UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

LEANDRO DO NASCIMENTO COSTA RODRIGUES

MÉTRICAS DE SOFTWARE:

medição do Sistema de Acompanhamento de Procedimentos Licitatórios do Tribunal de Contas do Estado do Maranhão - SAPLIC

São Luís
2012

LEANDRO DO NASCIMENTO COSTA RODRIGUES

MÉTRICAS DE SOFTWARE:
medição do Sistema de Acompanhamento de Procedimentos Licitatórios do Tribunal de Contas do Estado do Maranhão - SAPLIC

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Orientadora: Profa. Msc. Maria Auxiliadora Freire

São Luis
2012

LEANDRO DO NASCIMENTO COSTA RODRIGUES

MÉTRICAS DE SOFTWARE:

medição do Sistema de Acompanhamento de Procedimentos Licitatórios do Tribunal de Contas do Estado do Maranhão - SAPLIC

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Aprovada em / /

BANCA EXAMINADORA

_____

Profª. Msc. Maria Auxiliadora Freire (Orientadora)

Universidade Federal do Maranhão - UFMA

_____

1º Examinador

Prof. Dr. Mário Antonio Meireles Teixeira

Universidade Federal do Maranhão – UFMA

_____

2º Examinador

Prof. Dr. Samyr Béliche Vale

Universidade Federal do Maranhão - UFMA

A minha família, em especial minha mãe Lucia Cristina, por ter dado muito apoio na minha educação e pelo seu amor incondicional.

AGRADECIMENTOS

Agradecemos a todos aqueles que, direta ou indiretamente, contribuíram para a elaboração desta monografia e, de modo especial:

A Deus, força maior.

Aos meus pais, por terem me ensinado valores fundamentais, como a importância dos estudos, a disciplina, a persistência, que foram essenciais durante o curso, principalmente a minha mãe Lucia Cristina, meu porto seguro, pelo incentivo na execução deste trabalho e disposição que sempre teve em me ajudar.

A minha namorada Flávia Guimarães pelo amor, carinho, paciência nos momentos difíceis e por estar sempre ao meu lado me proporcionando a felicidade.

A MSc. Maria Auxiliadora Freire, pela atenção e dedicação, que ao longo deste trabalho esteve presente, auxiliando nas dificuldades e contribuindo para o sucesso desta jornada.

Ao Tribunal de Contas do Estado do Maranhão, por ter me dado a oportunidade de desenvolver o software SAPLIC.

Aos Professores do Curso de Ciência da Computação da UFMA, pelo conhecimento compartilhado durante esses quase cinco anos.

Aos amigos(as) que tive a oportunidade de conhecer, por estarem comigo desde o inicio da caminhada, pelas vitorias e derrotas, com enorme esforço ate o fim da jornada.

“O que não se mede não se administra, o que não é mensurável, torne-o mensurável”.

Galileu Galilei

RESUMO

Métricas de software e a medição do Sistema de Acompanhamento de Procedimentos Licitatórios do Tribunal de Contas do Estado do Maranhão - SAPLIC. Definem-se os diversos conceitos que envolvem o tema métricas de software, enfatizando finalidade, características e importância das métricas de software junto à engenharia de software e constatação de qualidade. Apresentam-se as várias classificações de métricas de software. Discriminam-se em detalhes as técnicas de métricas de software Linhas de código e suas variações, Ponto de Função e Complexidade Ciclomática. Indica-se abordagem para medição de software em geral e nas organizações. Em seguida são enumeradas ferramentas que automatizam a aplicação de técnicas de métricas de software. Na seção de medição do SAPLIC, coletam-se e analisam-se algumas métricas com o objetivo de avaliar a qualidade do sistema.

Palavras-chaves: Software. Engenharia. Métricas de software. Software. Qualidade.

ABSTRACT

Software metrics and measurement Sistema de Acompanhamento de Procedimentos Licitatórios do Estado do Maranhão - SAPLIC. We define the various concepts surrounding the topic of software metrics, emphasizing purpose, characteristics and importance of software metrics from the software engineering and verification of quality. We present the various classifications of software metrics. Break down in detail the techniques of software metrics LOC and its variations, Function Point and Cyclomatic Complexity. Are indicated approaches to software measurement in general and in organizations. Next are listed tools that automate the application of techniques of software metrics. In the measurement section SAPLIC, to collect and analyze metrics to evaluate the quality of the system.

Keywords: Software. Engineering. Software metrics. Software. Quality.

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÂO

A necessidade de se automatizar os processos tem levado as organizações a um elevado índice de utilização de software, gerando uma crescente demanda pelos serviços de desenvolvimento. Nas grandes organizações, o processo de desenvolvimento e manutenção de softwares, bem como o produto de software construído já se tornou um dos maiores componentes do orçamento, levando-as ao reconhecimento da importância de controlar e analisar os gastos com software. Com o objetivo de agregar qualidade a este processo, aumentar sua produtividade, melhorar a qualidade dos produtos de software e consequentemente reduzir custos e tempo de implementação e manutenção, é que surgiu a Engenharia de Software.

Segundo Sommerville (2003), a Engenharia de Software é uma disciplina da engenharia que se ocupa de todos os aspectos da produção de software, desde os estágios iniciais de especificação do sistema até a manutenção, depois que ele entrou em operação.

Muitas organizações de desenvolvimento de software sabem da importância de ter um produto com qualidade, porém algumas não sabem como medir essa qualidade. A Engenharia de Software fornece como ferramentas possíveis de auxilio, as métricas de software, capazes de gerar dados quantitativos sobre o software e seu processo para serem analisados e utilizados como apoio para melhorias na qualidade do processo de desenvolvimento e produto de software. È um recurso de extrema ajuda para auxiliar a tomada de decisões gerenciais, pois os dados extraídos das métricas possibilitam traçar estimativas de dimensão do software, avaliar produtividade e qualidade, melhorar planejamento do projeto, bem como, avaliar a influência de linguagem de programação e plataforma na qualidade e produtividade do software desenvolvido. Nas palavras de Demarco (1989, p. 3): “Não se pode controlar o que não se pode medir”.

Portanto, o presente trabalho tem como objetivo principal medir o Sistema de Acompanhamento de Procedimentos Licitatórios do Tribunal de Contas do Estado do Maranhão (SAPLIC), mostrando a importância das métricas de software na engenharia de software como um mecanismo de avaliação de qualidade e produtividade.

O trabalho está organizado da seguinte maneira: na seção 2 serão apresentados os conceitos fundamentais acerca de métricas de software, classificação, princípios para garantir a qualidade e validade das métricas de software e algumas técnicas de métricas, como LOC, Pontos de Função e Complexidade Ciclomática. A seção 3 aborda como se dá a medição de

software nas organizações, apresentando o processo de medição, que corresponde a um conjunto de passos que orientam a realização da medição, a metodologia GQM, que auxilia na definição do conjunto de métricas a serem coletadas e, algumas ferramentas utilizadas na medição de software. Na seção 4 é abordada a medição do software SAPLIC, detalhando inicialmente o software, e logo após a definição, coleta e análise das métricas. E para finalizar, no capítulo 5, será feita a conclusão de todo este trabalho, apresentando o resultado da medição.

2 MÉTRICAS DE SOFTWARE

Quando se fala de métricas deve-se ter em mente que se trata de dados quantitativos que irão mostrar em forma de indicadores o estado atual de um determinado processo ou produto. Estes dados quantitativos são de suma importância para o gerenciamento de projetos e avaliação de produtos. Em geral, eles sofrem menos efeitos de subjetividade, permitem comparação direta entre dois ou mais atributos e a precisão pode ser controlada. "Trabalhar com números em vez de avaliações qualitativas reduz a possibilidade de erros de avaliação, que podem levar a decisões errôneas com conseqüências negativas ao projeto". (KOSCIANSKI; SOARES, 2006, p. 69).

Métricas de software são dados quantitativos sobre o software e seu processo de desenvolvimento. Sommerville (2011, p. 466) define métrica de software como sendo: "[...] uma característica de um sistema de software, documentação de sistema ou processo de desenvolvimento que pode ser objetivamente medido."

As métricas, de modo geral, são importantes para entender o comportamento e o funcionamento do produto ou do processo, para controlar os processos e serviços, para prever valores de atributos e para tomar decisões a partir de padrões, metas e critérios.

O princípio da métrica de software é especificar funções de coleta de dados quantitativos referentes à avaliação de qualidade e desempenho do produto ou processo de software, abrangendo inclusive os projetos anteriores, e analisar estes dados com toda a equipe envolvida no projeto. De posse desses dados, a equipe pode avaliar a qualidade do produto antes mesmo dele ser construído e, se necessário, traçar melhorias na qualidade do processo de desenvolvimento e do produto final.

Os dados extraídos das métricas possibilitam traçar estimativas de dimensão do software, avaliar produtividade e qualidade do processo e produto de software, embasar solicitações de novas ferramentas e treinamentos, melhorar planejamento do projeto, bem como, avaliar a influência de linguagem de programação e plataforma na qualidade e produtividade do software desenvolvido.

Portanto, as métricas de software surgem para apoiar a constatação da qualidade, bem como, apoiar o processo de gerenciamento do software. Com a utilização das métricas é possível conseguir melhorias e resultados mais satisfatórios do software, mais segurança para os gerentes de projeto. As classificações a respeito das métricas de software serão demonstradas a seguir.

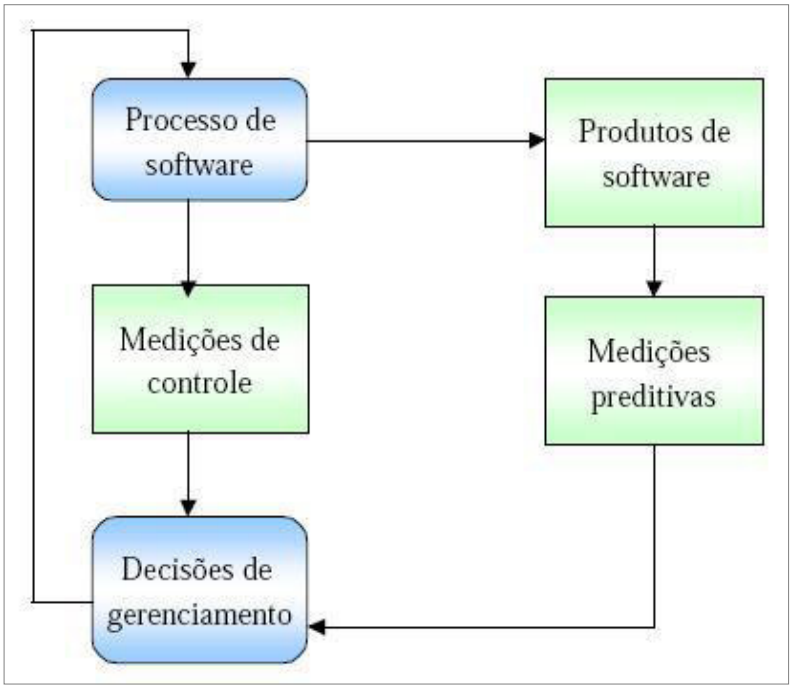
2.1 Categorização das métricas de software

Na literatura existem diferentes classificações a respeito das métricas de software. visto que existem métricas para todo o ciclo de vida do software. Muitos autores possuem suas próprias classificações. Nas seções a seguir, serão listadas as principais classificações das métricas de software.

2.1.1 Métricas de controle e de previsão

De maneira geral, as métricas de software podem ser de dois tipos: métricas de controle, que são associadas a processos de software ou métricas de previsão (preditivas), que são associadas a produtos de software. As métricas de controle, que também são chamadas de métricas de processo, servem para avaliar a melhoria da eficiência de um processo. As métricas de previsão, também chamadas de métricas de produto, ajudam a prever as características dos softwares. Exemplos de métricas de controle são o esforço e o tempo médio para reparar defeitos no produto. Exemplos de métricas de previsão incluem tamanho do código, facilidade de uso de um sistema de software. Tanto a métrica de controle como a preditiva podem influenciar na tomada de decisões em gerenciamento, como ilustra a Figura 2.1, conclui Sommerville (2011).

Figura 2.1 – Métricas de controle e preditiva



Fonte: (SOMMERVILLE, 2011, p. 466).

2.1.2 Métricas diretas e indiretas

As métricas diretas, também chamadas de fundamentais ou básicas, são aquelas que podem ser mensuradas a partir de observação direta dos atributos envolvidos, determinadas habitualmente pela contagem, como por exemplo, custo, esforço, número de linhas de código produzidas, total de defeitos registrados. Uma métrica direta é presumivelmente válida, pois não depende da medida de outros atributos. As métricas indiretas ou derivadas são obtidas a partir de métricas diretas, como por exemplo, a complexidade, eficiência, funcionalidade. Elas são dependentes da validade das métricas diretas.

O custo e o esforço exigidos para se construir o software, o número de linhas de código produzidas e outras medidas diretas são relativamente fáceis de serem reunidas, desde que convenções específicas para medição sejam estabelecidas antecipadamente. Porém, a qualidade e a funcionalidade do software, ou seja, eficiência e capacidade de manutenção são mais difíceis de serem avaliadas e somente podem ser medidas indiretamente.

2.1.3 Métricas de processo

As métricas de processo foram desenvolvidas para auxiliar no gerenciamento e controle dos processos de software, demonstrando as variações existentes nas características de um processo ao longo do tempo para que o mesmo possa ser aperfeiçoado. São dados quantitativos sobre o processo de software que podem ser usadas para avaliar a melhoria da eficiência de um processo. Por exemplo, o esforço e o tempo dedicado aos testes podem ser monitorados.

Três tipos de métricas de processo podem ser coletadas, diz Sommerville (2011, p. 497):

> 1. O tempo necessário para um processo específico ser concluído. Esse pode ser o tempo total dedicado para o processo, o tempo de calendário, o tempo dispendido no processo por engenheiros particulares, e assim por diante.
> 2. Os recursos necessários para um determinado evento. Os recursos podem incluir total empenho em pessoas/dia, custos de viagens ou recursos de computador.
> 3. O número de ocorrências de um determinado evento. Exemplos de eventos que podem ser monitorados incluem o número de defeitos descobertos durante a inspeção de código, o número de mudanças de requisitos solicitados e o número médio de linhas de código modificadas em resposta a uma mudança de requisitos.

Os dois primeiros tipos de métricas podem ser utilizadas para descobrir se houve mudanças relativas a eficiência de um processo. Já as medições do número de eventos ocorridos têm uma relação mais direta sobre a qualidade do software.

O objetivo das métricas classificadas como métricas de processos, portanto, é auxiliar no controle, gerenciamento e evolução dos processos de software através de dados quantitativos possibilitando um nível de qualidade mensurável.

