize maintenance cost. We describe experiments whose results provide an example of the model application and expose the correlation between connectivity and maintainability.

## 1 Introdução

Qualidade e custo são duas das mais importantes variáveis no processo de desenvolvimento de software. O custo de um software define sua viabilidade [29]. A manutenção é a fase que mais demanda esforço no ciclo de vida de um sistema. Meyer [23] destaca que a manutenção é penalizada pela dificuldade de realizar alterações no software e afirma que 70% do custo total de um sistema se refere a custo de manutenção. Para Pfleeger [30], este percentual ultrapassa 80%. Esses dados mostram que reduzir os custos da manutenção de software é imperativo, e que tal redução pode ser obtida principalmente quando for possível realizar alterações no software de forma mais fácil. A modularidade é a chave para esta questão [25, 28]. Um dos benefícios da modularidade é a extensibilidade do software, que torna

[1]Universidade Federal de Minas Gerais - ICEx - DCC
Caixa Postal 702 – Belo Horizonte – MG – Brazil
{kecia,mariza,bigonha@dcc.ufmg.br}

possível alterar um módulo sem afetar os demais ou afetando um número pequeno deles. Um sistema com esta característica possui alto grau de *estabilidade*, que é a capacidade de um software de se manter inalterado diante de uma alteração em seu ambiente [13]. Segundo Meyers [25], a modularidade é determinada basicamente por duas práticas fundamentais: a minimização do acoplamento, o grau de relacionamento entre módulos, e a maximização da coesão, o grau de relacionamento entre elementos de um mesmo módulo.

A busca por instrumentos que aumentem a manutenibilidade de software, bem como aqueles que apóiem a predição de esforço de manutenção, tem sido motivação para vários estudos relatados na literatura, tais como [5, 15, 20, 26, 32]. Defendemos a tese de que a conectividade, o grau de intercomunicação entre os módulos de um sistema, sobrepuja os demais fatores na estabilidade do sistema e na sua manutenção. Para saber se a conectividade em um determinado software é satisfatória, é preciso conhecê-la e, para isso, faz-se necessário medi-la. Se constatado que ela não é adequada, deve-se analisar os fatores adjacentes que contribuem para tal situação. Para tal é necessário: identificar os fatores que têm impacto na conectividade de um software; identificar uma métrica que permita a avaliação da conectividade de um software; e identificar as métricas que permitam que os fatores determinantes da conectividade sejam avaliados.

Neste trabalho, apresentamos o Modelo de Avaliação de Conectividade em Sistemas OO (MACSOO), um método de reestruturação para obtenção de software com baixo grau de conectividade. Para viabilizar a aplicação do modelo, desenvolvemos a ferramenta *Connecta*, que nos permitiu realizar um conjunto de experimentos. Destes experimentos relatamos dois neste artigo, cujos objetivos são exemplificar o uso do modelo proposto e verificar a relação entre conectividade e estabilidade.

Este artigo está estruturado da seguinte maneira: Seção 2 aborda nossa tese da conectividade como fator preponderante no custo de manutenção de software; Seção 3 descreve MACSOO; Seção 4 relata os experimentos realizados; Seção 5 relata alguns trabalhos relacionados; Seção 6 apresenta as conclusões do trabalho.

## 2 A Conectividade como Fator Preponderante no Custo de Manutenção

Uma situação ideal na produção de um software seria aquela em que o projetista ou o desenvolvedor pudesse realizar uma alteração em determinada parte do software e tal alteração não afetasse as demais partes deste software. Entretanto, ao contrário desta situação ideal, é comum ocorrer o que Martin [22] chama de apodrecimento de projeto (*rotting design*), quando o software se transforma em um conjunto de código de difícil manutenção. Martin identifica a rigidez como um dos sintomas principais deste processo. Em um software de estrutura rígida, uma alteração em um módulo causa uma cascata de outras alterações nos demais. De acordo com Martin[22], a rigidez de um software determina a sua degrada-

ção e se estabelece devido à existência de alto grau de interdependência entre os módulos do software.

Gilb [13] denomina a característica de um sistema que se mantém relativamente inalterado diante de uma alteração em seu ambiente como *estabilidade*. A estabilidade é o fator que indica o *impacto de uma manutenção* em um sistema. Conhecer a estabilidade de um sistema é um recurso poderoso no processo de software, pois isso representa um forte dado para a predição do esforço real necessário para a realização de alterações em sistemas. O âmbito do problema investigado neste trabalho diz respeito à identificação de um meio de avaliação da estabilidade de sistemas, em particular aqueles desenvolvidos no paradigma OO.

Conforme Meyer [23] aponta, o caminho para se obter software constituído por módulos tão independentes entre si quanto possível é tornar a comunicação entre eles disciplinada. Para alcançar isso, ele define, dentre outras regras, a regra de *poucas interfaces*, a qual determina que todo módulo deve se comunicar com o menor número possível de outros módulos. Com isso, uma alteração em determinado módulo propaga-se para poucos módulos. Um software cuja estrutura é planejada segundo essa regra tem estrutura mais flexível e possui alta estabilidade. Staa [33] diz que os módulos de um sistema comunicam-se entre si por meio de *conectores*, que são vistos como canais por meio dos quais os módulos trocam dados ou sinais de controle. São exemplos de conectores: variáveis globais, membros de dados e métodos públicos de uma classe. A maneira pela qual módulos estão conectados define o grau do acoplamento entre eles. O acoplamento é uma medida importante pois representa o grau da dependência entre módulos. Por exemplo, dois módulos que se comunicam por meio de acesso a dados comuns têm um grau dependência maior entre si do que dois módulos que se comunicam por meio de métodos públicos.

Definimos, aqui, o conceito de *conectividade* como o grau de intercomunicação entre os módulos de um sistema. Defendemos a tese de que a conectividade é o fator que sobrepuja os demais na estabilidade do sistema e na sua manutenção. A conectividade é o aspecto principal a ser analisado para obter a resposta de questões importantes no processo de software, como: dificuldade de manter o sistema e amplitude do impacto de determinada modificação no sistema.

Myers [25] modela a importância da conectividade para a estabilidade de um sistema por meio de um conjunto de 100 lâmpadas, no qual uma lâmpada representa um módulo, e uma ligação entre duas lâmpadas representa a existência de comunicação entre dois módulos. Uma alteração em um módulo corresponde a ligar a lâmpada que representa o módulo. No esquema de conexões das lâmpadas, a probabilidade de uma lâmpada passar do estado de ligada para desligada no próximo segundo é de 50%; se está desligada, a probabilidade de ser ligada no próximo segundo é de 50% se pelo menos outra lâmpada que esteja a ela conectada for ligada; se uma lâmpada estiver desligada, permanecerá assim enquanto todas as lâmpadas conectadas a ela estiverem desligadas. Diz-se que o circuito atingiu equilíbrio quando todas

as lâmpadas estiverem desligadas. Nesse Modelo das Lâmpadas, Myers analisa três situações possíveis de arranjo das conexões: (1) quando não há conexões entre as lâmpadas o tempo de equilíbrio do circuito é de 7 segundos; (2) quando o circuito é constituído por agrupamentos de 10 lâmpadas, sendo que os agrupamentos são independentes entre si e cada um deles é totalmente conectado, o tempo de equilíbrio do circuito é de 20 minutos; (3) quando o circuito é totalmente conectado, o tempo para atingir o equilíbrio do circuito é de $10^{22}$ anos. As situações mostradas ilustram quão importante a conectividade é para a estabilidade e manutenção de sistemas. O efeito de uma alteração em um módulo de um sistema com alto grau de conectividade é explosivo, o que torna a sua manutenção um problema intratável.

O Modelo das Lâmpadas é a inspiração para a nossa tese de que a conectividade é o indicador primordial para a avaliação da estabilidade de um sistema e, conseqüentemente, é fator determinante para a facilidade de manutenção de sistemas. O acoplamento também é um indicador importante neste aspecto, mas, no caso de sistemas de grande porte, em que a avaliação deste fator pode ser dificultada, ele assume papel secundário em relação à conectividade. Outros fatores como complexidade do tipo de aplicação, rotatividade de pessoal nas equipes e qualidade da documentação contribuem para o aumento do custo de manutenção [30], entretanto têm importância menor frente à conectividade.

