

Paulo Roberto Miranda Meirelles

**Levantamento de Métricas de Avaliação de
Projetos de Software Livre**

Orientador: Prof. Dr. Fabio Kon

Durante a elaboração deste trabalho, o autor teve apoio financeiro do Projeto Qualipso e CNPq

São Paulo, Dezembro de 2008

Sumário

1	Introdução	1
2	Métricas de Software	3
2.1	Classificação das Métricas de Software	4
2.2	Escalas de Medição para Métricas de Software	4
2.3	Métricas de Software Tradicionais	5
2.3.1	Métricas Objetivas	6
	Métricas de Dimensão de Software	6
	Métricas de Halstead	7
	Métricas de Complexidade de Software	9
	Métricas de Qualidade de Software	10
2.3.2	Outras Métricas Tradicionais	12
2.4	Métricas de Orientação a Objetos	14
	Métricas de Classes	14
	Métricas de Métodos	16
	Métricas de Acoplamento	16
	Métricas de Herança	16
	Métricas do Sistema	17
2.5	Medidas Aplicadas aos Métodos Ágeis	18
3	Aspectos Relevantes e Métricas de Qualidade para Adoção de um Software Livre	20
3.1	Métricas e Aspectos Seleccionados	20
4	Resultados do Levantamento de Métricas de Avaliação de Projetos de Software Livre	28
4.1	Perfil dos Entrevistados	28
4.2	Resultados das Métricas e Aspectos Pesquisados	41
4.2.1	Aspectos e Métricas mais Relevantes	56
5	Considerações Finais	59
A	Questionário de Levantamento de Aspectos Relevantes e Métricas de Qualidade para Adoção de um Software Livre	60
	Referências Bibliográficas	69

Resumo

Tradicionalmente, qualidade de Software é comumente atribuída ao processo de desenvolvimento do software que teoricamente leva à conformidade com os requisitos elencados. O surgimento e a adoção de metodologias ágeis, procurando “desburocratizar” o processo de desenvolvimento, e o crescimento do impacto do software livre na indústria demonstram que, em muitos casos, o código-fonte de um determinado programa passa a ser o principal e/ou único artefato ao final de sua construção. Assim, as métricas convencionais de qualidade que têm como base a documentação do processo de software já não podem ser consideradas suficientes para a avaliação de muitas aplicações disponíveis. Desse modo, este trabalho pretende investigar, dentro do estado-da-arte e de uma ampla pesquisa realizada com especialistas em desenvolvimento de software, as métricas e os aspectos baseados no código-fonte, testes, manutenibilidade, comunidade, entre outros apresentados neste trabalho, capazes de determinar características da qualidade do software, ajudando na escolha de adoção de um software livre.

Capítulo 1

Introdução

A indústria de software incorporou ao longo do tempo a idéia de que qualidade de software é composta, em resumo, por qualidade do processo de desenvolvimento, qualidade do projeto e qualidade do produto (Rocha, Maldonado e Weber, 2001). De acordo com essa visão, ela é definida como a conformidade do software desenvolvido com os requisitos e características implícitas que são esperadas do software (Pressman, 1995). Ou seja, pode ser vista como um conjunto de características que devem ser atingidas em um determinado grau para que o produto atenda às necessidades de seus usuários (Rocha, Maldonado e Weber, 2001). Dessa forma, muitas das métricas comumente usadas são baseadas na documentação gerada no processo de desenvolvimento.

As metodologias ágeis, como a Programação eXtrema (XP) (Beck e Fowler, 2000; Beck e Andres, 2004), surgem como um contraponto aos métodos tradicionais de desenvolvimento de software, de maneira que os recursos e tempo de desenvolvimento priorizam a construção de código-fonte de qualidade. Isso é similar à forma de desenvolvimento de software adotada pelas comunidades de desenvolvedores de software livre, ou seja, software que tem como uma de suas características ser de código aberto.