2.1.4 Métricas de produto

As métricas de produto são métricas de previsão usadas para medir atributos internos de um sistema de software. São exemplos de métricas de produto o número de linhas de código e o número de métodos associados a cada classe de objeto. As métricas de produto se dividem em duas classes, diz Sommerville (2011):

a) **Métricas dinâmicas:** são coletadas por meio de medições obtidas a partir de um programa em execução. Essas métricas podem ser coletadas durante o teste de sistema ou após o sistema estar em uso. Elas ajudam a avaliar a eficiência e a confiabilidade de um programa. O número de relatórios de bugs ou o tempo necessário para concluir uma computação são exemplos dessa classe de métricas;
b) **Métricas estáticas:** são coletadas por meio de medições feitas de representações do sistema, como o projeto, o programa ou a documentação. Elas ajudam a avaliar a complexidade, a compreensibilidade e a manutenibilidade de um sistema ou componentes de um sistema de software O tamanho do código e o comprimento médio de identificadores usados são exemplos de métricas estáticas.

2.2 Qualidade das métricas

Um grande número de métricas tem sido propostas para softwares, porém, nem todas são consideradas efetivas, ou seja, alcançam a meta de auxiliar de forma prática os participantes do desenvolvimento de software. Algumas possuem um processo de medição complexo, outras são de difícil análise, e ainda existem aquelas que agregam pouca informação. No entanto, garantir a qualidade das métricas é essencial no processo de

avaliação de um produto ou processo, uma vez que “Todo o trabalho de avaliação é colocado em risco se não for possível garantir a obtenção de dados confiáveis” (KOSCIANSKI; SOARES, 2006, p. 225).

Koscianski e Soares (2006) definem alguns critérios de caráter qualitativo que devem ser observados no momento de elaborar novas métricas, são eles:

a) As métricas devem ter significância: os resultados da medição devem agregar informação útil à avaliação de qualidade do produto ou processo de software. Descartar as métricas que não cumprem esse requisito gera uma importante economia de recursos;
b) As métricas devem ter custo e complexidade de aplicação compatíveis com a avaliação a ser realizada: trata-se da viabilidade da medição e se ela é realmente necessária, se o custo para a medição condiz com a avaliação a ser realizada;
c) As métricas devem ser repetíveis: significa que as medições podem ser feitas repetidamente. È importante ressaltar que se uma medição é feita repetidamente e nenhuma condição é modificada, os resultados devem ser sempre os mesmos. Em alguns casos, isso se torna inviável pelo fato de envolverem condições não controláveis, porém, quando isso acontece, tratamentos estatísticos podem ser utilizados;
d) As métricas devem ser reproduzíveis: diz respeito ao fato do resultado de uma medição ser sempre o mesmo independente do avaliador, ou seja, uma medida que dependa da opinião pessoal de um avaliador ou usuário não é reproduzível;
e) As métricas devem ser objetivas: significa que elas devem ser claras, ter definições que não sejam ambíguas e explicações sobre as mesmas, buscando o melhor entendimento e a facilidade na captura. A influência de opinião de pessoas envolvidas deve ser limitada ou totalmente evitada;
f) As métricas devem ser imparciais: a escolha dos parâmetros do ambiente de execução pode, muitas vezes, favorecer ou não um determinado produto, prejudicando o resultado da avaliação;
g) As métricas devem prover evidência para sua validação: significa que os valores apresentados precisam ser confiáveis.

Ejiogu citado por Pressman (2006, p. 355) também define um conjunto de atributos que devem ser contemplados para que métricas de software sejam consideradas efetivas, são eles:

- Simples e computáveis. Deve ser relativamente fácil aprender como derivar a métrica e seu cálculo não deve exigir esforço ou tempo exagerado.
- Empíricas e intuitivamente persuasivas. A métrica deve satisfazer às noções intuitivas do engenheiro sobre o atributo do produto que está sendo considerado.
- Consistentes e objetivas. A métrica deve produzir sempre resultados que não sejam ambíguos.
- Consistentes no uso de unidades e dimensões. O cálculo matemático da métrica deve usar medidas que não levam a combinações de unidades bizarras.
- Independentes da linguagem de programação: Métricas devem ser baseadas no modelo de análise, modelo de projeto ou na estrutura do programa propriamente dita.
- Mecanismo efetivo para realimentação de alta qualidade. Isto é, a métrica deve levar a um produto final da mais alta qualidade.

2.3 Validação das métricas

Métricas de software são úteis apenas se forem caracterizadas efetivamente e validadas de modo que o seu valor seja comprovado. Os seguintes princípios são representativos de muitos que já foram propostos para a caracterização e validação de métricas:

- Uma métrica deve ter propriedades matemáticas desejáveis, isto é, o valor da métrica deve estar em um intervalo significativo (por exemplo, zero a um, em que zero na realidade significa ausência, um indica o valor máximo e 0,5 representa o "ponto médio"). Também, uma métrica que se propõe a estar em uma escala racional não deve ser composta por componentes que são apenas medidos em uma escala ordinal.
- Quando uma métrica representa uma característica de software que aumenta quando acontecimentos positivos ocorrem ou diminui quando acontecimentos indesejáveis são encontrados, o valor da métrica deve aumentar ou diminuir do mesmo modo;
- Cada métrica deve ser validada empiricamente em uma ampla variedade de contextos antes de ser publicada ou usada para tomar decisões. Uma métrica deve medir o fator de interesse, independentemente de outros fatores. Ela deve se "adequar" a sistemas grandes e funcionar em uma variedade de linguagens de programação e domínios de sistemas. (PRESSMAN, 2006, p. 353).

2.4 Técnicas para métricas de software

Na literatura atual são citados vários exemplos de técnicas de métricas de software. Nesta seção serão abordados três exemplos mais utilizados, que são:

a) Linhas de código (LOC);

b) Ponto de Função;

c) Complexidade Ciclomática.

2.4.1 Linhas de código (LOC)

A métrica de software Linhas de código (do inglês, *Lines of Code - LOC*) é uma métrica estática de produto usada para medir o tamanho de um programa de software pela contagem do número de linhas. É uma medida simples de tamanho, obtida a partir de uma medição direta, que pode ser um indicador para várias características, como prever a quantidade de esforço que será necessária para desenvolver um programa, bem como estimar a produtividade de programação ou manutenção. Esta métrica ainda é utilizada para prever quantidade de erros no código, pois "[...] quanto maior o tamanho do código de um componente, mais complexo e sujeito a erros o componente é." (SOMMERVILLE, 2011, p. 469).

As medidas de tamanho LOC continuam sendo muito utilizadas pelo fato de serem de fácil aplicação e possuírem baixo custo de aplicação. No entanto, estes pontos positivos da utilização de LOC contrastam com um ponto negativo dessa métrica que é a imprecisão: utilizando LOC, não se distingue linhas de código propriamente ditas de linhas em branco ou de comentários. Apesar disso, para grandes volumes de código ela pode ser suficientemente precisa como exemplifica Koscianski e Soares (2006), afirmando que naturalmente, um programa de 4KLOC (milhares de linhas de código) deve ser mais complexo e levar mais tempo para ser desenvolvido que outro de 2KLOC.

Como forma de melhorar a precisão da medida, sugeriu-se o uso da métrica chamada *Source Lines of Code (SLOC)*, onde as linhas em branco ou com comentários não são consideradas. SLOC's são medidas um pouco controversas. Elas podem ser muito úteis nas estimativas de esforço, no entanto, em funcionalidade elas não são: um desenvolvedor qualificado pode ser capaz de desenvolver a mesma funcionalidade com menos código, então um programa com menos SLOC pode apresentar mais funcionalidade do que outro programa similar. Além disso, SLOC é particularmente ineficaz para comparar os programas escritos em linguagens diferentes. Outro problema cada vez mais comum em comparar métricas SLOC é a diferença entre o código gerado automaticamente e escrito à mão: ferramentas de software costumam ter a capacidade de auto-gerar enormes quantidades de código que, se fossem feitos por um desenvolvedor, poderia ser menor.

As métricas de software baseadas na contagem de linhas de código, portanto, é o método mais simples e de baixo custo de aplicação para tamanho de um programa, porém também é o mais impreciso, salvo engano quando for utilizado para grandes volumes de código. Esta imprecisão pode ser atenuada com a utilização do SLOC, que apesar de algumas desvantagens, é uma métrica de caráter quantitativo para produtos de software, e pode ajudar a revelar dados qualitativos como complexidade do software.

2.4.2 Ponto de Função (PF)

Pontos de função (*Function Point - FP*) é uma medida que estima o tamanho de um software baseando-se nas funcionalidades. O método para se chegar a esta medida é chamado de Análise de Pontos de Função - APF (do inglês, *Function Point Analysis - FPA*).

A análise de pontos de função, portanto, é uma métrica de software que tem por objetivo obter uma medida funcional de tamanho do software, ou seja, medição do tamanho do software considerando-se apenas a funcionalidade solicitada e recebida pelos respectivos usuários. Ela focaliza a perspectiva de como os usuários “enxergam” os resultados que um sistema produz. Nesse sentido, uma medida funcional de tamanho é uma medida externa, pois mede somente aquilo que é percebido pelos usuários do produto de software, independentemente da forma de implementação escolhida.

Koscianski e Soares (2006, p. 230) definem APF da seguinte maneira:

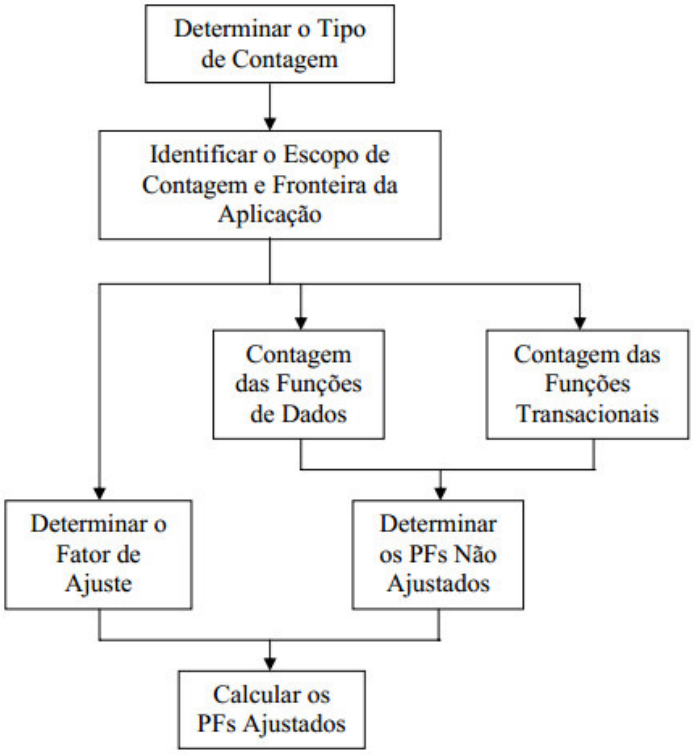
> A métrica análise por pontos de função (ou simplesmente pontos de função) foi desenvolvida na década de 1970, por Albrecht [1979], como uma forma de medir software considerando as funcionalidades criadas. A medida pode ser aplicada antes de o código ser escrito, baseando-se na descrição arquitetural do projeto. Além disso, PF é independente da tecnologia usada no desenvolvimento: pode ser empregada em ambiente mainframe com a linguagem COBOL, cliente-servidor com C++ ou Web com PHP. Em cada caso o que se observa é um custo de desenvolvimento diferente para uma mesma contagem de PF.

Desse modo, percebe-se que a medida PF é independente da linguagem de programação ou da tecnologia que será usada para implementação do software. Além disso, ela é imediatamente aplicável ao longo da vida de um projeto de desenvolvimento, desde a definição inicial dos requisitos, permitindo estimar o esforço e o cronograma de implementação, até a fase de plena utilização operacional, substituindo LOC para estimar esforço de manutenção e medindo a funcionalidade recebida pelo usuário após o projeto de software, de forma a verificar seu tamanho e custo.

A contagem dos PFs é regulamentada pelo *International Function Point Users Group (IFPUG),* organização internacional sem fins lucrativos sediada nos Estados Unidos. O

IFPUG publica o *Counting Practices Manual* (Manual de Práticas de Contagem), que estabelece os padrões para o cálculo dos pontos de função. Segundo o *Function Point Counting Pratices Manual – Release 4.1.1*, publicado pelo IFPUG em 1999, o procedimento para contagem de pontos de função compreende sete passos como mostra a Figura 2.2.

Figura 2.2 - Procedimento de Contagem de Pontos de Função



Fonte: (HAZAN, 2001).

O primeiro passo a ser seguido para a contagem de PF de um projeto de software é determinar o tipo de contagem. Segundo o IFPUG (1999), são possíveis três tipos de contagem: contagem de projeto de desenvolvimento, contagem de projeto de melhoria (manutenção) e contagem de aplicação (produção). Logo após deve-se identificar e definir o escopo de contagem e fronteira da aplicação. A identificação do escopo visa definir a abrangência da contagem estipulando se a contagem vai se referir a um ou mais sistemas ou a apenas parte de um sistema e a fronteira da aplicação separa a aplicação que esta sendo contada das aplicações externas indicando o limite entre a aplicação e os demais usuários.

A contagem das funções de dados representa a funcionalidade provida ao usuário por meio de dados internos e externos, que são os arquivos lógicos internos ou os arquivos de interface externos. As funções de transações, por sua vez, são provenientes de entradas,

consultas e saídas externas. Koscianski e Soares (2006 p. 231) definem cinco fatores para a contagem do número de PFs:

- Arquivos lógicos internos (ALI): grupo de dados persistentes, relacionados logicamente, mantidos pelo sistema e alimentados por entradas externas ou valores calculados. São relativos às funcionalidades do software, como, por exemplo, cadastro de clientes e alteração de produtos. Não são ALI, por exemplo, arquivos temporários e de classificação.
- Arquivos de interface externos (AIE): grupo de dados persistentes, relacionados logicamente, porém utilizados apenas para consultas do sistema. Os dados são mantidos e/ou alimentados por outros programas.
- Entradas externas (EE): um processo lógico em que dados são introduzidos no sistema. Os dados podem ser informações de controle ou de negócios. As entradas externas representam o fluxo de informações que adentra o sistema. Exemplos de EE são a exclusão, alteração e inclusão de registros.
- Saídas externas (SE): um processo lógico em que dados são enviados ao exterior das fronteiras do sistema – por exemplo, na emissão de relatórios ou mesmo na apresentação de dados na tela. Tais dados são computados a partir de arquivos lógicos internos e arquivos de interface externos.
- Consulta externa (CE): um processo lógico que envolve um par consulta-resposta. Os dados são recuperados exclusivamente de arquivos internos e interfaces externas (nada é computado). Nenhum Arquivo Lógico Interno é alterado nesse processo. São exemplos de CE as consultas ao cadastro de clientes.