Diante da necessidade de uma modificação qualquer no sistema, por exemplo em decorrência de uma alteração de um requisito ou de uma correção de um erro identificado, em primeiro lugar, deve ser possível a análise completa do impacto desta modificação. Deve ser possível que o projetista ou o implementador possa identificar quais outros módulos do sistema sofrerão impacto em conseqüência da modificação a ser realizada. Deixar de analisar esse impacto favorece o surgimento de problemas em outros pontos do sistema e, na pior das hipóteses, o sistema pode passar a operar de forma incorreta. A conectividade é o aspecto principal nesta análise, pois o fato de um módulo estar conectado a outro implica que uma alteração em um dos módulos pode gerar impacto no outro. Diante de um nível de conectividade alto, deve ser possível realizar uma análise sobre a estrutura do sistema, a estrutura de seus módulos e a forma como as conexões se estabelecem nele e atuar nos fatores que elevam a conectividade com o objetivo de reduzi-la.

## 3 Descrição do Modelo MACSOO

O Modelo de Avaliação de Conectividade em Sistemas Orientados por Objetos (MACSOO) visa contribuir para a solução de um dos problemas mais graves vivenciados pela comunidade produtora de software: a instabilidade de sistema, tendo como base a conectividade como fator mais importante. MACSOO é um método que usa métricas para avaliar os principais fatores de impacto na conectividade em sistemas OO, e, diante de um alto grau de conectividade, indica os passos do processo decisório para sua redução. O processo pro-

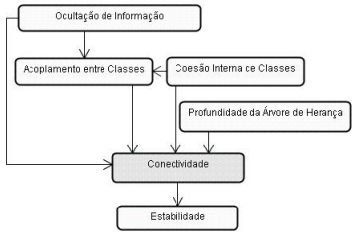


**Figura 1.** Fatores Determinantes da Conectividade

posto em MACSOO é realimentado, pois baseia-se na análise da conectividade, intervenção no sistema e reavaliação da conectividade para verificar o impacto da intervenção realizada. MACSOO pode ser utilizado tanto na fase de desenvolvimento quanto nas atividades de reestruturação de software. Constitui-se (1) da indicação dos fatores mais importantes de impacto na conectividade; (2) da indicação de um conjunto de métricas necessárias para avaliação dos fatores envolvidos; (3) de passos e estágios para a obtenção de software com baixo grau de conectividade. Esses tópicos são detalhados a seguir.

### 3.1 Fatores de Impacto na Conectividade

Identificamos aqui quatro dos mais importantes fatores que contribuem para a conectividade de um sistema: profundidade da árvore de herança, ocultação de informação, coesão interna de classes e acoplamento entre classes. O relacionamento entre esses fatores bem como suas respectivas importâncias para a *conectividade* são analisadas a seguir. Essa relação é ilustrada na Figura 1.

• Ocultação de Informação*:* manter as informações de determinada classe o mais oculta possível dentro dela contribuem para a diminuição da conectividade do sistema, uma vez que minimiza o número de portas abertas para conexão com uma classe. Além disso, a ocultação de informação contribui para a diminuição do grau de acoplamento, pois evita o surgimento de acoplamentos de alto grau decorrentes do acesso direto a dados de um módulo por outros módulos.

• Acoplamento entre Classes*:* em ambientes OO, duas formas de conexões podem ser estabelecidas entre duas classes: por herança, quando uma classe herda características de outra, ou por uso, quando uma classe usa serviços ou dados de outra, o que caracteriza uma relação do tipo cliente-servidor [33]. Em conexões estabelecidas por uso, as classes envolvidas são estruturadas de tal forma que objetos de uma fazem referência a objetos da outra. Ideal-

mente, neste tipo de conexão, um objeto deve usar o outro apenas via suas interfaces públicas, o que garante o respeito à ocultação de informação e ao encapsulamento. Para conhecer o impacto real das conexões estabelecidas em um sistema, é preciso conhecer as características de tais conexões. É necessário conhecer a largura de cada conexão, que corresponde ao grau de acoplamento existente nelas. Uma conexão fina é aquela que resulta em baixo grau de acoplamento, por exemplo, quando um método de uma classe invoca um método da outra com passagem de parâmetro por valor; uma conexão grossa resulta na existência de acoplamento forte entre duas classes, por exemplo, quando uma atualiza diretamente um dado da outra. Uma conexão fina não representa para o sistema o mesmo grau de dependência de uma conexão grossa. Efeitos de alterações em um sistema cujas conexões sejam finas são mais controláveis do que as alterações em um segundo sistema com o mesmo nível de conectividade do primeiro, porém com conexões mais grossas. A espessura das conexões – o grau de acoplamento – é um fator de impacto na conectividade de sistemas, porque atuando-se sobre o grau de acoplamento das conexões do tipo cliente-servidor do sistema, obtém-se um ganho quando se consegue diminuir a largura das conexões ou quando, na melhor das hipóteses, consegue-se eliminar conexões.

• Coesão Interna de Classes*:* módulos pouco coesos tendem a realizar tarefas diversas e a quantidade de conexões com os demais módulos tende a ser maior, assim como o grau de acoplamento estabelecido com outras classes tende a ser maior. A reestruturação de uma classe de baixa coesão interna pode resultar na criação de duas ou mais classes de alto grau de coesão interna. A conectividade das classes resultantes tende a ser menor do que a da classe original, o que impacta na diminuição da conectividade do sistema.

• Profundidade da Árvore de Herança*:* a conexão de herança surge quando uma classe *B* é herdeira de uma classe *A*. Neste caso, existe entre *A* e *B* uma relação estreita que determina que uma alteração em *A* impacta em alteração em *B*. Embora haja um consenso na literatura que a herança é um recurso poderoso da OO para a obtenção de reusabilidade [23], estudos revelam que seu uso deve ser cauteloso [4]. Gamma et al. [12] denominam a reutilização obtida por herança como reutilização caixa-branca, pois os detalhes de implementação de uma classe ficam expostos às suas descendentes e uma mudança na superclasse impacta em alterações nas suas descendentes. Daly et al. [7] mostram que árvores de herança profundas provocam perda de manutenibilidade de software, pois quanto mais profunda a árvore, maior o tempo e maior a dificuldade para compreendê-la. Para Sommerville [34], a herança introduz dificuldades para análise e entendimento do comportamento dos objetos, o que torna mais difícil solucionar erros nos sistemas.

### 3.2 Conjunto de Métricas

Indicamos em MACSOO as métricas descritas a seguir para avaliação de estabilidade e conectividade. A idéia do modelo é: a métrica estabilidade, definida por Myers[25], denota o *impacto de manutenção* um software; as demais métricas são indicadores de conectividade e de seus aspectos determinantes. A coleta de uma métrica de estabilidade justifica-se por ser um parâmetro de comparação para a métrica de conectividade, ou seja, para baixa estabilidade, espera-se um valor alto de conectividade, e para alta estabilidade, espera-se um valor baixo de conectividade.

**Estabilidade:** A métrica de Estabilidade proposta Myers [25] indica o *impacto de manutenção*, a quantidade média de módulos do sistema que sofrerão impacto em decorrência da alteração de um módulo qualquer no sistema, bem como, para cada módulo, a quantidade de módulos que sofrerão impacto devido a sua modificação. No cálculo desta métrica, o sistema é representado por um grafo não direcionado. Módulos do sistema são representados pelos vértices do grafo. Uma aresta entre dois vértices representa a existência de comunicação entre os dois módulos. A cada tipo de acoplamento entre módulos e coesão interna de módulos é associado um valor entre 0 e 1, com base na análise dos problemas ocasionados por cada tipo de coesão e acoplamento.