O impacto e a utilização dos programas de código aberto na indústria de software têm aumentado gradativamente nas últimas décadas e tais programas já influenciam significativamente a economia global (Benkler, 2006). No entanto, há empresas e governos ainda relutantes em adotar software livre devido a dúvidas legais, incertezas comerciais, questões culturais ou falta de confiança tanto na qualidade do produto (código produzido) quanto no suporte para esse tipo de software. Por outro lado, por suas características, os programas de código aberto apresentam um enorme potencial para melhorar a produtividade, competitividade e agilidade da indústria de software brasileira, hoje ainda muito aquém das possibilidades do país.

Uma melhor exploração dos problemas relacionados à qualidade do software de código aberto pode apontar o quão pertinente é um estudo aprofundado das métricas para a avaliação da qualidade de software baseada no código-fonte e características e aspectos como testes, atividades das comunidades, canais de distribuição, licenças, ferramentas entre outros também apresentados neste trabalho. Tal estudo será focado no desenvolvimento de métricas que permitirão a análise de aspectos como segurança, flexibilidade, clareza, modularidade, manutenibilidade, portabilidade do software e que possam ser levantadas automaticamente por ferramentas criadas e/ou adaptadas para prover tais informações.

Embora haja centenas de empresas e governos interessados em utilizar mais amplamente componentes de software livre em suas infra-estruturas de tecnologia da informação, eles muitas vezes podem se perder diante da enorme quantidade de opções disponíveis e da falta de critérios objetivos para avaliar

a confiabilidade das opções existentes. Nesse contexto, *como é possível medir a qualidade do software para ajudar na escolha da adoção de um desses programas ou bibliotecas livres disponíveis? Quais são as métricas de qualidade? Como medir essas métricas? Como comparar os programas de forma a embasar uma escolha entre várias alternativas semelhantes? Como automatizar e avaliar a aplicação dessas métricas?*. Este presente trabalho é o ponto inicial para se obter algumas dessas respostas.

Ao contrário do software “proprietário”, cujo código-fonte normalmente não é disponível para seus clientes e usuários, o software livre permite a análise de seu código e mesmo a execução de testes do tipo “caixa branca”, onde os módulos internos do software são avaliados. Quando se tem acesso ao código-fonte, é possível realizar medições sobre a sua estrutura e organização interna de forma a avaliar sua qualidade. Em estudos recentes (Henderson-Sellers, 1996; Fenton e Pfleeger, 1998; Sato, Goldman e Kon, 2007), é enfatizado o fato das métricas de código-fonte permitirem a análise de dados relacionados aos erros, flexibilidade, complexidade, legibilidade, manutenibilidade, segurança e portabilidade do código, que são fatores para a aceitação de um produto de software. Além dessa análise do código-fonte, em boa parte dos projetos de software livre também é possível acessar seu sistema de controle de versões, lista de participantes e desenvolvedores e base de dados de erros e correções (Gousios et al., 2007). A partir desses dados seria possível obter informações sobre a agilidade da equipe de desenvolvimento e comunidade de um projeto em atender as necessidades de seus usuários, sobre a rapidez com que os erros reportados são corrigidos, sobre o quão ativa é a comunidade de usuários, entre outras. Um estudo sistematizado sobre esses dados poderia ajudar empresas e governos na decisão sobre qual alternativa adotar, uma vez que eles poderiam ter uma idéia sobre a qualidade do suporte oferecido pela comunidade por trás de um determinado software livre.

Esses novos conceitos, maneiras de desenvolver software e questões sobre a adoção dos software de código aberto exigem também uma alteração ou adaptação na forma de avaliação da qualidade do software, tendo como base o código-fonte, métricas objetivas e aspectos da equipe de desenvolvimento e comunidade envolvida, e não uma ênfase na documentação ou o processo de desenvolvimento. Dessa forma, o presente trabalho prevê a abordagem de três tópicos: (i) no Capítulo 2, a investigação do estado-da-arte em métricas de software com o intuito de conhecer os primeiros estudos e acompanhar a evolução e as mudanças de contextos nas últimas décadas; (ii) no Capítulo 3, uma seleção de aspectos e métricas objetivas e subjetivas com uma abordagem diferente em relação ao que é qualidade de software; (iii) no Capítulo 4, os resultados de uma pesquisa sobre os aspectos e métricas para adoção de software livre, que além dos dados sobre relevância dos aspectos e métricas, tem um levantamento do perfil dos entrevistados, da atuação dos mesmos em projetos de software livre e participação e relação das organizações e empresas, que os entrevistados pertencem, com os sistemas livres disponíveis. Serão apresentados, de forma resumida, as opiniões dos especialistas em desenvolvimento de software sobre os 106 aspectos e métricas elencados no questionário disponível na íntegra no Anexo A deste trabalho.