Uma vez identificados e coletados esses dados, esses cinco fatores devem ter um valor de complexidade classificado como baixa, média ou alta. Cada Arquivo Lógico Interno deve ser classificado de acordo com sua complexidade funcional relativa, que é baseada no número de Registros Lógicos(RL) e no número de Itens de Dados (ID) do arquivo (IFPUG,1999). Um Registro Lógico é um subgrupo de dados reconhecido pelo usuário dentro de um ALI, enquanto que um item de dados representa um segmento de um ALI que possui um significado único, não repetitivo e pode ser reconhecido pelo usuário. Conforme o número de itens de dados referenciados e o número de registros lógicos encontrados, a complexidade de um ALI pode ser classificada em baixa, média e alta de acordo com o Quadro 2.1. Como um AIE de uma aplicação sempre será contado como um ALI na aplicação de origem, este quadro também vale para medir a complexidade de um AIE.

Quadro 2.1 - Complexidade para Arquivo Lógico Interno – ALI e Arquivo de Interface Externa - AIE

	1 a 19 itens de dados referenciados	**20 a 50 itens de dados referenciados**	**51 ou mais itens de dados referenciados**
1 Registros Lógicos referenciados	Baixa	Baixa	Média
2 a 5 Registros Lógicos referenciados	Baixa	Média	Alta
6 ou mais Registros Lógicos referenciados	Média	Alta	Alta

Fonte: (IFPUG, 1999)

Cada EE, SE e CE deve ser classificada conforme sua complexidade funcional que é baseada no número de Arquivos Referenciados (ALI e AIE) e no número de itens de dados (ID). A complexidade funcional de uma EE pode ser classificada em baixa, média e alta conforme Quadro 2.2.

Quadro 2.2 - Complexidade para Entrada Externa – EE

	1 a 4 itens de dados referenciados	**5 a 15 itens de dados referenciados**	**16 ou mais itens de dados referenciados**
0 ou 1 Arquivos referenciados	Baixa	Baixa	Média
2 Arquivos referenciados	Baixa	Média	Alta
3 ou mais Arquivos referenciados	Média	Alta	Alta

Fonte: (IFPUG, 1999)

A complexidade funcional de uma SE pode ser classificada em baixa, média e alta conforme Quadro 2.3.

Quadro 2.3 - Complexidade para Saída Externa - SE

	1 a 5 itens de dados referenciados	**6 a 19 itens de dados referenciados**	**20 ou mais itens de dados referenciados**
0 ou 1 Arquivos referenciados	Baixa	Baixa	Média
2 ou 3 Arquivos referenciados	Baixa	Média	Alta
4 ou mais Arquivos referenciados	Média	Alta	Alta

Fonte: (IFPUG, 1999)

A complexidade funcional de uma CE pode ser classificada em baixa, média e alta conforme Quadro 2.4.

Quadro 2.4 - Complexidade para Consulta Externa – CE

	1 a 5 itens de dados referenciados	**6 a 19 itens de dados referenciados**	**20 ou mais itens de dados referenciados**
0 ou 1 Arquivos referenciados	Baixa	Baixa	Média
2 ou 3 Arquivos referenciados	Baixa	Média	Alta
4 ou mais Arquivos referenciados	Média	Alta	Alta

Fonte: (IFPUG, 1999)

Após definir o tipo de contagem, a fronteira da aplicação e reconhecer as funções de dados e de transação, pode-se calcular os pontos de função não ajustados, ou brutos,

multiplicando-se o total de ALI, AIE, EE, SE, e CE pela respectiva complexidade conforme o Quadro 2.5.

Quadro 2.5 - Contribuição das Funções de dados e de transação na contagem dos PFs não ajustados

Funções	**Complexidade**		
	BAIXA	MÉDIA	ALTA
ALI	7	10	15
AIE	5	7	10
EE	3	4	6
SE	4	5	7
CE	3	4	6

Fonte: (IFPUG, 1999)

As contribuições dos componentes somadas resultam em uma quantidade de pontos de função não ajustadas. Isso se deve ao fato de que a técnica de Análise por Pontos de Função considera que outros fatores afetam o tamanho funcional de um sistema, por exemplo, no cálculo dos PF não ajustados não é levado em conta a tecnologia usada nem os requisitos não funcionais. Por este motivo é calculado o valor do fator de ajuste. O fator de ajuste dos pontos de função que, serão o resultado final da medição, é obtido a partir da aplicação de 14 características denominadas Características Gerais dos Sistemas (CGS) que podem receber um valor de 0 a 5, representando nenhuma influência até influência forte ou total. As CGS podem ser vistas no Quadro 2.6 e o fator de ajuste pode ser calculado da seguinte maneira:

"FA = 0,65 + 0,01 x (n1 + n2 + ... +n14)

onde cada n_i representa um dos 14 níveis de influência." (KOSCIANSKI; SOARES, p. 232).

Quadro 2.6 – Características Gerais dos Sistemas (CGS)

Número	Característica
1	Comunicação de dados
2	Funções distribuídas
3	Desempenho
4	Configuração do equipamento
5	Volume de transações
6	Entrada de dados on-line
7	Interface com o usuário
8	Atualização on-line
9	Processamento complexo
10	Reusabilidade
11	Facilidade de implantação
12	Facilidade operacional
13	Múltiplos locais
14	Flexibilidade a mudanças

Fonte: (KOSCIANSKI; SOARES, p. 232).

Devido a sua subjetividade, a utilização do fator de ajuste tornou-se opcional ao final do ano de 2002, como medida para aceitação dos pontos de função do IFPUG como um método padrão de medida funcional, pois várias características estão relacionadas com requisitos não funcionais da aplicação (VAZQUEZ, 2005). Muitas organizações desconsideram o fator de ajuste e usam apenas a medição dos pontos de função não ajustados.

Depois de realizado o ajuste do cálculo, chega-se ao número de Pontos de Função Ajustados (PFA):

"PFA = FA x PF" (KOSCIANSKI; SOARES, p. 232).

Esses dados finais representam à complexidade funcional do software onde, pode-se estimar a partir daí o esforço para produção de 1 PF, ou ainda, o custo para produção de 1PF.

Os pontos de função são utilizados como fator normalizador do tamanho do software, permitindo o estabelecimento de métricas tais como produtividade (pontos de função produzidos por pessoa-mês), taxa de entrega (homens-hora para a produção de um Ponto de Função), densidade de defeitos (defeitos encontrados por Ponto de Função) e outras.

Nas organizações, a APF permite determinar o tamanho de pacotes de softwares adquiridos, estimar custos e recursos envolvidos em projetos de desenvolvimento e manutenção de software e ainda serve como fator de normalização para comparação de software.

Por fim pode-se classificar a métrica PF tanto como uma métrica de produto de software quanto como uma métrica de processo de software, uma vez que essa métrica busca medir características internas do software, bem como ser utilizada para o melhoramento de processos.

2.4.3 Complexidade ciclomática

A métrica de software denominada complexidade ciclomática é uma métrica estática de produto que estima a complexidade de um programa. Proposta inicialmente por Thomas McCabe em 1976, ela visa analisar a estrutura do código-fonte fornecendo um valor numérico que o caracteriza. (KOSCIANSKI; SOARES, 2006).

Medir a complexidade ciclomática nada mais é do que medir a quantidade de caminhos de execução diferentes de um dado programa. Um caminho de execução pode ser entendido como qualquer caminho ao longo do programa que introduz pelo menos um novo

conjunto de comandos de processamento ou uma nova condição. Dessa maneira, esta métrica funciona também como um indício da complexidade do programa e, com base nessa informação, pode-se prever o esforço para implementá-lo ou modificá-lo.

A alta complexidade de um algoritmo devido a uma quantidade grande de caminhos existentes no código torna-o, segundo Koscianski e Soares (2006), mais difícil de compreender e manter, pois os testes realizados nele tornam-se mais complexos, pois é preciso garantir que cada caminho de execução seja executado e que as condições apropriadas sejam obtidas.

Pressman (2006, p. 321), afirma que a complexidade ciclomática tem fundamentação na teoria dos grafos, cada nó representa um trecho de código ou um comando, e pode ser calculada por uma de três maneiras:

1. O número de regiões correspondente à complexidade ciclomática.
2. A complexidade ciclomática, V(G), para um grafo de fluxo, G, é definida como V(G) = E − N +2, em que E é o número de arestas do grafo de fluxo e N é o número de nós do grafo de fluxo.
3. A complexidade ciclomática, V(G), para um grafo de fluxo, G, é também definida como V(G) = P + 1, em que P é o número de nós predicados (*Nó predicado* é o que tem duas ou mais arestas saindo dele) contidos no grafo de fluxo G.

Existe ainda uma maneira mais simples de se calcular o v(G) que, consiste em contar o número de comandos de desvios condicionais ou incondicionais existentes no código e adicionar 1 a esse valor (Kosciasnki e Soares, 2006). Em um trecho de código que contêm dois comandos condicionais if, por exemplo, o valor da complexidade ciclomática v(G) seria 3.

De uma forma geral, "[...] a complexidade ciclomática deve ser mantida em um valor máximo de 10, além do qual se deve particionar o código utilizando subrotinas." (WASTON; MACCABE apud KOSCIANSKI; SOARES, 2006, p. 236).

A complexidade ciclomática, portanto, é uma métrica de software que auxilia nos testes de software prevendo módulos propensos a erros, e também na avaliação da sua complexidade, para que os desenvolvedores possam estabelecer, por exemplo, um nível máximo de complexidade para os módulos do software em desenvolvimento.

3 MEDIÇÃO DE SOFTWARE NAS ORGANIZAÇÕES

A medição de software começou a ser praticada na década de 70, inicialmente apenas para medir o numero de linhas de código dos softwares produzidos. Na década de 80, começaram a ser utilizadas outras medidas relacionadas à fase de desenvolvimento, porém as organizações não reconheciam a importância dessas medidas, tratavam apenas como mais uma coisa a ser feita e isto resultava em medições inúteis não alinhadas as necessidades das organizações. Nos anos 90, impulsionados por aplicações bem sucedidas, modelos para o processo de medição foram desenvolvidos baseados na melhoria de processos e nos princípios da qualidade total, fornecendo as diretrizes e a infraestrutura básicas para definir, coletar, validar e analisar medidas (BASS *et al.*, 1999; FENTON; NEIL, 1999). Atualmente, com o amadurecimento da engenharia de software, a medição assumiu um papel fundamental na compreensão e controle de práticas e produtos do desenvolvimento de software. Ela já é considerada uma pratica básica da Engenharia de Software, sendo evidenciada por sua inclusão nos requisitos do nível 2 do CMMI(*Capability Maturity Model Integration*) da SEI, na área de processo *Medição e Analise,* que busca garantir que não ocorram desvios ou variações durante o desenvolvimento do projeto.

A implantação de um programa de medições de software em uma organização requer um esforço específico. Estas medições devem ser eficientes, devem ser percebidas como uma prática fundamental para a sobrevivência e crescimento organizacional e ainda devem apoiar a tomada de decisões.

Muitas organizações coletam dados sobre os seus softwares, porém não utilizam estas medições sistematicamente para comparar produtos e processos de software ou avaliar o impacto das mudanças nos processos e ferramentas de software. Elas simplesmente despendem tempo e esforço para acumular dados inúteis. Segundo (SOMMERVILLE, 2011, p. 468), esta é uma tarefa difícil devido a algumas razões:

1. É impossível quantificar o retorno sobre o investimento da introdução de um programa de métricas em organizações. [...].
2. Não existe um padrão para métricas de software ou processos padronizados para medição e análise. [...].
3. Em muitas empresas, os processos de software não são padronizados e são mal definidos e mal controlados. Além disso, existe muita variabilidade em processos da mesma empresa para que as medições sejam usadas de forma significativa.
4. A introdução da medição acrescenta overhead aos processos.[...].

Além disso, a iniciativa de implementar um programa de medição eficiente e eficaz na organização, depara-se muitas vezes com problemas do tipo:

a) Da extensa lista de métricas, quais são as adequadas para o meu caso?

b) Quais são as ferramentas e os processos necessários e disponíveis para realizar as medições desejadas?
c) Quais são os recursos financeiros e humanos necessários para implementar e manter o programa de medição?
d) Como analisar os dados coletados?
e) Como apresentar as informações relacionadas aos dados coletados?
f) Como atuar sobre ferramentas, métodos e processos no sentido de automatizar a medição?

Um sistema de medição em uma organização deve ser coerente com a sua realidade e deve suprir as diferentes necessidades de informação da organização. Novas métricas poderão ser criadas no sentido de atender novas demandas de informação. Entretanto, as medições devem ser realizadas de forma a não interferir no processo de produção e devem ser registradas em um banco de dados, permitindo diferentes agregações de dados e possibilitando consultas *ad hoc*.

De posse das informações providas pela medição, as organizações poderão elaborar planos mais realistas para projetos e, em um segundo momento, comparar o desempenho do projeto com seu plano inicial. Além disso, estas informações irão orientar as decisões de melhoria e os investimentos e auxiliar a prever se o projeto em andamento irá alcançar seu objetivo com êxito.

Para tanto, é preciso controlar e supervisionar o processo produtivo, visando manter a produtividade nos níveis previstos, remover o quanto antes defeitos introduzidos no produto, reduzindo ou eliminando o esforço de retrabalho, consequentemente mantendo o orçamento sob controle.

Portanto, além de apoiar veementemente a tomada de decisão, as medições aceleram o aprendizado organizacional, uma vez que a análise dos dados coletados em um projeto provê uma base para os projetos futuros, e melhoram o desempenho da organização, permitindo otimizarem seus processos técnicos e de negócio quando necessário.

Segundo Barcellos (2009, p. 12):

> Somente quando as informações obtidas na análise dos dados coletados são utilizadas para direcionar as ações necessárias às organizações e seus projetos é que o objetivo fundamental da medição é alcançado e percebido pelas organizações, fator que contribui para a real implantação de um programa de medição eficiente.

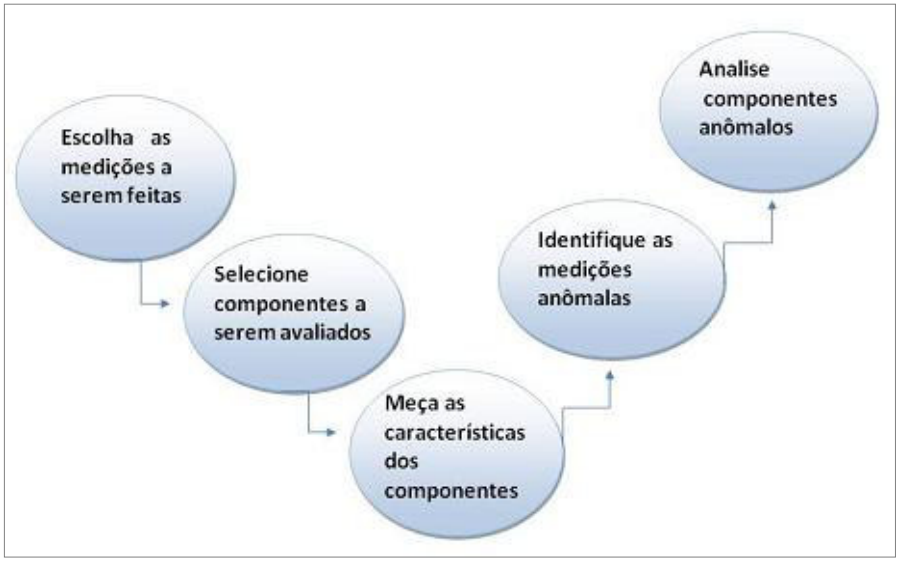
3.1 Processo de medição de software

O processo de medição corresponde a um conjunto de passos que orientam a realização da medição. Segundo Wang e Li (2005) um processo de medição eficiente é fator crítico ao sucesso da medição na organização, pois é ele que direciona as atividades a serem realizadas para que com os resultados da analise dos dados coletados seja possível a identificação de tendências e antecipação aos problemas, a fim de prover melhor controle dos custos, redução dos riscos, melhoria da qualidade e, consequentemente, alcance dos objetivos técnicos e de negocio.