O cálculo é feito em duas fases: a primeira gera uma matriz de dependência de primeira ordem, que considera os fatores coesão e acoplamento para cada par de módulos. A segunda fase deriva uma matriz de dependência completa a partir da matriz de dependência de primeira ordem. Seja *m* o número de módulos de um sistema. A matriz de dependência de primeira ordem é obtida da seguinte forma:

1. Constrói-se uma matriz *C*, com dimensões $m \times m$. Avalia-se o tipo de acoplamento existente entre cada par de módulo e preenche-se a posição $C_{ij}$ da matriz, correspondente ao par analisado, com o valor associado ao tipo de acoplamento.
2. Constrói-se um vetor *S* de *m* posições. Avalia-se o tipo de coesão interna de cada módulo e preenche-se a posição $S_i$ correspondente do vetor como o valor associado ao tipo de coesão.
3. Constrói-se a matriz *D* de dependência de primeira ordem, a partir da seguinte fórmula:
   $D_{ij} = 0.15( S_i + S_j) + 0.7C_{ij}$, se $C_{ij} \neq 0$
   $D_{ij} = 0$ , se $C_{ij} = 0$
   $D_{ii} = 1$ para todo i

De acordo com Myers, esta fórmula não foi derivada empiricamente, foi definida como uma equação linear baseada em acoplamento e coesão devido ao fato de que estes dois as-

pectos têm impacto na estabilidade de programas. Os dois fatores utilizados na fórmula, 0.15 associado à coesão e 0.7 ao acoplamento, representam a idéia de que o acoplamento é mais importante do que a coesão para a estabilidade do software. Deriva-se uma nova matriz *E* de dependência completa, considerando os efeitos indiretos de alteração. Em sistemas reais, a estrutura é geralmente complexa e pode existir inúmeros caminhos de se chegar de um módulo *i* a um módulo *j*. Esse modelo de avaliação propõe utilizar os três caminhos com maior peso, isto é, os três caminhos com maior probabilidade de impacto de uma alteração em *j* devido a uma alteração ocorrida em *i*. A probabilidade de um caminho é dada pelo produto das probabilidades de cada aresta do caminho.

A matriz *E* é obtida da seguinte forma: para cada par de módulos *i* e *j*:

1. Encontram-se todos os caminhos de *i* para *j*.
2. Se há apenas um caminho de *i* para *j*, $E_{ij} = E_{ji}$ = P(x), onde P(x) é a probabilidade do caminho.
3. Se há dois caminhos de *i* para *j*, $E_{ij} = E_{ji}$ = P(x) + P(y) - P(x)P(y), onde P(x) e P(y) são as probabilidades dos dois caminhos.
4. Se há três caminhos de *i* para *j*, $E_{ij} = E_{ji}$ = P(x) + P(y) + P(z) - P(x)P(y) - P(x)P(z) - P(y)P(z) + P(x)P(y)P(z), onde P(x), P(y) e P(z) são as probabilidades dos três caminhos.
5. Se há mais de três caminhos de *i* a *j*, encontram-se os caminhos de maiores probabilidades. Aplica-se a regra do item anterior para esses três caminhos.

A matriz *E* de dependência completa obtida fornece as seguintes informações:

- Quantidade de módulos que serão alterados no total em decorrência de uma alteração em um módulo qualquer do sistema. Quanto menor esse número, maior a estabilidade e melhor a manutenibilidade. Esse valor é obtido somando-se os valores de todas as posições da matriz e dividindo-se o resultado pelo total de módulos do sistema.
- A soma dos elementos de cada linha da matriz E fornece o total de módulos que são alterados em conseqüência da alteração do módulo representado pela linha.

Para adaptar esta métricas à OO, consideramos uma classe como um módulo e propomos uma escala de acoplamento e coesão para o paradigma OO, descrita em [10]. Myers não define como proceder no caso de um módulo estar acoplado a outro por mais de uma forma; neste caso, definimos que deve ser considerada conexão com maior grau de acoplamento, o

que é avaliado pela métrica *peso da conexão aferente*, descrita posteriormente. Os demais passos do cálculo da métrica são mantidos.

MACSOO usa esta métrica de estabilidade adaptada à OO como indicador do custo de manutenção de um software. Por exemplo, se esta métrica resulta em 20 para um sistema que possui 100 classes, significa que o impacto da manutenção é de 20% do total de classes do sistema.

**Conectividade:** Consideramos que uma classe *A* está conectada a uma classe *B* se *A* usa um serviço ou um atributo de *B* ou se é sub-classe de *B*. Em um software com *n* classes, o maior número possível de conexões é $n^2 - n$. A métrica *conectividade* refere-se à métrica COF do conjunto MOOD [1], cujo cálculo é a razão entre o número total de conexões existentes entre as classes do software e o maior número possível de conexões para o software. Por exemplo, o cálculo de COF para um software com 20 classes que possui 10 conexões é dado por $10/\left(20^2 - 20\right) = 0,026$. Um software totalmente conectado possui COF = 1. Um software fortemente conectado possui estrutura rígida, baixo grau de independência entre os módulos e, conseqüentemente, o custo na sua manutenção é explosivo.

**Quantidade de elementos públicos:** é a soma de métodos e atributos públicos de uma classe. Permite identificar as classes com um maior número de *conectores* (atributos e métodos), pois estas, possivelmente, são as que mais contribuem para a alta conectividade do sistema.

**Profundidade das árvores de herança:** indica a posição de uma classe na árvore de herança da qual faz parte; é a métrica DIT (*Depth of Inheritance Tree*), que faz parte do conjunto de métricas CK [6]. Com DIT, é possível identificar as hierarquias de maior profundidade e, a partir daí, verificar a necessidade de reestruturação delas.

**Grau do acoplamento entre classes:** conexões cujo grau de acoplamento seja alto determinam um grau de dependência maior do que aquelas com grau de acoplamento mais fraco. Identificar tais conexões é ponto chave para auxiliar na redução da conectividade e no aumento da estabilidade do sistema.

**Grau de coesão interna da classe:** a coesão interna de uma classe impacta na conectividade porque classes pouco coesas tendem a realizar mais serviços. Assim, para a redução da conectividade, é importante identificar essas classes para que se possa avaliar a necessidade de reestruturação delas. Avaliação de uma classe inclui a avaliação do relacionamento entre os seus elementos, entre os elementos de suas interface e de seus métodos. LCOM (*Lack of Cohesion in Methods*) [6] é uma métrica útil neste aspecto, indica a ausência de coesão entre os métodos de uma classe.

**Conexões aferentes:** as conexões entre classes podem ser *eferentes* - aquelas que se originam nas classes, ou *aferentes* - aquelas que chegam até ela. Na avaliação de conectividade, propomos a métrica número de conexões aferentes, importante para identificar as classes determinantes para a conectividade geral do software. Esta métrica é dada pela quantidade de classes que conectam-se à classe em questão.

**Peso da conexão aferente:** dada a importância de se conhecer as conexões de maior impacto no sistema, propomos aqui uma métrica para este aspecto. Duas classes podem estar conectadas por mais de uma conexão e a cada uma delas corresponde um grau de acoplamento. Por exemplo, uma classe *A* pode usar um método e um atributo de uma classe *B*, existindo duas conexões de *A* para *B*, sendo que aquela em que há uso de atributo corresponde a maior grau de acoplamento do que a outra. A métrica *peso da conexão aferente* entre uma classe *A* e uma classe *B* é dada pelo valor correspondente ao maior grau de acoplamento entre elas.

### 3.3 Estágios e Passos de MACSOO

MACSOO é estruturado em estágios e passos. Os estágios correspondem à avaliação dos fatores relacionados à estabilidade e à conectividade. Cada estágio é constituído por passos, que são as etapas necessárias para concluir a avaliação e a melhoria do fator avaliado. Nos passos coletam-se métricas, avalia-se o resultado e atua-se no respectivo fator com o objetivo de redução da conectividade do sistema. A Figura 2 ilustra a estrutura de MACSOO.

A seguir são descritos os estágios de MACSOO e, para cada um, seus respectivos passos e métricas usadas:

• **Avaliação de estabilidade***:*

1. *Avalie a estabilidade do software:* consiste na coleta e avaliação da *métrica para Estabilidade proposta por Myers adaptada à OO*.

• **Avaliação de conectividade do sistema***:*

1. *Obtenha métricas de conectividade do sistema:* coleta-se a métrica *conectividade*. Se o resultado obtido neste passo indicar alta conectividade, deve-se realizar o passo seguinte a fim de identificar as classes críticas e avaliá-las.