Capítulo 2

Métricas de Software

O objetivo das métricas de software, de acordo com o seu conceito original, é a identificação e medição dos principais parâmetros que afetam o desenvolvimento de software (Mills, 1998). A necessidade das métricas de software teve maior atenção quando se constatou a chamada “crise do software”, como bem argumentado em (Arthur, 1985). Outro fator que corroborou essa necessidade foi a constatação de uma gerência ineficaz durante o desenvolvimento da maior parte das soluções de software, uma vez que o desenvolvimento de software é complexo e não se tinha medidas bem definidas e viáveis para a avaliação do software e seu desenvolvimento. Porém, estimativas precisas e eficazes, planejamento e controle são aspectos difíceis de se concretizar em conjunto (Mills, 1998). Assim, com as métricas de software, se propõe uma melhoria do processo de gestão com a identificação, medição e controle dos parâmetros essenciais do software.

Métricas de software têm sido propostas e usadas faz algum tempo como citado no trabalho de (Wolverton, 1974; Perlis, Sayward e Shaw, 1981). Uma métrica, segundo definição da ISO 9126 (ISO/IEC9126-1, 2001), é a composição de procedimentos para medição e escalas de medidas. O Instituto de Engenharia de Software (*SEI - Software Engineering Institute*) definiu o termo “métricas de software” como um conjunto de medidas de um processo ou produto de software, onde um produto de software pode ser visto como um objeto abstrato que evolui de uma instrução inicial para o sistema de software finalizado, incluindo o código-fonte e variadas formas de documentação produzidas durante o desenvolvimento (Mills, 1998). Então, métricas são utilizadas para estimar um cronograma e custos de desenvolvimento do software e medir a produtividade e qualidade do produto.

Métricas que facilitam o desenvolvimento de modelos capazes de estimar o processo e/ou os parâmetros do software são consideradas boas métricas, sendo ideal terem as seguintes características (Mills, 1998):

- Simplicidade: o significado da métrica para a avaliação é claro;
- Objetividade;
- Fácil obtenção: o custo de obtenção da métrica não é excessivo;
- Validade: a métrica efetivamente mede o que se propõe a medir;
- Robustez: a métrica não sofre grandes desvios com pequenas mudanças no processo e no software;
- Linearidade de escala: há formas de mapeamento para o entendimento do comportamento das entidades através da manipulação dos números obtidos (Fenton e Pfleeger, 1998).

2.1 Classificação das Métricas de Software

Métricas de software podem ser classificadas quanto ao objeto das métricas, ou seja, o âmbito da sua aplicação, quanto ao critério utilizado na sua determinação e quanto ao método de obtenção da medida.

A maneira mais ampla de classificá-las é quanto ao objeto da métrica, subdividindo-as em *métricas de produtos*, que medem a complexidade e tamanho final do programa ou sua qualidade (confiabilidade, manutenibilidade etc.), e em *métricas de processo*, que referem-se ao processo de concepção e desenvolvimento do software, medindo, por exemplo, o processo de desenvolvimento, tipo de metodologia usada e tempo de desenvolvimento (Mills, 1998).

Quanto aos critérios, as métricas são diferenciadas em *métricas objetivas* e *métricas subjetivas*. As objetivas são obtidas através de regras bem definidas, sendo a melhor forma de possibilitar comparações posteriores consistentes. Assim, os valores obtidos por elas deveriam ser sempre os mesmos, independentemente do instante, condições ou indivíduo que os determinam. A obtenção dessas métricas é passível de automatização, como por exemplo número de linhas de código (LOC) (Conte, Dunsmore e Shen, 1986). As subjetivas podem partir de valores, mas dependem de um julgamento que também é um dado de entrada para ser levantadas, como por exemplo o modelo de estimativa de custo (Boehm, 1981), que depende da classificação do tipo de software (se o software é embarcado, distribuído etc.).