Sommerville (2011, p. 470) afirma que em um processo de medição, cada componente de sistema pode ser analisado separadamente, usando uma variedade de métricas. De forma bem interessante, ele propôs um processo de medição de componentes de software, que é ilustrado pela Figura 3.1, e é especificado nas atividades descritas abaixo:

1. Escolher medições a serem efetuadas: As questões que a medição está destinada a responder devem ser formuladas e as medições necessárias para responder a essas questões devem ser definidas. As medições que não são diretamente relevantes para essas questões não necessitam ser coletadas. O paradigma Meta-Questão-Métrica (GQM, do inglês Goal-Question-Metric) de Basili (BASILI E ROMBACH, 1994) é uma boa abordagem a ser utilizada quando se deseja definir que medidas devem ser coletadas.
2. Selecionar componentes a serem avaliados: Pode não ser necessário ou desejável avaliar valores de métricas para todos os componentes de um sistema de software. Uma seleção representativa de componentes pode ser escolhida para a medição. De outro modo, podem ser avaliados somente os componentes cruciais, que estão em uso constantemente. A qualidade desses componentes é mais importante do que a qualidade de componentes raramente usados.
3. Medir características de componentes: Os componentes selecionados são medidos e os valores e as métricas associados são computados. Normalmente, isso envolve o processamento da representação de componente (código, projeto, documentação) utilizando uma ferramenta automatizada de coleta de dados. Essa ferramenta pode ser especialmente escrita ou pode ser um recurso de ferramenta de projeto que já esteja em uso.
4. Identificar medições anômalas: Após terem sido realizadas as medições de componentes, elas devem ser comparadas entre si e com as medições precedentes que tenham sido registradas em um banco de dados de medições. É preciso procurar valores anormalmente altos ou baixos para cada métrica, uma vez que isso sugere que pode haver problemas com o componente que apresenta estes valores.
5. Analisar componentes anômalos: Ao identificar os componentes que têm valores anômalos para sua métrica escolhida, os mesmo devem ser examinados para decidir se esses valores de métricas anômalos significam que a qualidade do componente esta comprometida. Um valor anômalo de métrica de complexidade (por exemplo) não significa, necessariamente, um componente de má qualidade. Pode haver alguma outra razão para o alto valor, por isso pode não significar que existam problemas de qualidade de componente.

Figura 3.1 - Esboço de processo genérico de medição de produto de software



Fonte: (SOMMERVILLE, 2011, p. 470)

Outros autores também propuseram uma forma de medição de produto de software. Na visão de Roche (1994), um processo de medição pode ser caracterizado por cinco atividades:

a) **Formulação:** Derivação de medidas e métricas de software adequadas para a representação do software que está sendo considerado;
b) **Coleta:** Mecanismo usado na obtenção dos dados necessários para derivar as métricas formuladas;
c) **Análise:** Cálculo de métricas por meio de ferramentas matemáticas;
d) **Interpretação:** Avaliação das métricas de forma a se obter profundidade nas características do sistema e respaldo para a tomada de decisão;
e) **Realimentação:** Recomendações derivadas da interpretação das métricas de produto, transmitidas à equipe de software.

Apesar das propostas acima possuírem diferenças entre si, nota-se que as definições propostas consistem basicamente de quatro etapas: definição das medidas, coleta das medidas, análise das medidas coletadas e utilização dos resultados da analise em ações.

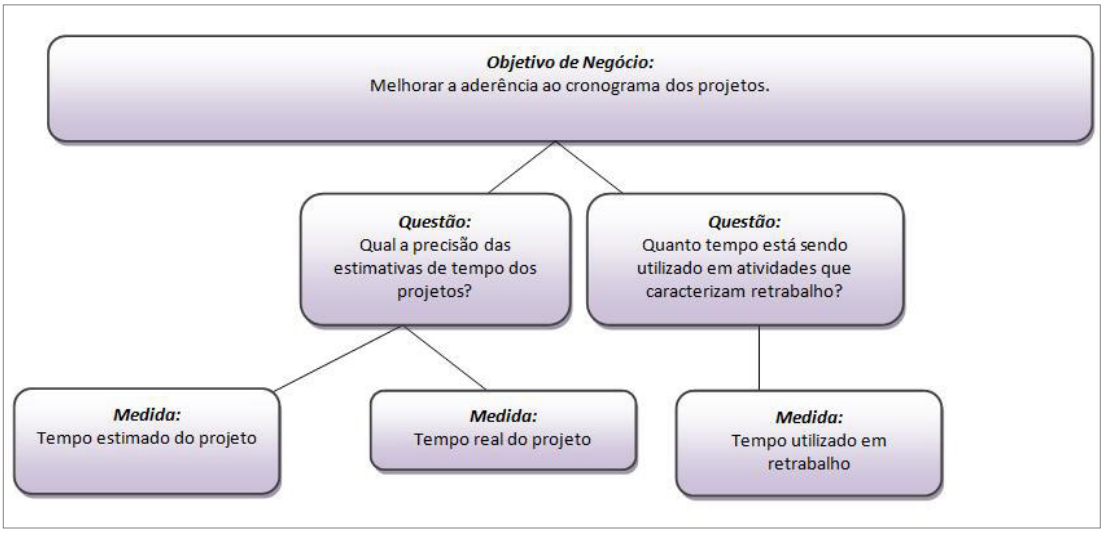
3.2 Abordagem para medição de software

Abordagem para medição de software apóia a definição do processo de medição, bem como sua execução. Uma das abordagens mais conhecidas é o GQM, que será discutido a seguir.

3.2.1 O paradigma GQM (*Goal-Question-Metric*)

Saber quais as medidas adequadas sobre o processo que devem ser coletadas é uma dificuldade corriqueira do processo de medição. O paradigma GQM (Metas-questões-métricas, do inglês *Goal-Question-Metric*), proposto por Basile e Weiss (apud KOSCIANSKI; SOARES, 2006, p. 224) surge com o objetivo de auxiliar na definição desse conjunto de métricas a serem coletadas, identificando e selecionando medidas adequadas à avaliação do alcance dos objetivos organizacionais. È baseado na idéia de que deve existir uma necessidade clara associada a cada métrica, que para cada meta estabelecida, é possível determinar questões cujas respostas estão associadas a medidas. A Figura 3.2 apresenta um exemplo para a relação de um objetivo de negócio, suas questões e medidas associadas.

Figura 3.2 - Exemplo do uso do GQM



Fonte: (SOMMERVILLE, 2003).

O paradigma GQM é uma maneira organizada de tratar o planejamento do trabalho de medição e implantação de melhorias. Ele responde questões que surgem no momento, como o porquê de introduzir melhorias, quais informações são necessárias para identificar e avaliar as melhorias e quais métricas fornecerá estas informações.

De acordo com Sommerville (2011, p. 498).

> A vantagem de usar a abordagem GQM na melhoria de processos é que ela separa os interesses organizacionais (as metas) dos interesses específicos de processos (as questões). Ela fornece uma base para decidir quais dados devem ser coletados e sugere que os dados coletados devem ser analisados de diferentes maneiras, dependendo da questão que devem responder.

O método GQM organiza o planejamento de uma medição de software em etapas, dizem Koscianski e Soares (2006). A cada etapa deve-se definir um elemento conforme descrito a seguir:

a) Objetivos: são estabelecidos de acordo com as necessidades da pessoa, grupo, organização ou sistema. Os objetivos de medição devem ser fixados em função dos requisitos de software;
b) Questões: são as perguntas que se espera responder com o estudo;
c) Categorias: particionam o conjunto de dados obtidos. As perguntas criadas no passo anterior podem trazer à tona diferentes tipos de informação;
d) Formulários: conduzem o trabalho dos avaliadores estabelecendo padrões de formatos facilitando a análise das informações;
e) Coleta: recolhe e analisa os dados em tempo real pra reabastecer os projetos para que se realizem as ações corretivas;
f) Validação: nessa etapa os dados são analisados e são feitas operações para ver se os mesmos estão em conformidade com os objetivos e, além disso, são validados.

Através dessas etapas espera-se organizar o processo de medição de software de um modo planejado e que possa ser seguido pela organização.

Portanto, uma maneira organizada de tratar o planejamento do trabalho de medição é o método GQM. Ele apóia a identificação e seleção de medidas adequadas à avaliação do alcance dos objetivos organizacionais e assegura que todos os defeitos sejam corrigidos antes do software ser liberado para o uso.

3.3 Ferramentas para medição

Realizar medições em produtos de software é uma tarefa que demanda da análise de códigos fontes, modelos e diagramas UML, complexidade dos algoritmos implementados e contagens relativas à programação do software em geral. Diante disso, as ferramentas para a automatização de métricas de software surgem como uma maneira de tornar o processo de

medição menos oneroso e para dar aos programadores um *feedback* sobre o software desenvolvido.

As ferramentas de captura de métricas de software abrangem um conjunto de métricas que são implementadas quer como aplicações isoladas ou (mais comumente) como funcionalidade que existe dentro de ferramentas para análise e projeto, codificação ou teste. Elas apóiam os engenheiros de software na avaliação dos produtos de trabalho produzidos durante a modelagem de análise e projeto, geração de código-fonte e teste.

Durante os estudos realizados, foram encontradas, após pesquisas na web, muitas ferramentas de métricas de software, de diversos tipos, para várias linguagens. Abaixo são relacionadas algumas ferramentas pesquisadas juntamente com uma breve descrição:

a) **Eclipse Metrics Plugin -** ferramenta *open source* que apresenta métricas para programas em Java e funciona como um plugin para a plataforma de desenvolvimento Eclipse. Este plugin extrai métricas de tamanho, complexidade e gera gráficos de dependências entre as classes utilizadas no projeto analisado. Além disso, as métricas podem ser exportadas para um arquivo XML a ser processado com XSL em qualquer tipo de relatório desejado;

b) **Code Analyzer -** ferramenta construída em Java que fornecer métricas básicas sobre o tamanho dos códigos fontes analisados através de múltiplas árvores de um projeto. É uma ferramenta *open Source* voltada para softwares construídos nas linguagens C, C++, Java, Assembly e Html. Apresenta resultados gráficos e gera relatórios com os resumos das medições realizadas em um projeto;

c) **PMD For Eclipse -** ferramenta que analisa código Java e procura por potenciais problemas como: Possíveis erros - try / catch / finally / vazios; Código morto – variáveis locais, parâmetros e métodos privados não utilizados; Código não otimizado - Desperdício de String / StringBuffer; Expressões demasiadamente complicadas – declarações if desnecessárias, loops que poderiam ser feitos com while; Duplicação de código – códigos copiados significam erros copiados.

d) **Jira and Bugzila –** ferramenta open source voltada para gerentes de projetos, gerentes de produtos e engenheiros de software. Fornece suporte ao gerenciamento de métricas voltadas a processos de software**;**

e) **Jmetric -** ferramenta funcional de análise de código java distribuída sob os termos da General Public License GPL. Coleta informações dos arquivos de origem java e compila um modelo de métricas de qualidade e tamanho. Este modelo é então preenchida com métricas de informações, tais como linhas de código, número de declarações e complexidade ciclomática. O JMetric gera tabelas e gráficos com os resultados e ainda pode exportar relatórios com estas informações. A partir do modelo outras métricas podem ser calculados, incluindo o número de classes, pacotes, métodos etc;

f) **CMT++ -** ferramenta proprietária de medida de complexidade para as linguagens de programação C e C++. Tem como objetivo fornecer métricas de tamanho e complexidade sobre os códigos fontes analisados;

g) **LDRA testbed –** ferramenta proprietária voltada para testes de códigos fonte, permitindo testes de validação e verificação bem como, análises estáticas que fornecem métricas de tamanho e complexidade do software. É voltada para softwares construídos com as seguintes linguagens de programação: C, C++, Ada, Java, Visual Basic, Cobol, Coral, Fortran e Pascal;

h) **Jdepend Plugin para Eclipse -** ferramenta open source que fornece métricas orientadas a objeto para as dependências de pacotes, que dão uma indicação da resistência de uma base de código. Em outras palavras, JDepend mede a robustez (ou inversamente, a fragilidade) de uma arquitectura. Esta ferramenta percorre os diretórios de arquivos de classe Java e gera métricas de projeto de qualidade para cada pacote Java. Ela permite medir automaticamente a qualidade de um projeto em termos de sua extensibilidade, reusabilidade e manutenção para gerenciar efetivamente as dependências do pacote, ou seja, permite analisar características como dependência, coesão e acoplamento;

i) **Cyvis -** ferramenta open source desenvolvida em Java para analisar e gerar métricas de softwares também desenvolvidos em Java. O principal diferencial da ferramenta Cyvis é que ela coleta métricas de tamanho e complexidade das classes (.class) em Java e não dos códigos fontes (.java) como outras ferramentas. Apresenta as medições através de gráficos gerados e também é capaz de gerar relatórios em Html e Xml;

j) **OSPC -** ferramenta proprietária que tem o objetivo de auxiliar o processo de garantia de qualidade de produtos de software. Para tanto ela realiza analises

no código fonte fornecendo métricas de complexidade, bem como, apontando possíveis problemas de compilação e fornecendo informações sobre a manutenibilidade do produto. É voltada para softwares construídos nas linguagens de programação C, C++ e Java;

k) **Jmove -** ferramenta open source desenvolvida em Java que possibilita o gerenciamento de projetos de softwares escritos em Java através da extração de métricas de tamanho, complexidade e análise de dependências. É voltada também para métricas de processos de software. Jmove permite a exibição dos resultados das analises em gráficos e gera relatórios pertinentes a medições do projeto;

l) **FindBugs Plugin para Eclipse -** ferramenta open source, utilizada juntamente com a plataforma de desenvolvimento Eclipse para encontrar erros em programas Java. Ela procura por casos de "padrões de bugs" - instâncias de código que são susceptíveis de ser erros. Utilizando-se de análise estática para analisar bytecodes (arquivo binário Java), ela gera uma listagem, trazendo um resumo dos números de bugs encontrados e exibe as advertências e as fontes relevantes.

m) **C and C++ Code Counter (CCCC) -** ferramenta open source implementada em C++ que analisa o código fonte de softwares construídos em C, C++ e Java e gera relatórios contendo métricas de tamanho e complexidade a respeito dos mesmos. Não possui interface gráfica, as métricas são extraídas pela ferramenta através de linhas de comando e seus resultados gerados em Html, Txt e Xml.