• **Identificação das classes mais conectadas***:*

1. *Identifique as classes mais conectadas no sistema:* obtém-se para cada classe do sistema a métrica *conexões aferentes*. Este é um passo importante pois nele identificam-se

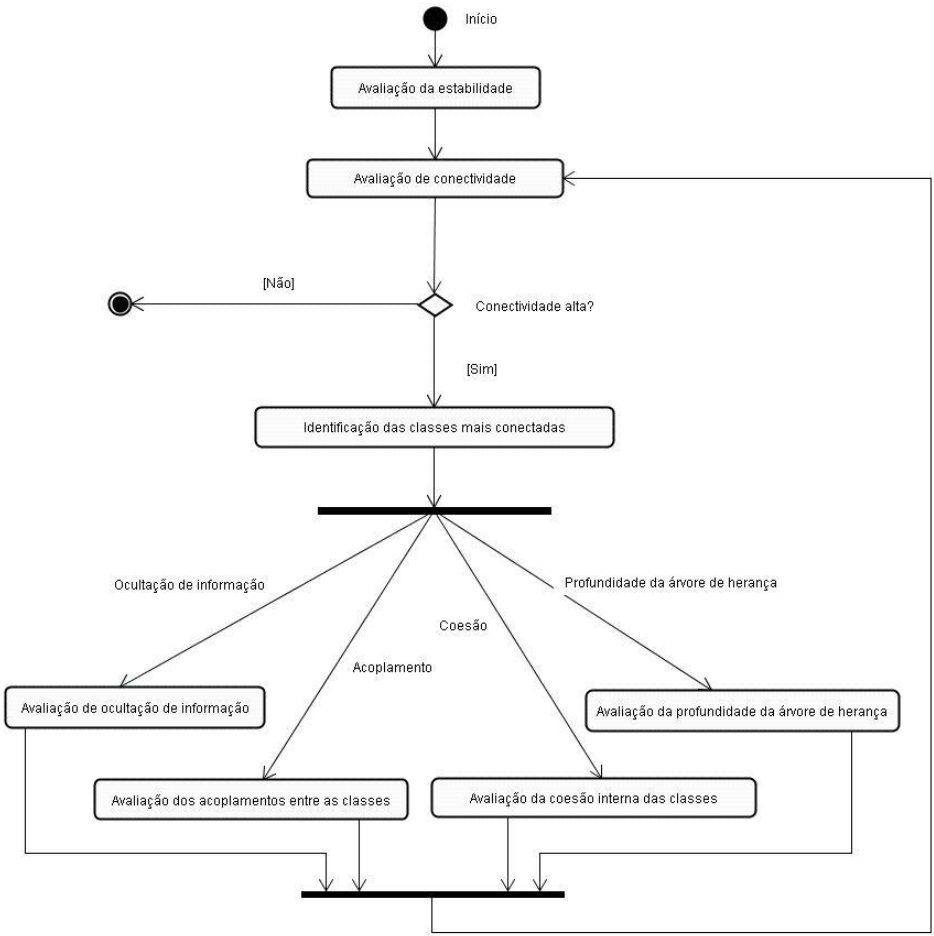


**Figura 2.** Estágios de MACSOO

as classes cujas mudanças têm grande impacto no sistema. Deve-se analisar tais classes sob os aspectos: ocultação de informação, acoplamento com outras classes, coesão interna e a profundidade na árvore de herança. Neste ponto, então, para cada classe identificada como crítica, deve-se analisar os aspectos acoplamento, profundidade na árvore de herança, coesão interna e ocultação de informação a fim de se identificar aqueles que necessitam melhorias.

2. *Avalie os acoplamentos entre as classes:* esse passo representa a realização dos passos descritos no estágio de *Avaliação de acoplamento*.
3. *Avalie a coesão interna das classes:* esse passo representa a realização dos passos descritos no estágio de *Avaliação de coesão interna*.
4. *Avalie a ocultação de informação:* este passo representa a realização dos passos descritos no estágio de *Avaliação de ocultação de informação*.
5. *Avalie a profundidade da árvore de herança:* representa a realização dos passos descritos no estágio de *Avaliação de profundidade da árvore de herança*.

• **Avaliação de acoplamento***:*

1. *Identifique conexões de grau de acoplamento elevado:* usa-se a métrica *peso de conexão aferente* que indica as conexões de maior impacto na estabilidade do sistema.
2. *Reestruture as classes envolvidas:* analisa as classes envolvidas, bem como os relacionamentos existentes entre elas. Para diminuir a conectividade, na reestruturação das classes convém seguir as regras definidas por Meyer [23] para a construção de software modular, em especial: poucas interfaces, interface pequena e ocultação de informação.

• **Avaliação de coesão***:*

1. *Avalie a coesão da classe:* obtém-se a métrica LCOM de coesão interna das classes.
2. *Reestruture a classe:* visa o aumento da coesão interna da classe, principalmente de sua interface, uma vez que é por meio dela que uma classe se conecta com as demais.

• **Avaliação de ocultação de informação***:*

1. *Obtenha métricas de ocultação de informação da classe:* a métrica usada é a quantidade de elementos públicos. Valor alto para esse indicador pode denotar a causa do alto grau de conectividade da classe.
2. *Reestruture a classe:* caso o resultado obtido no passo anterior não seja satisfatório, é necessário reestruturar a classe para ocultar o máximo possível as suas informações. Neste caso, é possível que outras classes do sistema sejam afetadas e demandem reestruturação também.

• **Avaliação da profundidade da árvore de herança***:*

1. *Obtenha métrica da profundidade da árvore de herança:* neste passo, coleta-se a métrica DIT para a classe em questão.

2. *Reestruture a árvore de herança:* um valor alto para esta métrica indica que a árvore na qual a classe está inserida pode precisar ser reestruturada, em benefício da diminuição da conectividade do sistema. Deve-se, então, avaliar a hierarquia à qual a classe pertence, buscando obter hierarquias de profundidade menores.

## 4 Experimentos e Resultados

Desenvolvemos a ferramenta *Connecta* para aplicarmos MACSOO. Esta ferramenta foi utilizada para a realização dos experimentos relatados aqui; foi desenvolvida em Java e coleta métricas em software implementado em Java. Baseia-se na análise do código das classes de um sistema, que é feita diretamente no *bytecode* das classes. A razão desta decisão é prover maior portabilidade da ferramenta em relação às diferenças entre as versões de Java. Foi usada a biblioteca BCEL (*Byte Code Engineering Library*) [3], que fornece recursos para análise de arquivos de *bytecode* de Java. Apresentamos nesta seção 2 experimentos realizados com o objetivo de avaliar o modelo proposto. O primeiro experimento, denominado Caso 1, exemplifica e avalia o uso de MACSOO. O segundo, denominado Caso 2, investiga se a conectividade pode ser usada para aferir a manutenibilidade de um software, comparando os valores aferidos pela métrica conectivade com os valores da métrica estabilidade e a avaliação realizada por um especialista.

### 4.1 Caso 1: Controle Acadêmico

Este experimento teve por objetivo exemplificar o uso de MACSOO e apresentar a forma como os valores das métricas devem ser interpretados. O experimento foi planejado da seguinte forma: uma aplicação pequena é construída para que se possa analisar os resultados em um nível de detalhes pequeno; a primeira versão da aplicação deve apresentar transgressões a princípios de modularidade de software, como a utilização de dados públicos e baixa coesão das classes; gradualmente, reestruturamos a aplicação de acordo com MACSOO; para cada uma das versões da aplicação, coletamos suas métricas usando *Connecta*. A hipótese investigada neste experimento é que com as reestruturações realizadas de forma a melhorar a modularidade, a conectividade diminui e a estabilidade aumenta. Como a estabilidade refere-se ao impacto de uma manutenção realizada no software, referenciamos esta métrica aqui também como *impacto de manutenção*.

• **Versão 1 do Controle Acadêmico:** nesta versão, o controle acadêmico é constituído pelas

classes: `Constantes`, que possui todas as constantes usadas na aplicação; `ProfessorAluno` contém dados e métodos aplicáveis a professores e alunos; `Disciplina` e `Turma`. A Versão 1 possui classes de baixa coesão e um número grande de conexões entre elas, com forte acoplamento, devido ao uso de atributos públicos. Para esta versão, *Connecta* gerou os resultados reportados nas Tabelas 1 e 2.

A aplicação possui 4 classes e há 6 conexões no total entre elas, o que resulta em conectividade igual a 0,30. A métrica de estabilidade da aplicação, que corresponde ao *impacto de manutenção*, resultou em 2,1, o que indica que a cada necessidade de manutenção na aplicação, 2,1 classes sofrerão alterações. Considerando-se que o sistema possui apenas 4 classes, este número indica que 52% do sistema será alterado, o que é uma situação crítica. Desta forma, como orienta MACSOO, deve-se buscar formas de reduzir a conectividade por meio da intervenção nos fatores que a determinam.