Outra maneira de classificar as métricas de software, quanto ao *método de obtenção*, é dividi-las em *primitivas* ou *compostas* (Grady e Caswell, 1987). As métricas primitivas são aquelas que podem ser diretamente observadas em uma única medida, como o número de linhas de código, erros indicados em um teste de unidade ou ainda o total de tempo de desenvolvimento de um projeto. As métricas compostas são as combinações de uma ou mais medidas como o número de erros encontrados a cada mil linhas de código ou ainda o número de linhas de teste por linha de código.

Existem ainda classificações que combinam as maneiras e características apresentadas para atribuir um tipo as métricas. Neste trabalho vamos usar como parâmetro de classificação a distinção entre métricas objetivas e subjetivas e consideraremos algumas métricas de produto também como métricas objetivas.

2.2 Escalas de Medição para Métricas de Software

As métricas de software precisam ser coletadas em um modelo de dados específico que pode envolver cálculos ou análise estatística subjetiva. Para isso, é importante considerar o tipo de informação obtida. Assim, quatro tipos de dados de medidas foram reconhecidos por estatísticos para as métricas de software (Conte, Dunsmore e Shen, 1986; Fenton e Pfleeger, 1998). A tabela abaixo mostra resumidamente o conceito e as diferenças entre os tipos de escalas de medição.

- Nominal: existe um nome ou um valor para um atributo; no entanto, a ordem dos valores não tem nenhum significado para a sua interpretação;
- Ordinal: os resultados estão em uma determinada ordem (ascendente ou descendente), mas a distância entre os pontos dessa escala não tem significado;
- Intervalo: preserva a importância da ordem dos resultados e possui informações sobre o tamanho dos intervalos que separaram os pontos da escala, mas as relações entre valores não são necessári-

Tipo de Dados	Operações	Descrição	Exemplo
Nominal	=,!=	Categorias	Dados do programa, Sistema Operacional
Ordinal	<,>	Classificação	Nível de experiência dos programadores
Intervalo	+,-	Diferenças	Complexidade Ciclométrica
Racional	/	Zero Absoluto	Linhas de Código (LOC)

Tabela 2.2: Tipo des Escalas de Medição

amente válidas. Por exemplo, um programa com complexidade de valor 6 é mais complexo em 4 unidades do que um programa com a complexidade de valor 2, mas isso não é muito significativo para dizer que o primeiro programa é 3 vezes mais complexo do que o segundo;

- Racional: semelhante à escala de intervalo, mas representando também as proporções entre as entidades e possuindo um zero absoluto. Um programa de 2000 linhas pode razoavelmente ser interpretado como sendo duas vezes maior que um programa de 1000 linhas, e obviamente os programas podem ter comprimento zero, de acordo com essa medida.

As escalas de medição devem estar associadas a uma dada métrica. Muitas métricas propostas têm valores a partir de um intervalo, um ordinal, ou mesmo uma escala nominal. Métricas associadas a uma escala racional podem ser preferíveis, uma vez que uma escala racional tem mais dados que permitem aplicar operações matemáticas de forma mais significativa. No entanto, segundo (Mills, 1998), os valores de vários parâmetros essenciais ao processo de desenvolvimento de software não estão associados com uma escala racional.

2.3 Métricas de Software Tradicionais

As métricas de software tradicionais existentes, geralmente aplicadas de forma isolada, vêm sendo consideradas insatisfatórias há décadas (Mills, 1998). No passado, muitas métricas e uma série de processos foram propostos (Mohanty, 1981; Kafura e Canning, 1985; Kemerer, 1987; Rubin, 1987), mas a maioria das métricas definidas não possuem uma base teórica suficiente e/ou uma significativa validação experimental. Em sua maioria, cada uma delas foi definida por um indivíduo e, em seguida, testada e utilizada em apenas um ambiente, provavelmente limitado. Segundo (Mills, 1998), em alguns casos, existem relatos significativos de validação ou de aplicação dessas métricas. No entanto, outros testes ou o seu uso em ambientes diferentes têm produzido resultados não esperados. Essas diferenças não são surpreendentes, uma vez que faltam definições claras e hipóteses de testes bem definidas.