O Quadro 3.1 apresenta todas as ferramentas pesquisadas e suas características mais relevantes para a medição do SAPLIC, que se dará na próxima seção.Pode-se observar que as ferramentas pesquisadas são voltadas para uma grande variedade de linguagens de programação, predominando as linguagens C, C++ e Java. Outro fator a se observar é que algumas das ferramentas pesquisadas são proprietárias, ou seja, não possuem licenças livres. Diante disso, alguns pontos negativos das mesmas são percebidos, como: a impossibilidade de conseguir versões *demos* ou *trials* para realizar a medição e o alto custo envolvido em adquirir as ferramentas. Notou-se também que somente quatro ferramentas são implementadas como funcionalidade dentro do Eclipse. Isso de certa forma é um ponto fraco das ferramentas, visto

que esta funcionalidade facilita o processo de medição. Estas informações irão auxiliar a seleção das ferramentas a serem utilizadas na medição do software SAPLIC.

Quadro 3.1 – Características das ferramentas pesquisadas

Nome da Ferramenta	Linguagem a que se destina	Métricas	Open Source	Eclipse Plugin
Eclipse Metrics Plugin	Java	Tamanho e complexidade	X	X
Code Analyzer	C, C++, Java, Assembly e Html	Tamanho	X	
PMD For Eclipse	Java	Erros no código	X	X
Jira and Bugzila	Java, JavaScript, XSL (XSLT /XPath / XSLFO)	Processos de software	X	
Jmetric	Java	Tamanho	X	
CMT++	C e C++	Tamanho e complexidade		
LDRA testbed	C++, Ada, Java, Visual Basic, Cobol, Coral, Fortran e Pascal	Tamanho e complexidade		
Jdepend Plugin para Eclipse	Java	Análise de dependências	X	X
Cyvis	Java	Tamanho e complexidade	X	
OSPC	C, C++ e Java	Complexidade		
Jmove	Java	Tamanho, complexidade, dependência e processos de software	X	
FindBugs Plugin para Eclipse	Java	Erros no código	X	X
C and C++ Code Counter (CCCC)	C, C++ e Java	Tamanho e complexidade	X	

4 MEDIÇÃO DO SOFTWARE SAPLIC – SISTEMA DE ACOMPANHAMENTO DE PROCEDIMENTOS LICITATÓRIOS DO TRIBUNAL DE CONTAS DO ESTADO DO MARANHÃO

O Tribunal de Contas do Estado do Maranhão - TCE-MA tem como missão "Exercer o controle externo e orientar a gestão pública em benefício da sociedade" (MARANHÃO, 2012). No exercício desta atividade, uma das suas atribuições é estabelecer mecanismos de controle mais eficientes sobre os procedimentos de licitação e contratos realizados por todas as instituições obrigadas a licitar, quais sejam: órgãos integrantes da administração direta, os fundos especiais, as autarquias, as fundações públicas, as empresas públicas, as sociedades de economia mista e demais entidades controladas direta ou indiretamente pela União, Estados, Distrito Federal e Municípios, conforme previsto na Lei nº 8.666, de 1993. (BRASIL, 1993).

Segundo o entendimento de Piscitelli (2004, p. 234):

> Licitação é o conjunto de procedimentos administrativos, legalmente estabelecidos, através da qual a administração pública cria meios de verificar, entre os interessados habilitados, quem oferece melhores condições para a realização de obras, serviços, inclusive de publicidade, compras, alienações, concessões, permissões e locações.

Desse modo, partindo do pressuposto de que a gestão pública implica administrar o bem público de forma a garantir a cidadania, ou seja, fazer com que o patrimônio público seja utilizado de forma pública, o TCE-MA, fazendo uso de instrumentos ou ferramentas gerenciais modernas, desenvolveu o Sistema de Acompanhamento de Procedimentos Licitatórios do Estado do Maranhão – SAPLIC.

O software SAPLIC é uma ferramenta de acompanhamento de procedimentos licitatórios que tem o objetivo de garantir agilidade, eficiência, transparência, bem como a legalidade dos procedimentos nas aquisições públicas. É uma aplicação web codificada na linguagem orientada a objetos Java conciliada com um framework de criação de interfaces ricas, chamado Zkoss. A construção desse sistema é baseado em camada de persistência EJB (*Enterprise JavaBeans*) , componente da plataforma J2EE (*Java 2 Enterprise Edition*), que é uma arquitetura de componentes do lado do servidor para *Java Platform*, *Enterprise Edition (Java EE)* e tem o objetivo de fornecer um rápido e simplificado desenvolvimento de aplicações Java baseado em componentes distribuídas, transacionais, seguras e portáveis (ORACLE, 2012, tradução nossa). Além desse padrão de arquitetura, o SAPLIC também usa MVC, que consiste em um padrão de arquitetura de software que visa a separar a lógica de negócio da lógica de apresentação. O processo de desenvolvimento foi todo baseado no

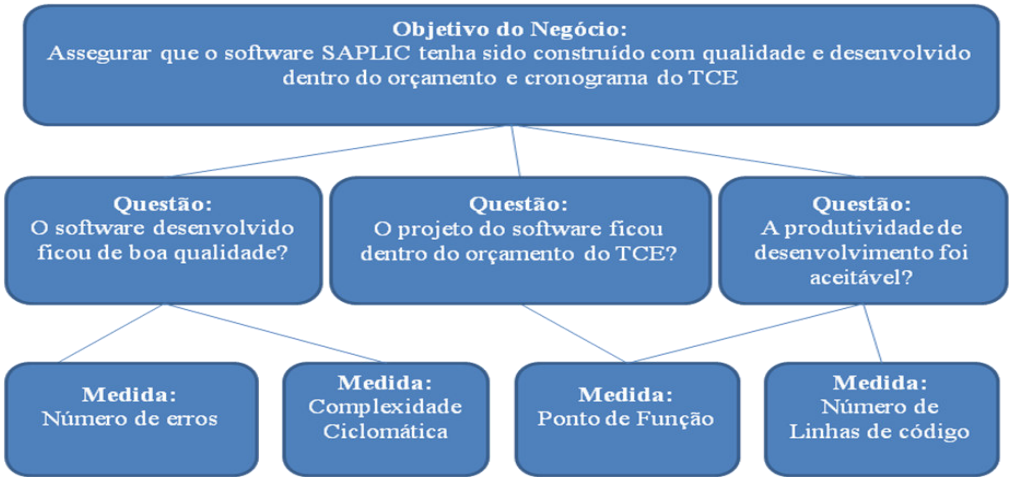
modelo XP, onde o desenvolvimento é ágil e iterativo, com a entrega constante de pequenas partes da funcionalidade do software, e a codificação foi toda feita na plataforma de desenvolvimento Eclipse.

A medição do software SAPLIC foi baseada nos modelos de processo de medição propostos por Roche (1994) e Sommervile (2011), onde se destacam basicamente 4 atividades: definição das métricas, coleta das métricas, análise das métricas coletadas e utilização dos resultados da analise em ações.

4.1 Definição das métricas

A definição das métricas foi feita com a ajuda da abordagem GQM (*Goal-Question-Metric*), que apóia o processo de medição auxiliando na definição do conjunto de métricas a serem coletadas. A Figura 4.1 ilustra a aplicação do GQM para o software SAPLIC. Inicialmente foi definido um objetivo: Assegurar que o software SAPLIC tenha sido construído com qualidade e desenvolvido dentro do orçamento e cronograma do TCE. A partir desse objetivo, algumas questões foram formuladas: O software desenvolvido ficou de boa qualidade?; O projeto do software ficou dentro do orçamento do TCE?; A produtividade de desenvolvimento foi aceitável? As respostas destas questões estão associadas a medidas. Para melhor analisar a qualidade do software SAPLIC, foi estabelecido que o número de erros juntamente com a métrica derivada densidade de erros responderiam essa questão. A métrica Complexidade Ciclomática foi escolhida também para mostrar a complexidade do código. Para responder se o orçamento estava correto e dentro do esperado, a análise por Pontos de função permitiu fazer estimativas de valor e produtividade. O número de linhas de código ajudou a estimar produtividade dos desenvolvedores.

Figura 4.1 - Uso do GQM na definição das métricas para medição do software SAPLIC



4.1.1 Escolha das ferramentas

Algumas das métricas definidas na seção anterior foram extraídas com o auxilio de ferramentas para medição vistas no Quadro 3.1 (p. 36). A escolha dessas ferramentas foi baseada em alguns critérios que levaram em conta a compatibilidade da ferramenta com o software, a disponibilidade (tipo de licença) das ferramentas, os custos da medição, os tipos de métricas extraídas e o ambiente de execução da ferramenta. Estes critérios foram:

a) a ferramenta escolhida deveria extrair métricas de softwares codificados em Java, que é a mesma linguagem de programação do software SAPLIC: a ferramenta CMT ++ já poderia ser descartada, visto que esta ferramenta é voltada apenas para as linguagens C e C++;
b) a ferramenta escolhida deveria ser *open source* e gratuita exclusivamente: as ferramentas CMT++, LDRA e OSPC não podem ser utilizadas na medição, pois possuem licenças proprietárias;
c) a ferramenta escolhida deveria extrair métricas de produto: algumas ferramentas estão voltadas para processos de software, desde a captura dos requisitos até a fase de testes e validação o que, tornaria os casos de teste teoricamente complexos visto que as saídas dessas ferramentas iriam depender de entradas de informações completas sobre todo o projeto do software. Ferramentas como Jira and Bugzila e Jmove não serão utilizadas na medição por este motivo;
d) a ferramenta escolhida deveria ser implementada como funcionalidade que existe dentro de outras ferramentas para análise e projeto, codificação ou teste: essa medida, de certa forma, torna mais fácil o processo de medição;
e) a ferramenta escolhida deveria extrair métricas de tamanho e complexidade: as ferramentas devem extrair métricas que abrangem Total de Linhas de código, Número de erros e Complexidade Ciclomática.

Portanto, com os critérios de escolha das ferramentas definidos, foram escolhidas ferramentas que estavam de acordo com os critérios acima. Logo, para extrair a métrica de Linhas de código (LOC) e Complexidade Ciclomática foi selecionada a ferramenta Eclipse Metrics Plugin. Para contar o número de erros no código foi selecionada a ferramenta FindBugs Plugin para Eclipse. Todas essas ferramentas, além de serem plugins para o eclipse

(IDE utilizada para a codificação do SAPLIC), são open source e suas métricas extraídas são métricas de produto.

A Análise por Pontos de Função do SAPLIC foi feita de forma manual. Existem ferramentas que, a partir do modelo de um programa ou de seu código, calcula seu tamanho em PF. Entretanto, os valores obtidos freqüentemente diferem em muito de uma contagem manual, pois muitas ferramentas têm dificuldades em diferenciar funções lógicas de funções físicas e ainda se baseiam na técnica de backfiring[1], que tem sido muito criticada. Logo, a forma de medição adotada não utilizou ferramenta de medição, foi feita de forma manual.

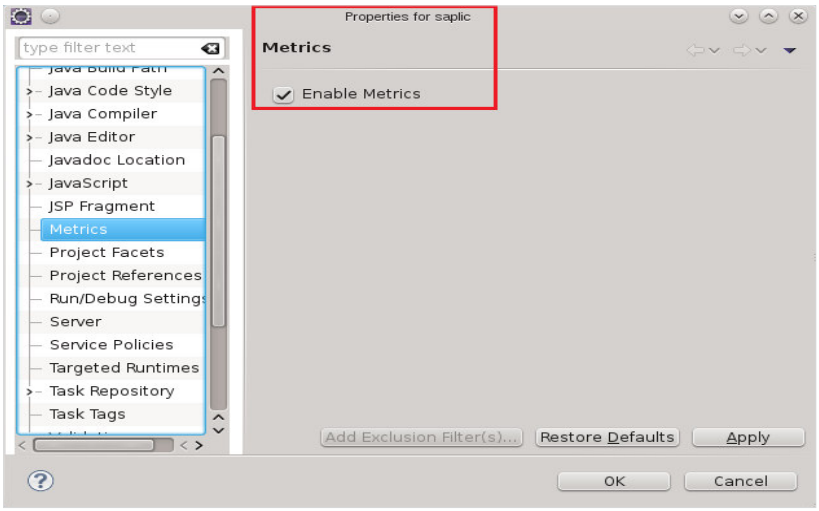
4.2 Coleta e análise das métricas

Após definida as métricas a serem coletadas e as ferramentas a serem utilizadas, a seguir será mostrado as etapas de coleta e análise das métricas Linhas de código (LOC), número de erros, Complexidade Ciclomática e Ponto de Função.

4.2.1 Linhas de código (LOC)

Inicialmente, para extrair esta métrica, a ferramenta Eclipse Metrics Plugin foi instalada no ambiente integrado para desenvolvimento de software Eclipse. Após sua instalação, para iniciar a coleta das métricas do projeto SAPLIC, a ferramenta foi ativada como mostra a Figura 4.2.

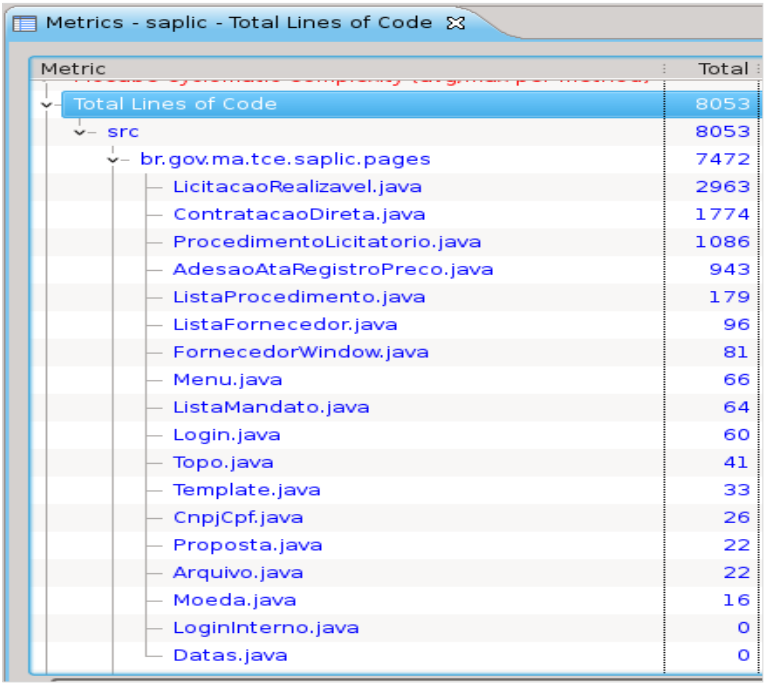
Figura 4.2 – Execução da ferramenta Eclipse Metrics Plugin no software SAPLIC



[1] Backfiring consiste em derivar o número de pontos de função a partir da quantidade de linhas de código do programa, baseado numa relação pré-estabelecida entre LOC e PF (FATTO, 2012).