Um problema estrutural grave desta versão é o uso de dados públicos. Em todas as classes da aplicação, todos os atributos são públicos. Este fato, por si só, não determina alta conectividade. Contudo, as classes usam os dados públicos umas das outras, o que origina o surgimento de conexões patológicas nas quais o grau de acoplamento entre as classes é altíssimo. Esta característica das classes da aplicação é refletida pelos valores obtidos para as métricas *conexões aferentes* e *impacto de manutenção*. A classe que possui o maior número de conexões aferentes é também aquela cuja alteração tem maior impacto: `Constantes`; para ela, obteve-se o valor 3,8 para a métrica *impacto de manutenção*. Este valor indica que para cada necessidade de manutenção nesta classe, no total, 3,8 classes serão afetadas, incluindo a própria classe. Como o sistema possui 4 classes, este valor representa 95% do total de suas classes, o que é um valor alto.

Outra questão de impacto no Caso 1 é a coesão interna das classes. Na análise qualitativa da estrutura desta versão, verifica-se que a classe `ProfessorAluno` agrega características de duas classes do domínio do problema, professor e aluno, o que caracteriza baixa coesão interna. Esta classe tem coesão interna igual a 3. Isso significa que a diferença entre a quantidade de pares de métodos sem similaridade e a quantidade de pares com similaridade é igual a 3. Considerando-se que a classe em questão possui 3 métodos, este número é alto, o que reflete que ela possui baixa coesão interna. Da mesma forma, observa-se que a classe `Constantes` apresenta problemas no aspecto coesão. Embora a métrica coesão interna seja igual a 1 para esta classe, o que poderia levar a se pensar que a classe tem um bom nível de coesão interna, observando sua estrutura, associada ao conhecimento do domínio do problema, vê-se que sua coesão é baixa, visto que trata-se de uma classe depositária de todas as constantes usadas na aplicação.

Considerando-se que a coesão interna das classes de um sistema impacta na sua conectividade, as classes `ProfessorAluno` e `Constantes` foram reestruturadas e os resultados das novas versões são descritos a seguir.

Tabela 1: Resultados Versão 1.

| *Métrica* | *Valor* |
|---|---|
| Total de Classes | 4 |
| Quantidade Conexões | 6 |
| Conectividade | 0,30 |
| Estabilidade | 2,1 |

Tabela 2: Resultados Versão 1 - classes

| *Classe* | *Métrica* | *Valor* |
|---|---|---|
| Constantes | Atributos públicos | 12 |
| | Coesão interna | 1 |
| | Conexões aferentes | 3 |
| | Impacto manutenção | 3,8 |
| ProfessorAluno | Atributos públicos | 6 |
| | Coesão interna | 3 |
| | Conexões aferentes | 1 |
| | Impacto manutenção | 1,9 |
| Turma | Atributos públicos | 3 |
| | Coesão interna | 1 |
| | Conexões aferentes | 1 |
| | Impacto manutenção | 1,9 |
| Disciplina | Atributos públicos | 3 |
| | Coesão interna | 0 |
| | Conexões aferentes | 1 |
| | Impacto manutenção | 1,9 |

Tabela 3: Resultados da Versão 2.

| *Métrica* | *Valor* |
|---|---|
| Total de Classes | 6 |
| Quantidade Conexões | 8 |
| Conectividade | 0,27 |
| Estabilidade | 2,2 |

Tabela 4: Resultados da Versão 2 - classes

| *Classe* | *Métrica* | *Valor* |
|---|---|---|
| Constantes | Atributos públicos | 12 |
| | Coesão interna | 1 |
| | Conexões aferentes | 4 |
| | Impacto manutenção | 4,7 |
| Professor | Atributos públicos | 4 |
| | Coesão interna | 1 |
| | Conexões aferentes | 1 |
| | Impacto manutenção | 1,9 |
| Aluno | Atributos públicos | 3 |
| | Coesão interna | 1 |
| | Conexões aferentes | 1 |
| | Impacto manutenção | 1,9 |

• **Versão 2 do Controle Acadêmico:** nesta versão, a classe `ProfessorAluno` foi substituída por duas classes de coesão interna maior: `Professor` e `Aluno`. Tabelas 3 e 4 mostram os resultados gerados por *Connecta* para a Versão 2.

A melhor coesão interna das classes `Professor` e `Aluno` refletiu-se no resultado obtido para a métrica coesão interna, que é igual a 1 para estas classes. A reestruturação da classe `ProfessorAluno` diminuiu levemente o grau de conectividade da aplicação, que passou para 0,27. O impacto de manutenção das classes resultantes continuou com o mesmo valor da classe na Versão 1, o que se explica pelo fato de as conexões ainda ocorrerem por meio de atributos públicos. As métricas coletadas indicam melhoria na estabilidade do sistema. Na Versão 2, para cada necessidade de manutenção do sistema, 2,2 classes são afetadas, o que corresponde a 37% do total de classes do sistema.

• **Versão 3 do Controle Acadêmico:** a classe `Constantes` contém todas as constantes da aplicação. Na Versão 3, essa classe foi trocada por 3 classes de coesão interna maior: `ConstantesAluno`, `ConstantesProfessor` e `ConstantesDisciplina`.
As Tabelas 5 e 6 mostram os resultados gerados por *Connecta* para a Versão 3.

A reestruturação da classe `Constantes` resultou na diminuição do grau de conectividade do sistema, que passou de 0,27 para 0,16. A métrica para estabilidade, nesta versão, aponta que 2,1 classes demandarão alterações, no caso de uma necessidade de manutenção do sistema, o que corresponde a 26% do total de classes. As intervenções realizadas nas Versões 2 e 3 da aplicação mostram que classes coesas têm impacto positivo na conectividade do sistema e na sua estabilidade.

Tabela 5: Resultados da Versão 3.

| ***Métrica*** | ***Valor*** |
|---|---|
| Total de Classes | 8 |
| Quantidade Conexões | 9 |
| Conectividade | 0,16 |
| Estabilidade | 2,1 |

Tabela 6: Resultados da Versão 3 - classes

| ***Classe*** | ***Métrica*** | ***Valor*** |
|---|---|---|
| ConstantesAluno | Atributos públicos | 6 |
| | Coesão interna | 1 |
| | Conexões aferentes | 2 |
| | Impacto manutenção | 2,9 |
| ConstantesProfessor | Atributos públicos | 4 |
| | Coesão interna | 1 |
| | Conexões aferentes | 2 |
| | Impacto manutenção | 2,9 |
| ConstantesDisciplina | Atributos públicos | 2 |
| | Coesão interna | 1 |
| | Conexões aferentes | 1 |
| | Impacto manutenção | 2,7 |

Tabela 7: Resultados da Versão 4

| ***Métrica*** | ***Valor*** |
|---|---|
| Total de Classes | 8 |
| Quantidade Conexões | 7 |
| Conectividade | 0,13 |
| Estabilidade | 1,5 |

• **Versão 4 do Controle Acadêmico:** as versões anteriores desta aplicação usam atributos públicos nas classes. Aqui, eliminamos o uso de dados públicos nas principais classes. As únicas exceções foram as classes depositárias de constantes, que continuam com atributos públicos. Tabelas 7 e 8 mostram os resultados da Versão 4 gerados por *Connecta*.

Eliminando os dados públicos, o impacto de manutenção das classes `Aluno`, `Turma` e `Professor` diminui. Contudo, estas classes ainda têm impacto relevante no sistema, dado o valor 1,4 para a métrica *impacto de manutenção*. Isto deve-se ao fato de que as conexões com tais classes não foram eliminadas; o que mudou foi a forma como as conexões ocorrem, pois, nesta versão, a comunicação ocorre por invocação de métodos. Foram eliminadas também as conexões que as classes `Professor`, `Aluno` e `Disciplina` tinham com as classes `ConstantesProfessor`, `ConstantesAluno` e `ConstantesDisciplina`, respectivamente. O uso destas classes concentrou-se na classe `Principal`. A Versão 4 da aplicação apresenta grau de conectividade igual a 0,13 e a métrica de estabilidade aponta que 1,5 classes sofrerão impacto, no caso de uma manutenção no sistema, o que corresponde a 19% do total de classes. Comparando-se a Versão 4 com a Versão 1, na avaliação qualitativa do software em questão, conclui-se que a intervenção nos aspectos de qualidade estrutural de software, como coesão, ocultação de informação e acoplamento, contribui para redução de conectividade em sistemas. Os dados coletados neste experimento refletem a relação entre a conetividade e a estabilidade de um software, estando de acordo como a idéia de que o grau de estabilidade de um sistema pode ser aferido pelo grau de sua conectividade.