Assim, existe um grande número de métricas, mas apenas algumas têm sido amplamente utilizadas ou aceitas. Mesmo nos casos das métricas amplamente estudadas, como o número de linhas de código (LOC), métricas de Halstead e Complexidade Ciclométrica, não são universalmente medidas de comum acordo. Estudos relatam experiências que tentam correlacionar as métricas com um número de propriedades de software, incluindo o tamanho, complexidade, confiabilidade, número de erros e manutenibilidade (Curtis et al., 1979; Curtis, Sheppard e Milliman, 1979; Kafura e Canning, 1985; Li e Cheung, 1987; Potier et al., 1982; Woodfield, Shen e Dunsmore, 1981).

Outras métricas que podem ser relevantes, como o tipo de produto e o nível de conhecimento do programadores, são exemplos de métricas subjetivas, porém difíceis de se especificar e com problemas ligados à avaliação de fatores individuais.

Outro obstáculo vem do fato de que é difícil interpretar e comparar resultados das métricas, especialmente se elas envolvem diferentes ambientes, linguagens, aplicações e/ou metodologias de desenvolvimento. Como consequência, não existem processos ou modelos de aplicação de métricas com bases teóricas relevantes. A maioria é baseada na combinação de intuição, características particulares e análise estatística de dados empíricos (Mills, 1998).

Mais um problema ocorre quando se usa métricas simples, como as que envolvem linhas de código (LOC), pois diferenças técnicas, como diferentes linguagens de programação, influem na contagem e podem impossibilitar a comparação dos resultados (Jones, 1986).

Entretanto, mesmo com os problemas levantados, ponderadamente, quando se tem métodos de aplicação de métricas de software em um ambiente limitado, é possível auxiliar, de forma significativa, na melhoria da qualidade do software e da produtividade (Basili e Rombach, 1987; Grady e Caswell, 1987). Em muitos casos, métricas relativamente simples, como linhas de código e complexidade ciclomática foram relativamente bons preditores de outras características, como número de erros, esforço total e manutenibilidade (Grady e Caswell, 1987; Li e Cheung, 1987; Rombach, 1987). Embora úteis, essas métricas não podem ainda ser utilizadas indiscriminadamente, mas a aplicação cuidadosa de algumas das métricas e modelos disponíveis podem produzir resultados úteis, se de acordo com o ambiente específico (Mills, 1998).

2.3.1 Métricas Objetivas

Boa parte das métricas objetivas, conforme a denominação deste trabalho, tratam características do código-fonte. Uma série de trabalhos deu início às abordagens como tamanho do programa e complexidade do software (Troy e Zweben, 1981; Henry e Kafura, 1984; Yau e Collofello, 1985). Existe uma boa quantidade de métricas desses tipos; por exemplo, são referidas e comparadas 31 métricas de complexidade diferentes em (Li e Cheung, 1987) e foram propostas novas métricas de complexidade em (Card e Agresti, 1988; Harrison e Cook, 1987). Um número de métricas objetivas mais referenciadas e conhecidas do estado da arte de métricas de software será tratado nas subseções a seguir. Serão discutidas as métricas que refletem as áreas em que a maioria dos trabalhos sobre métricas foram realizados.

Métricas de Dimensão de Software

Algumas métricas de software foram desenvolvidas para tentar quantificar o tamanho do software e auxiliar as medições na fase de concepção do software, ou seja, partindo dos princípios de um processo de desenvolvimento tradicional.

- *Linhas de Código (LOC)*

Possivelmente a mais usada para medir o tamanho do programa. Aparentemente é de fácil definição e precisão; entretanto, há uma série de pontos particulares para o número de linhas de código em um determinado programa. Essas particularidades de tratamento envolvem linhas em branco, linhas de comentário, trechos não-executáveis, múltiplas declarações por linha e várias linhas por declaração, assim como a questão das linhas de código repetidas ou reusadas. O conceito mais comum de LOC parte do princípio de contar qualquer linha que não seja linha em branco ou comentário, independentemente do número de declarações por linha (Boehm, 1981; Jones, 1986), só podendo ser comparados valores diferentes dessa métrica caso elas se referirem à mesma linguagem e se o estilo de programação estiver normalizado (Jones, 1991).