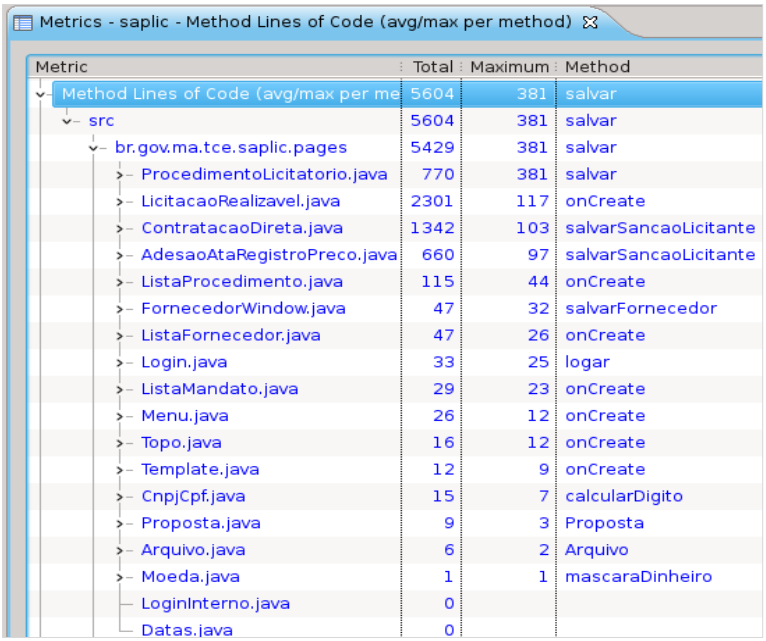
A métrica Linhas de código (LOC) é chamada na ferramenta como Total Lines of Code (TLOC). Os resultados obtidos da métrica TLOC para todo o projeto e para cada classe do projeto estão ilustrados na Figura 4.3. Observou-se que o total de linhas de código do software SAPLIC foi 8053.

Figura 4.3 – Métrica TLOC das classes do projeto do software SAPLIC



Na Figura 4.4 estão ilustrados os valores da métrica TLOC em nível de método. Além do valor total, é exibido também informações como o valor máximo e o método que se refere o valor máximo.

Figura 4.4 – Métrica TLOC dos métodos do projeto SAPLIC



A ferramenta Eclipse Metrics Plugin fornece algumas faixas seguras de métricas para Número de Linhas de código, as quais destacam-se:

a) Linhas de Código (em nível de método): Máximo 50 - Se um método tem mais que 50 linhas de código é sugerido que o método seja quebrado para favorecer a legibilidade e a manutenibilidade.

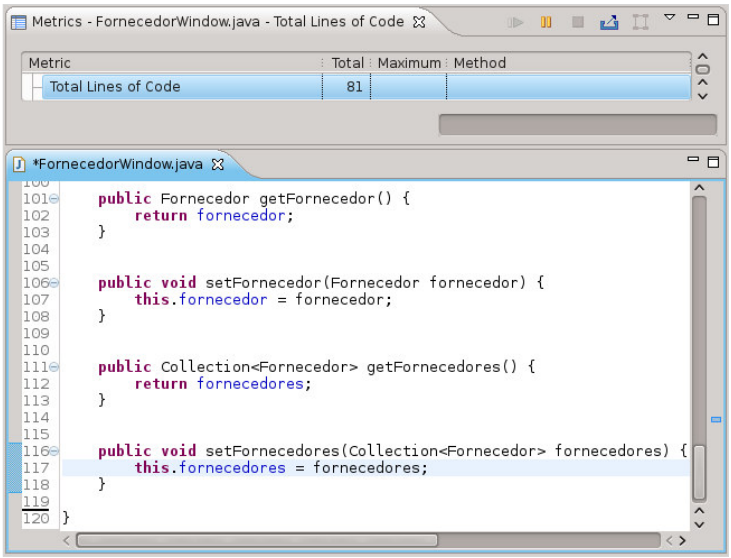
b) Linhas de Código (em nível de classe): Máximo 750 - Se uma classe tem mais que 750 linhas de código, quebre esta classe e delegue responsabilidades.

Diante disso, quatro classes pertencentes ao projeto deverão ser divididas, pois estas possuem o número total de linhas de código maior que 750, como se observa na Figura 4.3. Estas classes são: *LicitacaoRealizavel, ContratacaoDireta, ProcedimentoLicitatorio e AdesaoAtaRegistroPreco*.

Analisando a outra faixa segura de métricas fornecida pela ferramenta, se um método tem mais que 50 linhas de código, é sugerido que ele seja dividido para favorecer a legibilidade e a manutenibilidade. Na Figura 4.4, nota-se que quatro métodos estão com o número de linhas acima do sugerido, que são: método salvar() da classe *ProcedimentoLicitatorio*, método onCreate() da classe *LicitacaoRealizavel*, e os métodos salvarSancaoLicitante() das classes *ContratacaoDireta* e *AdesaoAtaRegistroPreco*. Dessa forma, estes métodos devem ser divididos no intuito de melhorar a qualidade do código e aumentar a legibilidade e manutenibilidade do código.

Na Figura 4.5, confirma-se que a métrica TLOC extraída pela ferramenta nada mais é que um SLOC (Source Lines of Code), onde as linhas em branco ou com comentários não são consideradas. A última linha de código da classe *FornecedorWindow* informa o número 120 enquanto que a métrica extraída informa o número 81.

Figura 4.5 – Métrica SLOC da Classe *FornecedorWindow*



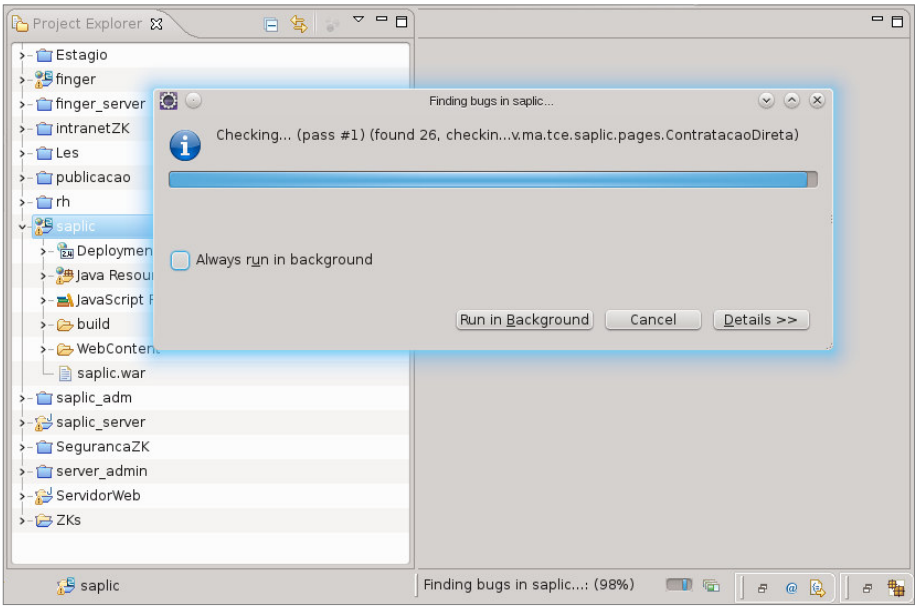
Com base na métrica LOC é possível obter indicadores como:

a) Produtividade: KLOC/pessoa-mês: Tomando como informação que o software SAPLIC foi desenvolvido por 1 programador durante 20 meses e que o valor obtido da métrica LOC foi 8053, tem-se um indicador de produtividade de aproximadamente 0,4 KLOC/pessoa-mês, ou 400 LOC/pessoa-mês, que significa que a cada mês do projeto foi desenvolvido 400 linhas de código.

4.2.2 Número de erros

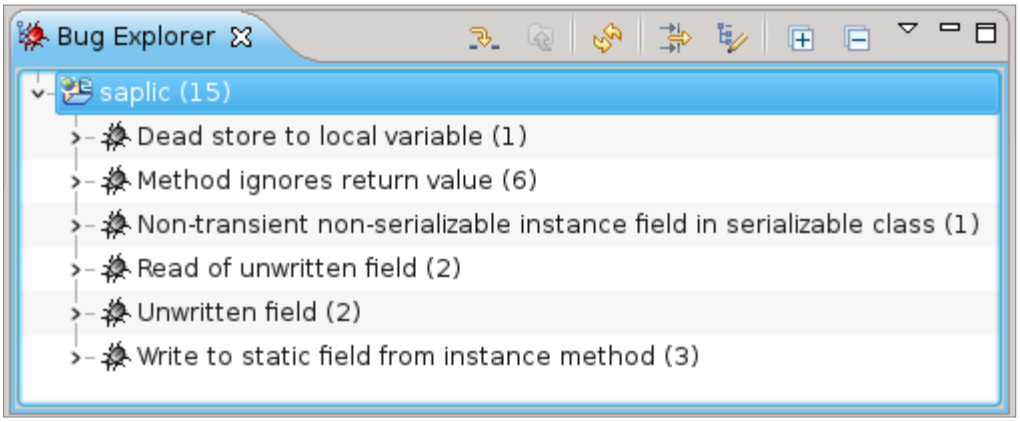
A ferramenta escolhida para a extração desta métrica foi a FindBugs Plugin para Eclipse. Inicialmente ela foi instalada no ambiente integrado para desenvolvimento de software Eclipse, e logo após foi executada no SAPLIC. A Figura 4.6 ilustra a execução da ferramenta.

Figura 4.6 – Execução da ferramenta FindBugs Plugin Eclipse no software SAPLIC



Após a execução da ferramenta, um quadro com os erros encontrados foi exibido, como mostra a Figura 4.7.

Figura 4.7 – Erros encontrados pelo FindBugs no software SAPLIC



De acordo com a Figura 4.7, nota-se que foram encontrados quinze ocorrências de erros no código do SAPLIC. Dentre as ocorrências, identifica-se seis tipos de erros, que serão detalhados a seguir de acordo com site da ferramenta FindBugs. (FINDBUGS, 2012).

a) **Dead store to local variable** - Esta instrução atribui um valor a uma variável local, mas o valor não é lido ou utilizado em qualquer instrução subsequente. Muitas vezes, isso indica um erro, porque o valor calculado nunca é utilizado;

b) **Method ignores return value -** o valor de retorno deste método está sendo ignorado e deve ser verificado. Uma causa comum deste aviso ocorre quando se invoca um método em um objeto imutável, pensando que ele atualiza o objeto;

c) **Non-transient non-serializable instance field in serializable class -** esta classe Serializable instancia um campo não primitivo que não é nem transient, serializable, ou java.lang.Object, e não aparece para implementar a interface Externalizable ou o readObject () e writeObject (). Objetos desta classe não serão serializados corretamente se um objeto não-serializable é armazenado neste campo;

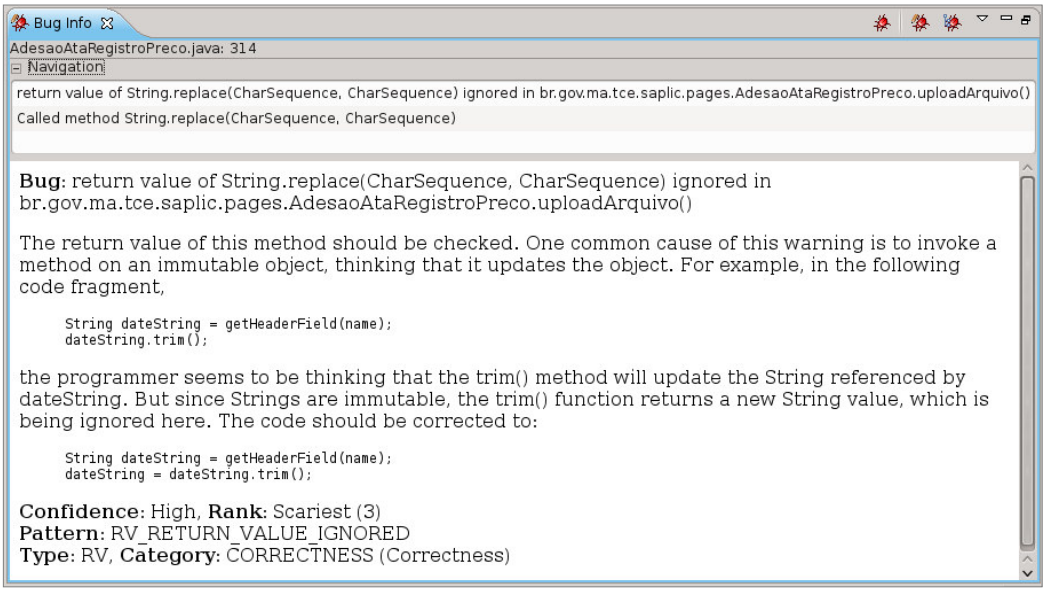
d) **Read of unwritten Field -** o programa tenta ler um campo com valor nulo. Referenciando este valor irá gerar uma exceção “null pointer”;

e) **Unwritten Field -** este campo nunca foi escrito. Todas as leituras dele irão retornar o valor padrão. Verifique se há erros (caso este tenha sido inicializado), ou remova o campo se ele é inútil;

f) **Write to static field from instance method -** este método grava em um campo estático. Se várias instâncias estão sendo manipuladas, isso se torna complicado. É uma prática geralmente ruim.

Ao selecionar um erro, as informações referentes a este erro são mostradas em uma janela chamada Bug Info, como mostra a Figura 4.8.

Figura 4.8 – Informações do bug



Analisando as informações detalhadas do erro encontrado na Figura 4.8 , verifica-se que ele se encontra na classe AdesaoAtaRegistroPreco.java, na linha 314, no corpo do método uploadArquivo(), mais especificamente na chamada do método String.replace (CharSequence, CharSequence). O erro apontado refere-se ao valor de retorno do método replace, que esta sendo ignorado, conforme especificado no tipo do erro.