Tabela 8: Resultados da Versão 4 - classes

| *Classe* | *Métrica* | *Valor* | *Classe* | *Métrica* | *Valor* |
|---|---|---|---|---|---|
| ConstantesAluno | Atributos públicos | 6 | Turma | Atributos públicos | 0 |
| | Coesão interna | 1 | | Coesão interna | 21 |
| | Conexões aferentes | 1 | | Conexões aferentes | 1 |
| | Impacto manutenção | 1,9 | | Impacto manutenção | 1,4 |
| ConstantesProfessor | Atributos públicos | 4 | Disciplina | Atributos públicos | 0 |
| | Coesão interna | 1 | | Coesão interna | 21 |
| | Conexões aferentes | 1 | | Conexões aferentes | 1 |
| | Impacto manutenção | 1,9 | | Impacto manutenção | 1,4 |
| ConstantesDisciplina | Atributos públicos | 2 | Professor | Atributos públicos | 0 |
| | Coesão interna | 1 | | Coesão interna | 55 |
| | Conexões aferentes | 1 | | Conexões aferentes | 1 |
| | Impacto manutenção | 1,9 | | Impacto manutenção | 1,4 |
| Aluno | Atributos públicos | 0 | | | |
| | Coesão interna | 45 | | | |
| | Conexões aferentes | 1 | | | |
| | Impacto manutenção | 1,4 | | | |

## 4.2 Caso 2: Análise de um Conjunto de Aplicações

O Caso 2 teve por objetivo investigar experimentalmente se a conectividade pode ser usada para aferir a dificuldade de manutenção de um software. Sabe-se que a qualidade estrutural de um software é determinante para a sua facilidade de manutenção. Desta forma, neste experimento buscamos comparar os valores aferidos pela métrica de conectivade em um conjunto de aplicações com a avaliação da qualidade estrutural do software realizada por um especialista. Além disso, tendo por base que a métrica Estabilidade pode ser usada para aferir a manutenibilidade de um software, comparamos também os valores da métrica de conectividade com a de estabilidade. Neste experimento foram usados os produtos dos trabalhos realizados por alunos da disciplina Linguagens de Programação em um curso de graduação em computação. O grupo de alunos era heterogêneo, contando com indivíduos com proficiência baixa, média e alta em programação OO. Foi solicitado que eles desenvolvessem uma aplicação para a gestão de uma clínica médica, cujo objetivo é manter os cadastros de especialidades médicas, médicos, pacientes e prontuários de pacientes. As 11 aplicações implementadas pelos alunos foram avaliadas qualitativamente pelo professor da disciplina que atribuiu notas aos trabalhos, em uma escala de 0 a 10, considerando o aspecto de qualidade estrutural do software. Foram coletadas as métricas de estabilidade e conectividade das aplicações. Os resultados das métricas foram comparados à avaliação do professor.

A hipótese investigada por este experimento é que os valores aferidos pela métrica de estabilidade devem ser proporcionais à nota dada pelo professor, e que a de conectividade deve ser inversamente proporcional, ou seja, quanto maior a qualidade estrutural do software, menor a sua conectividade e maior a sua estabilidade.

Tabela 9: Resultados do Caso 2

| Aplicação | Classes | Conexões | Conectividade (%) | Estabilidade (classes) | Estabilidade (%) | Nota | Conceito |
|---|---|---|---|---|---|---|---|
| 1 | 64 | 13 | 3 | 2,7 | 4,2 | 9 | Ótimo |
| 2 | 48 | 93 | 4 | 2,7 | 5,6 | 8 | Bom |
| 3 | 31 | 52 | 6 | 2,2 | 7,1 | 9 | Ótimo |
| 4 | 29 | 52 | 6 | 2,1 | 7,2 | 10 | Excelente |
| 5 | 28 | 52 | 7 | 2,3 | 8,2 | 10 | Excelente |
| 6 | 14 | 13 | 7 | 1,3 | 9,3 | 8 | Bom |
| 7 | 25 | 47 | 8 | 2,2 | 8,8 | 6 | Razoável |
| 8 | 11 | 17 | 16 | 1,8 | 16,4 | 6 | Razoável |
| 9 | 14 | 29 | 16 | 2,7 | 19,3 | 6 | Razoável |
| 10 | 10 | 15 | 17 | 2,6 | 26,0 | 3 | Fraco |
| 11 | 5 | 4 | 20 | 1,7 | 34,0 | 2 | Fraco |

As aplicações implementadas pelos alunos foram avaliadas qualitativamente atribuindo-se notas aos trabalhos, em uma escala de 0 a 10, apresentadas na Tabela 9. O resultado desta avaliação é o seguinte:

• Aplicação 1: não são usados dados públicos. No geral as classes estão bem construídas, com responsabilidades específicas, exceto uma delas, que deveria implementar as características de prontuários, mas possui também dados da classe `Paciente`.

• Aplicação 2: não são usados dados públicos. As classes de interface realizam algumas funções de leitura e gravação de dados em disco, resultando em baixa coesão.

• Aplicação 3: aplicação bem construída. A única ressalva é quanto o uso de vetor estático para o armazenamento de objetos da classe `Paciente` dentro da própria classe `Paciente`. Visto que este não é um dado desta classe, esse não é um local apropriado para o seu uso. Isso também ocorre com as demais classes de negócio da aplicação.

• Aplicação 4: a estrutura é muito boa, não foram usados dados públicos, o relacionamento entre as classes estão apropriados e as classes possuem papéis bem definidos.

• Aplicação 5: é a melhor aplicação, do ponto de vista estrutural, dentre as avaliadas. As classes foram separadas em pacotes lógicos, são altamente coesas, não são usados dados públicos. Foi construída uma classe utilitária para realizar funções como formatação de CEP, formatação de telefone, etc.

• Aplicação 6: embora as classes tenham sido divididas em pacotes lógicos, as classes de negócio realizam também funções de interface de usuário.

• Aplicação 7: nesta aplicação foi usada conexão com banco de dados. Há problemas de coesão das classes. As funcionalidades de conexão com banco de dados e recuperação de

dados estão espalhadas por todas as classes do sistema.

• Aplicação 8: usou-se nesta aplicação o armazenamento e a recuperação de dados em arquivos. As funcionalidades de gravação de dados em arquivos estão localizadas nas classes de negócio. Foram construídas classes para realizar a recuperação de dados de arquivos, mas estas também realizam funções de interface de usuário. Nas classes de negócio também foram incluídas funções referentes a interação com o usuário.

• Aplicação 9: foi implementada uma única classe para realizar todo o tratamento de interface gráfica com o usuário, o que confere baixa coesão a esta classe. As demais classes estão bem construídas.

• Aplicação 10: não foram aplicados os conceitos de programação orientada por objetos. As classes de negócio possuem apenas dados públicos e um método construtor. As funcionalidades de interface de usuário foram distribuídas em várias classes.

• Aplicação 11: também neste caso, não foram aplicados os conceitos de programação OO, pois as classes possuem apenas dados, todos eles públicos, e um método construtor. Foi construída uma classe principal que realiza todas as funcionalidades do sistema.

Os resultados das coletas das métricas são mostrados na Tabela 9. A coluna *conectividade* mostra o valores na métrica COF, a coluna *Estabilidade* indica o número médio de classes que sofrerão impacto devido à manutenção de um módulo qualquer do sistema, e a coluna *Estabilidade (%)* mostra o percentual que este número representa em relação ao total de classes do sistema. Comparando-se os valores de *conectividade* e *estabilidade (%)*, evidencia-se que a dificuldade de manutenção de uma aplicação é proporcional à conectividade.