LOC foi atribuída como preditor de complexidade do software, esforço total de desenvolvimento e desempenho do programa (depuração e produtividade). Estudos tentaram validar esses relacionamentos, como (Woodfield, Shen e Dunsmore, 1981) comparando LOC, complexidade ciclomática e métricas de Halstead (os dois últimos serão tratados nas próximas subseções) como indicadores de esforço de programação, assim como em (Curtis et al., 1979; Curtis, Sheppard e Milliman, 1979), usando LOC para comparação com outras métricas como indicadores de desempenho do programador. Outro estudo (Levitin, 1986) concluiu que LOC é uma medida de tamanho de software mais pobre que o “comprimento de programa” de Halstead.

- *Pontos de Função (FP)*

Baseado nos modelos e ciclos de vida tradicionais de desenvolvimento de software foi proposta uma medida de tamanho de software que pode ser estimada no início do desenvolvimento. A métrica calcula o valor total Pontos de Função para o projeto (Albrecht e Gaffney, 1983), que tem com parâmetros o número de entradas do usuário, as consultas, saídas e os principais arquivos. O valor dos FP é a soma desses valores individuais, com as seguintes ponderações aplicadas:

- Entradas = 4;
- Saídas = 5;
- Consultas = 4;
- Principais Arquivos = 10.

Entretanto, cada parâmetro pode ser ajustado dentro de um intervalo de $\pm 35\%$ de acordo com o projecto (Albrecht e Gaffney, 1983). Essa métrica foi validada em alguns trabalhos, comparando o LOC e PF como preditores de esforço de desenvolvimento (Albrecht e Gaffney, 1983) e relacionam PF com a produtividade em um ambiente desenvolvimento (Behrens, 1983). Mas outro estudo analisou o conjunto de dados usados no primeiro trabalho citado e concluiu que PF por si só é insuficientemente precisa para a predição de esforço de desenvolvimento de software (Knafl e Sacks, 1986).

- *System Bag*

É uma medida de dimensão total de um software determinada a partir das funcionalidades descritas em especificações formais (DeMarco, 1982). O algoritmo de cálculo se difere por se aplicar a sistemas orientados por dados ou orientados por processos. Um dos objetivos dessa métrica é maximizar o quociente Bag/Custo total no desenvolvimento do projeto.

Métricas de Halstead

É um conjunto de métricas baseadas na teoria da informação, consideradas as primeiras métricas com fundamentação teórica comum, também chamada de *Software Science* (Halstead, 1972, 1977). Essas métricas se aplicam a vários aspectos do software, diferentemente da maior parte das métricas, que tratam um aspecto particular, e também são usadas para avaliar o esforço global de desenvolvimento do software. O Vocabulário (n), Comprimento (N) e Volume (V) são métricas que aplicam-se especificamente ao software final. Também foram especificadas fórmulas para calcular o esforço total (E) e tempo de desenvolvimento (T) de software.

- *Vocabulário do Programa*

O software pode ser visualizado com uma sequência de símbolos (*tokens*), sendo cada símbolo classificado em operadores ou operandos. Assim, o vocabulário do programa (n) é definido como:

$$n = n_1 + n_2$$

, onde n_1 é o número de operadores únicos e n_2 é o número de operandos únicos do programa.

- *Comprimento do Programa*

Enquanto o vocabulário é a soma dos símbolos (*tokens*) diferentes, o comprimento do programa (N) é a soma do total de operadores e operandos, sendo dado como:

$$N = N_1 + N_2$$

, onde N_1 é o número total de operadores e N_2 é o número total de operandos do programa. Porém, a distinção entre operadores e operandos não é muito clara (Halstead, 1977). Então o valor N pode ser estimado como:

$$N' = n_1 * \log(n_1) + n_2 * \log(n_2)$$

Sendo N uma medida que pode ser observada ao final do desenvolvimento do código, diferentemente de N' que pode ser calculada em função dos atuais ou estimados valores de n_1 e n_2 . Estudos empíricos demonstram a validade da equação mostrada (Elshoff, 1976; Halstead, 1977). Outros estudos relacionaram N e N' com outras métricas, como complexidade (Potier et al., 1982) e número de erros (Elshoff, 1976; Halstead, 1977; Shen et al., 1985; Levitin, 1986; Li e Cheung, 1987). É defendido que a métrica de comprimento é mais objetiva e eficiente que o LOC (Lassez et al., 1981).