Os resultados obtidos com a execução da ferramenta FindBugs no software SAPLIC permitem ao programador identificar e localizar os possíveis erros presentes no código. Munido destas informações, o programador pode corrigi-los, tornando o código mais limpo e, consequentemente, melhora a qualidade do software. A identificação da quantidade de erros, em um estudo comparativo, auxilia o supervisor de desenvolvimento na avaliação do desempenho dos programadores e, além disso, ajuda a obter indicadores de qualidade, como:

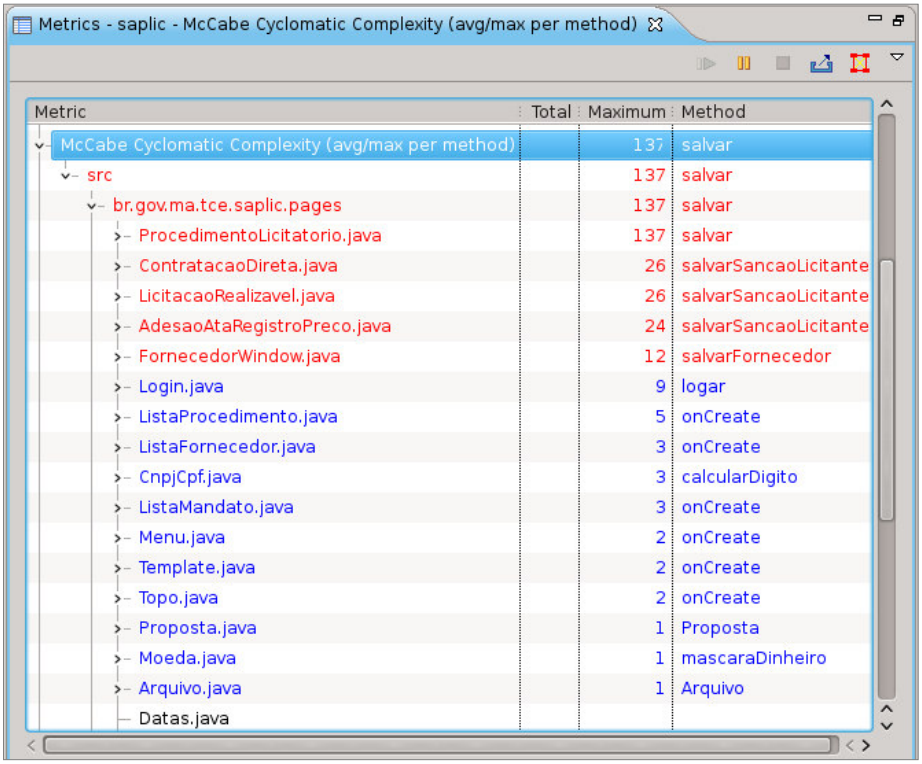
a) Qualidade: defeitos/KLOC: Com base nas métricas LOC e número de erros presentes no código do SAPLIC, foi possível obter um indicador de qualidade, que é a densidade de erros. A razão entre a quantidade de erros, que foram 15 encontrados no código do SAPLIC, pelo valor da métrica LOC do projeto, que foi 8053 LOCs ou 8,053 KLOCs (mil linhas de código), tem-se um resultado

aproximado de 1,86 defeitos/KLOC, ou seja, menos de 2 erros inseridos no código a cada mil linhas.

4.2.3 Complexidade Ciclomática

Para extrair esta métrica, a ferramenta Eclipse Metrics Plugin foi instalada no ambiente integrado para desenvolvimento de software Eclipse e foi ativada como mostrado na seção Linhas de código, Figura 4.2. Após a execução desse passo, os valores da métrica Complexidade Ciclomática foram coletados, como mostra a Figura 4.9.

Figura 4.9 – Métrica Complexidade Ciclomática das classes do projeto



A ferramenta Eclipse Metrics Plugin fornece uma faixa segura para esta métrica. Ela afirma que a Complexidade Ciclomática de McCabe , à nível de método, deve ter o valor em no máximo 10 e, se um método tem mais que 10 *loops* diferentes, ele deve ser particionado utilizando subrotinas.

Analisando as métricas extraídas na Figura 4.9, observa-se que os valores da Complexidade Ciclomática de McCabe que estão na cor vermelha não satisfazem a faixa segura para essa métrica. Logo, montou-se o Quadro 4.1 mostrando esses valores, a qual método eles correspondem e em qual classe estão presentes estes métodos.

Quadro 4.1 – Métrica Complexidade Ciclomática de McCabe fora da faixa segura

Classe	Valor máximo	Método
ProcedimentoLicitatorio	137	salvar()
ContratacaoDireta	26	salvarSancaoLicitante()
LicitacaoRealizavel	26	salvarSancaoLicitante()
AdesaoAtaRegistroPreco	24	salvarSancaoLicitante()
FornecedorWindow	12	salvarFornecedor()

Dessa forma, identifica-se no código do software SAPLIC cinco métodos propensos a erros. O método salvar() da classe *ProcedimentoLicitatorio*, os métodos salvarSancaoLicitante() das classes *ContratacaoDireta, LicitacaoRealizavel e AdesaoAtaRegistroPreco* e o método salvarFornecedor() da classe *FornecedorWindow* devem ser divididos de modo que o valor da métrica se estabilize em uma faixa segura e a complexidade do código diminua.

4.2.4 Ponto de Função

A Análise por Pontos de Função do SAPLIC será feita sequindo os passos do *Function Point Counting Pratices Manual – Release 4.1.1* (Manual de Práticas de Contagem de Pontos de Função), publicado pelo IFPUG em 1999. Como detalhado anteriormente na seção Técnicas para métricas de software, subitem Pontos de Função, o processo para contagem de pontos de função compreende sete passos. Abaixo serão mostrados os passos para a contagem com seus respectivos resultados.

Primeiro passo: Tipo de Contagem.

O tipo de Contagem escolhido foi a contagem de aplicação, pois o SAPLIC é uma aplicação já instalada e esse tipo de contagem mede a funcionalidade fornecida ao usuário pela aplicação instalada, provendo uma medida da atual funcionalidade ganha pelo usuário da aplicação.

Segundo passo: Identificação do escopo de contagem e fronteira da aplicação.

O Software SAPLIC e todas as suas funcionalidades. O escopo de contagem foi definido como sendo um único sistema: o software SAPLIC. A contagem irá abranger todas as funcionalidades desse software.

Terceiro passo: Contagem das funções de dados

A contagem das funções de dados representa a funcionalidade provida ao usuário por meio de dados internos e externos, que são os Arquivos Lógicos Internos e os Arquivos de Interface Externos.

Quadro 4.2 – Resultado da contagem dos Arquivos Lógicos Internos – ALI

ALI	Registros Lógicos	Itens de dados	Complexidade	PFs
ALTERACAO_CONTRATO	1	28	média	10
APURACAO	1	19	baixa	7
CONTRATO	1	10	baixa	7
DESISTENCIA	1	4	baixa	7
DILIGENCIA	1	5	baixa	7
DOCUMENTO	1	6	baixa	7
FORNECEDOR	1	5	baixa	7
IMPUGNACAO_INSTRUMENTO	1	6	baixa	7
IMPUGNACAO_PROCEDIMENTO	1	6	baixa	7
ITEM	1	6	baixa	7
LICITACAO	1	74	alta	15
LICITANTE	1	10	baixa	7
LOTE	1	3	baixa	7
MODALIDADE_LICITACAO	1	1	baixa	7
MODELO_DOCUMENTO	1	1	baixa	7
MODIFICACAO_INSTRUMENTO	1	7	baixa	7
RECURSO	1	8	baixa	7
REGIME_EXECUCAO_OBRA	1	1	baixa	7
SANCAO_LICITANTE	1	17	baixa	7
SANCAO_PROCEDIMENTO	1	10	baixa	7
SISTEMA_ELETRONICO	1	1	baixa	7
TIPO_APURACAO	1	1	baixa	7
TIPO_CONTRATACAO_DIRETA	1	1	baixa	7
TIPO_DOCUMENTO	1	1	baixa	7
TIPO_IMPUGNACAO	1	1	baixa	7
TIPO_LICITACAO	1	2	baixa	7
TIPO_MODIFICACAO_CONTRATO	1	1	baixa	7
TIPO_OBJETO	1	1	baixa	7
TIPO_PROCEDIMENTO_LIC	1	1	baixa	7
TIPO_RECURSO	1	1	baixa	7
TIPO_RESULTADO	1	1	baixa	7
TOTAL				**228**

Quadro 4.3 – Resultado da contagem dos Arquivos de Interface Externa – AIE

AIE	Registros Lógicos	Itens de dados	Complexidade	PFs
SAE_NATUREZA_DESPESA	1	10	baixa	5
MANDATO	3	14	baixa	5
UNIDADE	2	34	média	7
ENTE	1	9	baixa	5
RESPONSAVEL	1	22	baixa	5
TIPO_RESPONSAVEL	1	1	baixa	5
CARGO	1	1	baixa	5
TOTAL				**37**

Quarto passo: Contagem das funções de transação

As funções de transações são provenientes de entradas, consultas e saídas externas.

Quadro 4.4 - Resultado da contagem das Entradas Externas – EE

Processo	Itens de dados referenciados	ALI	#ALI	Complexidade	PFs
Incluir ALTERACAO_CONTRATO	14	ALTERACAO_CONTRATO, TIPO_MODIFICACAO_CONTRATO	2	média	4
Excluir ALTERACAO_CONTRATO	3	ALTERACAO_CONTRATO,	2	baixa	3
Alterar ALTERACAO_CONTRATO	14	ALTERACAO_CONTRATO, TIPO_MODIFICACAO_CONTRATO	2	média	4
Incluir APURACAO	5	APURACAO,LICITANTE,ITEM	3	alta	6
Excluir APURACAO	3	APURACAO, LICITACAO	2	baixa	3
Alterar APURACAO	5	APURACAO,LICITANTE,ITEM	3	alta	6
Incluir CONTRATO	12	CONTRATO, LICITANTE	2	média	4
Excluir CONTRATO	3	CONTRATO, LICITACAO	2	baixa	3
Alterar CONTRATO	12	CONTRATO, LICITANTE	2	média	4
Incluir DESISTENCIA	6	DESISTENCIA, LICITANTE	2	média	4
Excluir DESISTENCIA	3	DESISTENCIA, LICITACAO	2	baixa	3
Alterar DESISTENCIA	6	DESISTENCIA, LICITANTE	2	média	4
Incluir DILIGENCIA	7	DILIGENCIA, LICITANTE	2	média	4
Excluir DILIGENCIA	3	DILIGENCIA, LICITACAO	2	baixa	3
Alterar DILIGENCIA	7	DILIGENCIA, LICITANTE	2	média	4
Incluir DOCUMENTO	5	DOCUMENTO, TIPO_DOCUMENTO	2	média	4
Excluir DOCUMENTO	3	DOCUMENTO, LICITACAO	2	baixa	3
Alterar DOCUMENTO	5	DOCUMENTO, TIPO_DOCUMENTO	2	média	4
Incluir FORNECEDOR	7	FORNECEDOR	1	baixa	3
Excluir FORNECEDOR	3	FORNECEDOR	1	baixa	3

Alterar FORNECEDOR	7	FORNECEDOR	1	baixa	3
Incluir IMPUGNACAO_INSTRUMENTO	8	IMPUGNACAO_INSTRUMENTO	1	baixa	3
Excluir IMPUGNACAO_INSTRUMENTO	3	IMPUGNACAO_INSTRUMENTO, LICITACAO	2	baixa	3
Alterar IMPUGNACAO_INSTRUMENTO	8	IMPUGNACAO_INSTRUMENTO	1	baixa	3
Incluir IMPUGNACAO_PROCEDIMENTO	8	IMPUGNACAO_PROCEDIMENTO, LICITANTE, TIPO_IMPUGNACAO	3	alta	6
Excluir IMPUGNACAO_PROCEDIMENTO	3	IMPUGNACAO_PROCEDIMENTO, LICITACAO	2	baixa	3
Alterar IMPUGNACAO_PROCEDIMENTO	8	IMPUGNACAO_PROCEDIMENTO, LICITANTE, TIPO_IMPUGNACAO	3	alta	6
Incluir ITEM	8	ITEM	1	baixa	3
Excluir ITEM	3	ITEM, LOTE	2	baixa	3
Alterar ITEM	8	ITEM	1	baixa	3
Incluir LICITACAO	22	LICITACAO, TIPO_PROCEDIMENTO_LIC, TIPO_CONTRATACAO, MODALIDADE_LICITACAO, TIPO_LICITACAO, TIPO_OBJETO, REGIME_EXECUCAO_OBRA, NATUREZA_DESPESA	8	alta	6
Excluir LICITACAO	3	LICITACAO, APURACAO, CONTRATO, DESISTENCIA, DILIGENCIA, DOCUMENTO, IMPUGANACAO_PROCEDIMENTO, IMPUGNACAO_INSTRUMENTO, LOTE, LICITANTE, MODIFICACAO_INSTRUMENTO, RECURSO, SANCAO_PROCEDIMENTO, SANCAO_LICITANTE	14	média	4
Alterar LICITACAO	22	LICITACAO, TIPO_PROCEDIMENTO_LIC, TIPO_CONTRATACAO, MODALIDADE_LICITACAO, TIPO_LICITACAO, TIPO_OBJETO, REGIME_EXECUCAO_OBRA, NATUREZA_DESPESA	8	alta	6
Incluir LICITANTE	4	LICITANTE,FORNECEDOR	2	baixa	3
Excluir LICITANTE	3	LICITANTE, LICITACAO, APURACAO, CONTRATO, DESISTENCIA, DILIGENCIA, IMPUGANACAO_PROCEDIMENTO, RECURSO, SANCAO_LICITANTE	9	média	4
Alterar LICITANTE	4	LICITANTE,FORNECEDOR	2	baixa	3
Incluir IMPUGNACAO_INSTRUMENTO	8	IMPUGNACAO_INSTRUMENTO	1	baixa	3
Excluir IMPUGNACAO_INSTRUMENTO	3	IMPUGNACAO_INSTRUMENTO, LICITACAO	2	baixa	3
Alterar IMPUGNACAO_INSTRUMENTO	8	IMPUGNACAO_INSTRUMENTO	1	baixa	3
Incluir LOTE	5	LOTE, ITEM	2	média	4
Excluir LOTE	3	LOTE, LICITACAO, ITEM	3	média	4
Alterar LOTE	5	LOTE,ITEM	2	média	4
Incluir MODIFICACAO_INSTRUMENTO	8	MODIFICACAO_INSTRUMENTO	1	baixa	3

Excluir MODIFICACAO_INSTRUMENTO	3	MODIFICACAO_INSTRUMENTO,LICITACAO	2	baixa	3
Alterar MODIFICACAO_INSTRUMENTO	8	MODIFICACAO_INSTRUMENTO	1	baixa	3
Incluir RECURSO	10	RECURSO, TIPO_RECURSO	2	média	4
Excluir RECURSO	3	RECURSO, LICITACAO	2	baixa	3
Alterar RECURSO	10	RECURSO, TIPO_RECURSO	2	média	4
Incluir SANCAO_LICITANTE	15	SANCAO_LICITANTE, LICITANTE	2	média	4
Excluir SANCAO_LICITANTE	3	SANCAO_LICITANTE, LICITACAO	2	baixa	3
Alterar SANCAO_LICITANTE	15	SANCAO_LICITANTE, LICITANTE	2	média	4
Incluir SANCAO_PROCEDIMENTO	12	SANCAO_PROCEDIMENTO, NATUREZA DESPESA	2	média	4
Excluir SANCAO_PROCEDIMENTO	3	SANCAO_PROCEDIMENTO, LICITACAO	2	baixa	3
Alterar SANCAO_PROCEDIMENTO	12	SANCAO_PROCEDIMENTO, NATUREZA DESPESA	2	média	4
TOTAL					**201**