A Aplicação 11 possui grau de conectividade igual a 20% e, para cada manutenção feita no sistema, 34% de suas classes sofrerão impacto. De fato, observa-se que nesta aplicação foram usados somente atributos públicos nas classes e elas possuem apenas métodos construtores. A classe principal realiza todas as demais funcionalidades do sistema. A Aplicação 1 apresenta bom nível de conectividade, o que leva a crer que seja uma aplicação construída segundo bons princípios de programação OO. Isso é comprovado pela análise de sua estrutura. Na aplicação, não são usados dados públicos e as classes podem ser agrupadas em classes de negócio, de interface e de persistência, cada uma com responsabilidades bem definidas, com exceção de uma classe.

Comparando-se os resultados da avaliação qualitativa e os resultados quantitativos, conclui-se que, embora não se possa afirmar que há uma concordância perfeita entre ambos, as aplicações consideradas como boas na avaliação qualitativa também foram avaliadas quantitativamente como melhores, no aspecto de impacto de manutenção. Da mesma forma, aquelas aplicações consideradas razoáveis e fracas na avaliação qualitativa foram também apontadas como possuidoras de alto impacto de manutenção. Os resultados obtidos no Caso 2 são significativos na demonstração empírica dos impactos da conectividade de um sistema

na dificuldade de sua manutenção. A análise comparativa entre a avaliação qualitativa e a avaliação quantitativa das aplicaçõesrevela um bom nível de concordância entre ambas.

### 4.3 Conclusão dos Experimentos

Relatamos e analisamos os resultados de dois experimentos realizados com o objetivo de validar MACSOO. O primeiro ilustra que a aplicação do modelo proposto leva à diminuição de conectividade em um sistema. O segundo experimento encontrou relação entre os valores da métrica de conectividade e a avaliação de um especialista sobre a qualidade estrutural dos software: alta conectividade está relacionada a baixa qualidade estrutural do software. Também identificou relação entre os valores de conectividade e estabilidade: quanto mais conectado o software, menor a sua estabilidade, ou seja, menor a sua facilidade de manutenção. Ressaltamos que para concluirmos isso partimos da idéia intuitiva de que quanto menor a estabilidade do software, menor a sua manutenibilidade e do pressuposto que a métrica Estabilidade, proposta por Myers e adaptada por nós, avalia corretamente o aspecto estabilidade. Entretanto, para validar empiricamente esses pressupostos é necessário estudar diferentes cenários de manutenção, medindo, por exemplo, o tempo de manutenção, número de linhas de código alteradas e número de classes alteradas.

Os experimentos relatados neste trabalho têm porte pequeno e, portanto, não é possível generalizar os resultados obtidos para qualquer produto de software. Entretanto, constituem um passo inicial nesta direção, uma vez que podem ser repetidos para software de porte maior e de naturezas diversas.

## 5 Trabalhos Relacionados

O fato de a manutenção ser a fase que corresponde a maior parcela do custo total de um sistema tem motivado muitos trabalhos e investigações com o objetivo de encontrar soluções que reduzam custos e esforços na fase manutenção ou meios de predição destes aspectos. Wei Li et al [21] investigaram a correlação entre a métricas DIT, NOC, LCOM, RFC e WMC, do conjunto CK [6], e o esforço de manutenção de software. A métrica para esforço de manutenção considerada foi o número de linhas de código alteradas por classe. O estudo teve como conclusões que a predição de esforço de manutenção, tal como foi considerado, é possível a partir das métricas utilizadas. O estudo, porém, não avaliou o fator conectividade. Kabaili et al. [18] investigaram a correlação da coesão com a alterabilidade, a capacidade de um software absorver uma alteração, e é determinada pelo impacto de uma alteração, ou seja, o conjunto de classes de necessitam de alteração ou correção em decorrência de uma dada alteração no sistema. Os resultados do experimento não identificaram correlação entre coesão e alterabilidade. Os autores consideram que um dos seguinte fatores contribuiram para isso:

ou as métricas escolhidas não avaliam coesão corretamente, ou de fato não existe correlação entre coesão e alterabilidade. Gyimóty et al. [14] realizaram um estudo empírico sobre a predição de falhas em sistemas OO *open source*, um fator relacionado à manutenibilidade. O objetivo do trabalho é validar as métricas CK para predição de falhas em sistemas OO. Como estudo de caso, utilizaram um banco de dados de falhas detectadas no navegador Mozilla desde a sua versão 1.0 até a 1.7. Os valores das métricas CK foram comparados aos dados da base de falhas do Mozilla, o que levou à seguinte conclusão principal: a métrica CBO (*coupling between objects*) parece ser a melhor métrica para predição de falhas.

Reestruturação de software (do inglês *software refactoring*) foi introduzida por Fowler [11] como a alteração da estrutura do software sem afetar o seu comportamento com o objetivo de melhorar a sua estrutura. O processo de realização de uma reestruturação consiste em identificar qual parte do software será reestruturada, escolher uma reestruturação apropriada como solução e aplicá-la. Identificar pontos de necessidade de reestruturação em software, sobretudo naqueles de grande porte, pode ser uma tarefa inviável. Métricas de software têm sido estudadas há cerca de três décadas [1, 6, 9, 13, 31, 35] e têm sido utilizadas na identificação de necessidades de reestruturação Temos interesse aqui pelas métricas de software OO, dentre as quais destacam-se as do conjunto CK [6] e MOOD [1], utilizadas em MACSOO.

Marinescu [2] (2002 apud Munro et al. [24]) em sua tese de doutorado utilizou métricas de software para definir estratégias de identificar 14 tipos de problemas de desenho e classes que demandam refatorações em código fonte de software OO. Entretanto, como ana- lisa Munro, a escolha das métricas não foi satisfatoriamente justificada. Em seu trabalho, Munro estende o de Marinescu, utilizando métricas de software para identificar automaticamente *bad smell* em código fonte de software OO. Um *bad smell* é informalmente definido por Fowler como problemas de desenho em software, por exemplo, uma classe pouco coesa. Munro seleciona um conjunto de métricas de software que podem ser utilizadas para identificar um sub-conjunto de *bad smells* em código fonte Java. O arcabouço proposto por ele é constituído por: nome do *bad smell*, a descrição informal do problema definida por Fowler [11]; o processo de medição, que consiste na descrição das técninas de medição que podem identificar o problema em código Java; interpretação, que indica um conjunto de regras a serem aplicadas para definir como as métricas podem ser utilizadas para identificar os candidatos à reestruturação.

Munro [24] apresenta a sua proposta para dois *bad smells*: classe *lazy*, caracterizada por ser uma classe que não realiza tarefas relevantes e por isso deve ser eliminada; campo temporário, caracterizado como uma variável de instância em um objeto que é atualizada somente em certas circunstâncias. A proposta foi avaliada a partir de dois estudos de caso: um software de 1500 linhas de código, 13 classes e 124 métodos; outro de 16000 linhas de

[2]Marinescu, R.. *Measurement and Quality in Object-Oriented Design*. Tese de doutorado. Universidade de Timisoara. Outubro de 2002.

código, 730 métodos e 84 classes. Embora a pesquisa tenha mostrado resultados que indicam a eficiência do uso de métricas com o propósito de identificar pontos de reestruturação em sofware, ainda há pontos importantes a serem estudados, como: estender a proposta para os demais problemas de desenho de classe; realizar estudos de caso em sistemas de grande porte e reais; investigar se há uma métrica, ou um conjunto de métricas, que possam ser utilizados na identificação de todos os problemas de desenho de classes. Em nosso trabalho, buscamos a elaboração de um método mais abrangente de reestruturação de software que leve à construção de software com alto grau de manutenibilidade.

Para a aplicação de MACSOO construímos uma ferramenta, denominada Connecta, que coleta as métricas indicadas por MACSOO. A construção de Connecta se fez necessária porque embora haja algumas ferramentas disponíveis que realizam coleta de métricas em software OO [17, 19, 27, 2, 8, 16], nenhuma delas é adequada para a implementação de MACSOO, pois não têm o propósito específico de avaliar conectividade, tampouco coletam todas as métricas necessárias a MACSOO, tais como a métrica de estabilidade de Myers e grau de acoplamento.