- *Volume do Programa*

O volume de um programa, medido em bits, é dado em função do seu comprimento e do seu vocabulário:

$$V = N * \log(n).$$

Determinar o volume não é difícil, pois não envolve análise semântica. Porém, a soma do volume dos módulos de um programa é diferente em relação à soma total do volume do software, devido à variação do vocabulário. Há análises sobre a abrangência da medida do volume do software, destacando a independência em relação à linguagem de programação utilizada (Shen, Conte e Dunsmore, 1983). Foram apresentadas evidências empíricas de que as métricas de comprimento e de volume de Halstead estão linearmente relacionadas com a LOC (Li e Cheung, 1987; Christensen, Fitsos e Smith, 1981).

Métricas de Complexidade de Software

Assim como as métricas de dimensão de software, as métricas de complexidade foram pensadas de acordo com os modelos tradicionais de desenvolvimento e podem ser computadas no início do ciclo de desenvolvimento de software, para o maior sucesso na gerência do processo de software.

- *Complexidade Ciclomática ($v(G)$)*

Parte do princípio que a complexidade depende do número de condições (caminhos), correspondendo ao número máximo de percursos linearmente independentes em um software. A proposta é que se possa medir a complexidade do programa, assim orientando o desenvolvimento e os testes do software (McCabe, 1976). Essa métrica pode ser representada através de um grafo de fluxo de controle, onde os nós representam uma ou mais instruções sequenciais e os arcos orientados indicam o sentido do fluxo de controle entre várias instruções. A complexidade ciclomática de um determinado grafo pode ser calculada através de uma fórmula da teoria dos grafos:

$$v(G) = e - n + 2$$

onde e é o número de arestas e n é o número de nós do grafo.

Um estudo concluiu que a produtividade diminui de forma não linear proporcionalmente ao aumento da densidade dos pontos condicionais (Gaffney, 1979). A complexidade ciclomática também foi relacionada com os esforços de depuração e manutenção, podendo ser utilizada para estimar custos a partir do projeto detalhado dos módulos (Curtis, Sheppard e Milliman, 1979; Woodfield, Shen e Dunsmore, 1981; Harrison et al., 1982).

Surgiram propostas de extensões e alterações no cálculo dessa métrica para melhorar a sua validade (Myers, 1977; Stetter, 1984). Myers observou que complexidade ciclomática mede a complexidade do software, mas não consegue diferenciar a complexidade de alguns casos simples, em especial que envolvam uma única condição. Para melhorar a fórmula original, foi sugerido:

$$v(G)' = [l : u]$$

onde l e u são limites inferiores e superiores, respectivamente, para a complexidade, sendo mais satisfatórios para os casos verificados por Myers. Por outro lado, Stetter propôs que o grafo fosse expandido para incluir declarações e referências de dados, mostrando uma complexidade mais completa. Nesse grafo do fluxo do programa (H), geralmente, tem múltiplos nós de entradas e saídas. Assim, as irregularidades verificadas por Myers são eliminadas por uma função $f(h)$.

Uma correlação entre a complexidade ciclomática e as métricas de Halstead foi apresentada em (Henry, Kafura e Harris, 1981). O interesse na métrica de complexidade ciclomática levou à sua normalização (McCabe, 1982). Posteriormente, McCabe definiu outras cinco métricas: *complexidade da unidade real*, *complexidade essencial*, *complexidade do projeto de módulos*, *complexidade total do projeto* e *complexidade de integração* (McCabe e Butler, 1989). Essas métricas foram utilizadas para identificar e minimizar a complexidade em códigos não estruturados, decidindo sobre o número de testes necessários para uma cobertura total das possíveis execuções, eliminando a redundância e restringindo a complexidade de módulos produzidos a um nível aceitável (Mannino, Stoddard e Sudduth, 1990).