Quadro 4.5 – Resultado da contagem das Consultas Externas – CE

Processo	**Item de dados referenciados**	**ALI**	**#ALI**	**Complexidade**	**PFs**
Listar MANDATO	7	MANDATO, UNIDADE,ENTE, RESPONSAVEL, TIPO_RESPONSAVEL,CARGO	6	alta	6
Listar LICITACAO	4	LICITACAO, MODALIDADE_LICITACAO	2	baixa	3
Listar FORNECEDOR	4	FORNECEDOR	1	baixa	3
Listar MODIFICACAO_INSTRUMENTO	3	MODIFICACAO_INSTRUMENTO	1	baixa	3
Listar IMPUGNACAO_INSTRUMENTO	5	IMPUGNACAO_INSTRUMENTO	1	baixa	3
Listar LICITANTE	3	LICITANTE, FORNECEDOR	2	baixa	3
Listar APURACAO	7	APURACAO, ITEM, LOTE, LICITANTE	4	alta	6
Listar ITEM	6	ITEM, LOTE	2	média	4
Listar LOTE	2	LOTE, ITEM	2	baixa	3
Listar IMPUGNACAO_PROCEDIMENTO	5	IMPUGNACAO_PROCEDIMENTO, LICITANTE	2	baixa	3
Listar DESISTENCIA	4	DESISTENCIA, LICITANTE	2	baixa	3
Listar DILIGENCIA	4	DILIGENCIA	1	baixa	3
Listar SANCAO_PROCEDIMENTO	5	SANCAO_PROCEDIMENTO	1	baixa	3
Listar SANCAO_LICITANTE	2	SANCAO_LICITANTE, LICITANTE	2	baixa	3
Listar RECURSO	2	RECURSO, TIPO_RECURSO, LICITANTE	3	baixa	3
Listar CONTRATO	3	CONTRATO, LICITANTE	2	baixa	3
Listar DOCUMENTO	3	DOCUMENTO, TIPO_DOCUMENTO	2	baixa	3
Recuperar TIPO_PROCEDIMENTO_LIC	1	TIPO_PROCEDIMENTO_LIC	1	baixa	3
Recuperar MODALIDADE_LICITACAO	1	MODALIDADE_LICITACAO	1	baixa	3
Recuperar TIPO_LICITACAO	1	TIPO_LICITACAO	1	baixa	3

Recuperar TIPO_OBJETO	1	TIPO_OBJETO	1	baixa	3
Recuperar REGIME_EXECUCAO_OBRA	1	REGIME_EXECUCAO_OBRA	1	baixa	3
Recuperar SISTEMA_ELETRONICO	1	SISTEMA_ELETRONICO	1	baixa	3
Recuperar NATUREZA_DESPESA	2	NATUREZA_DESPESA	1	baixa	3
Recuperar FORNECEDOR	3	FORNECEDOR	1	baixa	3
Recuperar LOTE	2	LOTE	1	baixa	3
Recuperar ITEM	6	ITEM, LOTE	2	média	4
Recuperar LICITANTE	2	LICITANTE, FORNECEDOR	2	baixa	3
Recuperar TIPO_IMPUGNACAO	1	TIPO_IMPUGNACAO	1	baixa	3
Recuperar TIPO_RECURSO	1	TIPO_RECURSO	1	baixa	3
Recuperar TIPO_RESULTADO	1	TIPO_RESULTADO	1	baixa	3
Recuperar TIPO_MODIFICACAO_CONTRATO	1	TIPO_MODIFICACAO_CONTRATO	1	baixa	3
TOTAL					**104**

Quadro 4.6 – Resultado da contagem das Saídas Externas – SE

Processo	**Item de dados referenciados**	**ALI**	**#ALI**	**Complexidade**	**PFs**
Mapa de Apuração	8	APURACAO, LOTE, ITEM, LICITANTE	4	alta	7
Relatório de informações	14	LICITACAO, TIPO_PROCEDIMENTO_LIC, TIPO_LICITACAO, TIPO_OBJETO, NATUREZA_DESPESA	5	alta	7
TOTAL					**14**

Quinto passo: Determinar Pontos de Função não ajustados ou Brutos

Após definir o tipo de contagem, a fronteira da aplicação e reconhecer as funções de dados e de transação, foi calculado os pontos de função não ajustados, ou brutos.

Quadro 4.7 – Resultado da contagem dos Pontos de Função Não Ajustados

Função	**Complexidade**			
	Baixa	Média	Alta	Total
ALI	29 x 7 = 203	1 x 10 = 10	1 x 15 = 15	228
AIE	6 x 5 = 30	1 x 7 = 7	-	37
EE	27 x 3 = 81	21 x 4 = 84	6 x 6 = 36	201
CE	28 x 3 = 84	2 x 4 = 8	2 x 6 = 12	104
SE	-	-	2 x 7 = 14	14
Pontos de Função Não Ajustados				**584**

Sexto passo: Determinar o Fator de Ajuste (FA)

Para cada uma das 14 Características Gerais do Sistema vistas no Quadro 2.6 (p.25), foi atribuído um valor de 0 (nenhuma influência) a 5 (forte influência) que indica quanto uma característica tem influência no sistema, com visto no Quadro 4.8.

Quadro 4.8 – Valores atribuídos às Caracterísitcas Gerais do Sistema

	Características gerais do sistema	VALOR
1	Comunicação dos dados	5
2	Funções distribuídas	2
3	Desempenho	3
4	Configuração do equipamento	2
5	Volume de transações	4
6	Entrada de dados on-line	5
7	Interface com o usuário	5
8	Atualização on-line	5
9	Processamento complexo	4
10	Reusabilidade	4
11	Facilidade de implantação	3
12	Facilidade operacional	4
13	Múltiplos locais	5
14	Flexibilidade a mudanças	5

Logo após definir a influência que cada característica tem no software SAPLIC, foi aplicada a fórmula para a definição do Fator de Ajuste(FA):

"FA = 0,65 + 0,01 x (n1 + n2 + ... +n14)

onde cada n_i representa um dos 14 níveis de influência." (KOSCIANSKI; SOARES, p. 232).

FA = 0,65 + 0,01 x (5+2+3+2+4+5+5+5+4+4+3+4+5+5)

FA = 0,65 + 0,01 x 56

FA = 0,65 + 0,56

FA = 1,21

Sétimo passo: Calcular os Pontos de Função Ajustados (PFA)

Depois de realizado o cálculo do Fator de Ajuste, foi aplicada a formula abaixo para calcular os PFA da aplicação SAPLIC:

PFA = FA x PF. (KOSCIANSKI; SOARES, p. 232).

PFA = 1,21 X 584

PFA = 706,64

Ao final dos sete passos para a Análise por Ponto de Função do software SAPLIC, chegou-se ao valor de 706,64 pontos de função. Com base na quantidade de pontos de função, foi possível obter indicadores como:

a) Produtividade: PF/pessoa-mês: pontos de função implementados por pessoa/mês indicam a quantidade de esforço que será necessária para o desenvolvimento de um sistema. Tomando como informação que o software SAPLIC foi desenvolvido por 1 desenvolvedor durante 20 meses, temos um indicador de produtividade de aproximadamente 35 pontos de função por pessoa-mês;
b) Qualidade: defeitos/PF: com base no resultado obtido da métrica número de erros do software SAPLIC, pode-se definir a densidade de erros, que é um indicador de qualidade. A divisão da quantidade de erros, que foram 15 encontrados, por o total de pontos de função, que foram aproximadamente 706, nos dar um resultado de 0,021 defeitos por Ponto de Função;
c) Custo do software: não existe um preço único para Ponto de Função. O valor R$/PF irá variar de acordo com o trabalho exigido para a entrega das funcionalidades do software de acordo com o padrão técnico e de qualidade solicitado pelo cliente, como também conforme a quantidade de entregáveis (artefatos, documentos, modelos, etc) exigidos pelo cliente. Baseando-se em Editais de Serviços de Software por Ponto de Função pesquisados na web (FATTO, 2012), nota-se que os valores de um Ponto de Função possuem uma variação significativa, orbitando de R$100/PF a R$1.000/PF. Portanto, adotando um valor de R$ 400,00 por Ponto de Função, estimou-se o valor do software SAPLIC em R$ 282.656,00.

4.3 Resultados da Medição

Após coletar e analisar as métricas Linhas de Código (LOC), Número de erros, Complexidade Ciclomática e Ponto de Função do software SAPLIC, montou-se o Quadro 4.9 com os resultados dessas métricas juntamente com as métricas de qualidade, produtividade e custo derivadas a partir delas.

Quadro 4.9 – Resultado da medição do software SAPLIC

Métrica	Valor
Linhas de código (LOC)	8053
Número de erros	15
Complexidade Ciclomática	137 (max – método salvar())
Ponto de Função	706,64
Qualidade: defeitos/KLOC	1,86
Qualidade: defeitos/FP	0,021
Produtividade: KLOC/pessoa-mês	0,4
Produtividade: FP/pessoa-mês	35
Custo do software	R$ 282.656,00

5 CONCLUSÃO

A medição do Sistema de Acompanhamento de Procedimentos Licitatórios do Tribunal de Contas do Estado do Maranhão – SAPLIC teve como produto final um conjunto de métricas que descreveram o estado atual do software, evidenciando características relacionadas à qualidade, tamanho, produtividade e custo.

A aplicação de ferramentas facilitou a extração das métricas, tornando o processo de medição mais rápido e simples, exemplificando a extração da métrica Número de erros, onde após a execução da ferramenta foi possível identificar e localizar cada possível erro presente no código.

Diante do resultado obtido das métricas LOC, Número de erros e Complexidade Ciclomática, pôde-se concluir que a extração de métricas de software a partir do código sem dúvida é a maneira mais eficiente para identificar e resolver problemas da aplicação: o software SAPLIC precisa de correções no seu código, visto que foram encontrados possíveis erros, e a complexidade de alguns métodos estão muito elevadas, tornando a aplicação muito instável e dificultando uma futura manutenção.

Com a medição em Ponto de Função, foi possível obter uma noção de tamanho do software e estimar o seu custo. Então, pôde-se concluir que o software SAPLIC custou para os cofres públicos o valor de R$ 282.656,00.

Ao final deste trabalho, conclui-se que a medição de software deve ser uma constante no trabalho dos desenvolvedores de software, considerando que as vantagens decorrentes dessa prática são evidentes: as métricas extraídas possibilitaram traçar estimativas de dimensão e custo do software, avaliar produtividade e qualidade permitindo traçar melhorias no processo de desenvolvimento e no produto final, bem como melhorar o planejamento do projeto.

REFERÊNCIAS

BARCELLOS, Monalessa Perini. **Uma estratégia para medição desSoftware e avaliação de bases de medidas para controle estatístico de processos de software em organizações de alta.** 2009. 135 f. Tese (Doutorado em Programa de Engenharia de Sistemas e Computação) – Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2009.

BASILI, V. R.; ROMBACH, H. D.; CALDIERA, G. **Goal question metric paradigm**. Encyclopedia of Software Engineering, 2 Volume Set, John Wiley & Sons, Inc., 1994.

BASS, L. et al. **Constructing superior software, software quality institute series**. Macmillan Technical Publishing, 1999.

BRASIL. Lei Nº 8.666, de 21 de junho de 1993. Regulamenta o art. 37, inciso XXI, da Constituição Federal, institui normas para licitações e contratos da Administração Pública e dá outras providências. Disponível em: <http://www.planalto.gov.br/ccivil_03/ leis/l8666cons.htm>. Acesso em: 20 maio 2012.

DEMARCO, Tom. **Análise estruturada e especificação de sistema**. Rio de Janeiro: Campus, 1989.

FENTON, N. E.; NEIL, M. Software metrics: success, failures and new directions. **Journal of Systems and Software**, v. 47, p. 149-157, 1999.

FINDBUGS. FindBugs bug descriptions. Disponível em:<http://findbugs.sourceforge.net/ bugDescriptions.html>. Acesso em: 8 maio. 2012.

FATTO. Editais de serviços de software por ponto de função. Disponível em: <http://www.fattocs.com.br/editais.asp>. Acessado em: 29 jun. 2012.

HAZAN, Cláudia. Análise de pontos por função. Disponível em: <http://www.inf.ufes.br/~falbo/download/aulas/es-g/2005-1/APF.pdf>. Acesso em: 22 jun. 2012.

IFPUG. INTERNATIONAL FUNCTION POINT USERS GROUP. Princeton Junction. Function Point Counting Practices Manual release 4.1.1 [s.l.], 1999.

KOSCIANSKI, André; SOARES, Michel dos Santos. **Qualidade de software**: aprenda as metodologias e técnicas mais modernas para o desenvolvimento de software. São Paulo: Novatec, 2006.

MARANHÃO. Tribunal de Contas do Estado. O TCE-MA tem como missão. Disponível em:<http://www.tce.ma.gov.br/atribuicoesconstitucionais>. Acesso em: 8 jun. 2012.

ORACLE. Enterprise JavaBeans Technology. Disponível em:<http://www.oracle.com/technetwork/java/javaee/ejb/index.html>. Acesso em: 8 jun. 2012

PISCITELLI, Roberto Bocaccio. **Contabilidade pública**: uma abordagem da administração financeira pública. 8. ed. São Paulo: Atlas, 2004.

PRESSMAN, R.S. **Engenharia de software**. 6. ed. São Paulo: McGraw-Hill, 2006.

ROCHE, J. M. Software metrics and measurement principles, software engineering notes. **ACM**, v. 19, n. 1, p. 76 – 85, jan. 1994.

SOMMERVILLE, Ian. **Engenharia de software**. 6. ed. São Paulo: Addison Wesley, 2003.

SOMMERVILLE, Ian. **Engenharia de Software**. 9. ed. São Paulo: Pearson, 2011.

VAZQUEZ, C. E.; SIMÕES, G. S; ALBERT, R. M. **Análise de pontos de função**: medição, estimativas e gerenciamento de projetos de software. 3. ed. São Paulo: Erica, 2005.

WANG, Q.; LI, M. Measuring and Improving Software Process in China. In: Proceedings of International Symposium on Empirical Software Engineering. Hoosa Head, Australia: ISESE, 2005. p. 183-192.

Rodrigues, Leandro do Nascimento Costa

Métricas de software: medição do Sistema de Acompanhamento de Procedimentos Licitatórios do Tribunal de Contas do Estado do Maranhão - SAPLIC / Leandro do Nascimento Costa Rodrigues. –2012.

59.

Impresso por computador (Fotocópia).

Orientadora: Maria Auxiliadora Freire.

Monografia (Graduação) - Universidade Federal do Maranhão, Curso de Ciência da Computação, 2012.

1. Software – Engenharia 2. Métricas de software 3.Software – Qualidade

CDU 004.41