## 6 Conclusão

A atividade de manutenção de software é crítica na Engenharia de Software. Acreditamos que a *conectividade* seja um fator preponderante na determinação da estabilidade de um software e, conseqüentemente, de sua manutenibilidade. Neste trabalho apresentamos MACSOO, modelo que visa a avaliação e a diminuição da conectividade em sistemas OO. A idéia do modelo é que a reestruturação de software, para atingir alto grau de manutenibilidade, deve ser dirigida pela diminuição do grau de conectividade entre os módulos do software. A elaboração do modelo baseou-se nos seguinte aspectos: avaliação da relação entre conectividade e estabilidade de sistemas; identificação das métricas dos fatores de impacto no aspecto conectividade de sistema; proposta de MACSOO; implementação da ferramenta *Connecta* que implementa MACSOO, coletando as métricas necessárias para a aplicação do modelo.

Experimentos apresentados neste artigo ilustram a aplicação do modelo proposto e mostram que o seu uso pode levar à construção de software com maior qualidade estrutural e, portanto, de mais fácil manutenção. Fornecem também indícios de que a conectividade é inversamente proporcional à estabilidade e pode ser tomada como indicador da dificuldade de manutenção: quanto maior for a conectividade do software, pior será sua qualidade estrutural, menor será a sua estabilidade e mais difícil será sua manutenção. Entretanto, a quantidade e o porte dos experimentos realizados até então não nos permitem generalizar os resultados, salientando a necessidade de realização de um número maior de experimentos.

Este trabalho aponta para os seguintes trabalhos futuros principais, para os quais a nossa pesquisa está direcionada atualmente: (1) realização de experimentos em larga escala,

em um número considerável de softwares, de tamanhos e propósitos diversos; (2) avaliação de valores a serem considerados satisfatórios para as métricas coletadas, o que é de grande valia no processo decisório na produção de software; (3) proposta de um modelo estatístico de predição de esforço de manutenção de software; (4) realização de estudo similar a partir do modelos do sistema, como diagramas UML, em vez de código fonte, já que é desejável que problemas na construção de softwares sejam identificados já nas fases iniciais.

## Referências

[1] ABREU, F. B.; Carapuça, R. *Object-Oriented Software Engineering: Measuring and Controlling the Development Process.*. In: Proceedings of 4th Int. Conf. of Software Quality. McLean, Virgínia, Estados Unidos: 3-5 Out./1994.

[2] ABREU, F. B.; Ochoa, L.; Goulo, M. *The GOODLY Design Language for MOOD Metrics Collection*. In: ISEG/INESC, ECOOP Workshops, 1997. Portugal: 1997.

[3] BCEL- Byte Code Engineering Library. Apache Software Foundation, 2003. Disponível em http://jakarta.apache.org/bcel/. Acesso em Junho/2005.

[4] BEYER, D.; Lewerentz, C.; Simon, F. *Impact of Inheritance on Metrics for Size, Coupling, and Cohesion in Object-Oriented Systems*. In: Dumke/Abran: New Approaches in Software Measurement, LNCS 2006, Springer Publ. 2001, pp. 1-17.

[5] CHHABRA, J. K.; Aggarwal, K.K. *Measurement of Intra-Class and Inter-Class Weakness for Object-Oriented Software*. In: ITNG'06. 2006, p. 155-160.

[6] CHIDAMBER, S. R.; Kemerer, C.F. *A Metrics Suite for Object Oriented Design*. In: IEEE Transactions on Software Engineering. 1994, p. 476-493.

[7] DALY, J.; Brooks, A.; Miller, J.; Roper, M.; Wood, M. *An Empirical Study Evaluating Depth of Inheritance on the Maintainability of Object-Oriented Software*. In: ISERN. Scotland: 1996.

[8] DEPENDENCY Finder. Disponível em http://depfind.sourceforge.net/. Acesso em Abril/2007.

[9] FENTON, N.; Neil, M. *Software Metrics: Roadmap*. In: Proceedings of the Conference on the Future of Software Engineering. 2000.

[10] FERREIRA, K. A. M.; Bigonha, M. A. S.; Bigonha, R. S.. *Avaliação de Conectividade em Sistemas Orientados por Objetos*. Dissertação de Mestrado. Belo Horizonte: DCC/UFMG, 2006. Disponível em www.dcc.ufmg.br/pos.

[11] FOWLER, Martin. *Refactoring - Improving the Design of Existing Code*. Addison Wesley. 1999.

[12] GAMMA, E.; Helm, R.; Johnson, R.; Vlosside, J. *Padrões de Projeto - Soluções Reutilizáveis de Software Orientado a Objetos*. Porto Alegre: Bookman, 2000. 364p.

[13] GILB, T. *Software Metrics*. U.S.: Winthrop Publishers, 1977. 282p.

[14] GYIMOTHY, Tibor; Ferenc, Rudolf; Siket, Istvan. *Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction*. IEEE Transactions on Software Engineering, vol. 31, no. 10, pp. 897-910, Outubro de 2005

[15] HORDIJK, W.; Wieringa, R. *Surveying the factors that influence maintainability: research design.* In: Proceedings of the 10th European software engineering conference ESEC/FSE-13. Lisboa, Portugal: 2005, v. 30. p. 385-388.

[16] JDEPEND. Disponível em http://www.clarkware.com/software/JDepend.html. Acesso: 04/2007.

[17] UNDERSTAND for Java. Disponível em http://www.scitools.com/uj.html. Acesso em Abril/2007.

[18] KABAILI, H.; Keller, R. K.; Lustman, F.. *Cohesion as Changeability Indicator in Object-Oriented Systems*. IEEE. 2001. pp 39-46.

[19] KRAKATAU Essencial Metrics. Disponível em http://www.powersoftware.com/. Acesso: 05/2006.

[20] LI, P. Luo; Herbsleb, J.; Shaw, M.; Robinson, B. *Experiences and results from initiating field defect prediction and product test prioritization efforts at ABB Inc.* In: Proceeding of the 28th international Conference on Software Engineering ICSE '06. 2006.

[21] LI, Wei., Henry, S. *Maintenance Metrics for Object Oriented Paradigm*. IEEE. In: Proc. First Int'l Software Metrics Symp., pp. 52-60, Maio 1993.

[22] MARTIN, Robert. *OO Design Metrics - An Analysis of Dependencies*. Outubro de 1994.

[23] MEYER, Bertrand. *Object-oriented software construction*. 2. ed. Estados Unidos: Prentice Hall International Series in Computer Science, 1997. 1254 p.

[24] MUNRO, M.J.. *Product Metrics for Automatic Identification of "Bad Smell"Design Problems in Java Source-Code*. Software Metrics, 2005. 11th IEEE International Symposium 19-22 de Setembro de 2005. pp:15 - 15.

[25] MYERS, G. J. *Reliable software through composite design*. N. Y.: Petrocelli/Charter, 1975.

[26] NAGAPPAN, N.; Ball, T.; Zeller, A. *Mining Metrics to Predict Component Failures*. In: ICSE'06. Shangai: 2006. p. 452-461.

[27] OBJECTDETAIL. Disponível em http://www.obsoft.com/Product/ObjDet.html e http://www.obsoft.com/Product/DetailPaper.html. Acesso em Maio/2006.

[28] PARNAS, D. L. *On the criteria to be used in decomposing systems into modules* In: Communications of ACM. 1972. p. 1053-1058.

[29] PAULA FILHO, W. P. *Engenharia de Software - Fundamentos, Métodos e Padrões*. Rio de Janeiro: LTC, 2001. 584 p.

[30] PFLEEGER, S. L. *Software Engineering Theory and Practice*. Upper Saddle River: Prentice-Hall, 1998. 576p.

[31] PURAO, S.; Vaishnavi, V. *Produt Metrics for Object-Oriented Systems*. In: ACM Computing Surveys. v. 35, e 2003. p. 191-221

[32] SANT'ANNA, C. N.; Garcia, A. F.; Chaves, C. F. G.; Lucena, C. J. P.; Staa, A. S. *On the Reuse and Maintenance of Aspect-Oriented Software: an Assessment Framework*. In: 17$^o$ SBES. Anais. Porto Alegre: SBC, 2003.

[33] STAA, Arndt von. *Programação Modular - Desenvolvendo programas complexos de forma organizada e segura*. Rio de Janeiro: Editora Campus, 2000. 690p.

[34] SOMMERVILLE, I. *Engenharia de software*. 6.ed. SP:Addison Wesley, 2003. 592p

[35] XENOS, M.; Stavrinoudis, D.; Zikouli, K.; Christodoulakis, D. *Object-Oriented Metrics - A Survey*. In: Proceedings of the FESMA 2000. Madrid, Espanha: 2000.