- *Número de Nós*

Corresponde ao número de nós em um grafo de fluxo de controle que representa a sequência de controle de execução de instruções num programa (Woodward, Hennell e Hedley, 1979). Um nó é definido como uma passagem necessária das linhas direcionais no grafo, sendo assim uma proposta de métricas de complexidade de software.

- *Fluxo de Informação*

Foi proposto que os fluxos de informações na estrutura de um programa são medidas de complexidade de software (Kafura e Henry, 1981), sendo comparadas com a complexidade ciclomática e métricas de Halstead. Esse método conta o número de parâmetros de entradas (*fan-in*) e saídas (*fan-out*), sendo definido como:

$$c = [tamanhoMetodo] * [fan - in * fan - out]^2$$

Uma experiência de validação dessa métrica foi realizada na concepção e desenvolvimento de um núcleo para o sistema operacional UNIX (Henry e Kafura, 1984). Experiências combinando a utilização de fluxo de informação com várias outras métricas foram realizadas também em (Kafura e Canning, 1985; Kafura e Reddy, 1987). Além disso, fluxo de informação também permite estimar com uma certa precisão o esforço de manutenção (Rombach, 1987).

Métricas de Qualidade de Software

Corretude, eficiência, portabilidade, confiabilidade, usabilidade, modularidade, grau de reutilização e facilidade de manutenção (manutenibilidade) são características ligadas à qualidade de software (Boehm, Brown e Lipow, 1976; McCall, Richards e Walters, 1977; Basili, 1980). Provavelmente, em alguns casos, algumas dessas medidas se sobrepõem ou são antagônicas, como a eficiência e a portabilidade Mills (1998). Para se ter portabilidade pode-se abrir mão da eficiência, e exemplos como esse mostram que é difícil encontrar uma métrica que quantifique a qualidade globalmente. Métricas para a qualidade na fase de projeto também foram propostas há décadas (Myers, 1975; Yin e Winchester, 1978). Qualidade é uma característica que, nos modelos tradicionais de desenvolvimento de software, pode ser medida em cada fase do ciclo de desenvolvimento de software (Cerino, 1986), mas não serão abordadas métricas específicas do processo de desenvolvimento neste trabalho.

Os primeiros estudos sobre métricas demonstram mais relatos e experiências com métricas de dimensão e complexidade; posteriormente algumas áreas que tratam as características de qualidade de software citadas foram investigadas. A seguir serão apresentados alguns desse estudos:

- *Métricas de Funcionalidade*

Um estudo relevante na década passada sobre características da funcionalidade de software foi realizado pelo SUMI (*Software Usability Measurement Inventory*) (Kirakowski, Porteus e Corbett, 1992), definindo funcionalidade como a capacidade de um software ser usado com eficiência e satisfação para atingir objetivos específicos em um determinado ambiente. Pode-se destacar as seguintes características sobre a funcionalidade do software:

- Facilidade de aprendizagem: capacidade de um usuário atingir rapidamente um grau de proficiência elevada;

- Usabilidade: capacidade do software interagir amigavelmente com os usuários;
- Controle: capacidade de um produto em responder de uma forma natural e consistente aos comandos e entradas de dados fornecidos;
- Utilidade: capacidade de resolver ou ajudar a resolver problemas para os quais o software foi proposto;
- Eficiência: capacidade de resolver problemas de forma rápida e econômica.

Uma questão em aberto nesse tipo de métrica é como obter uma escala de medição com valores absolutos, o que as torna medidas com um grau de subjetividade.

- *Métricas de Manutenção Corretiva*

Erros encontrados no sistema e a sua eliminação rápida são pontos sensíveis na avaliação da qualidade do software. Métricas foram definidas para medir ou prever a manutenibilidade do software (Yau e Collofello, 1980, 1985), e também foram estudadas as possibilidades das métricas de Halstead e complexidade ciclomática para prever complexidade das tarefas de manutenção de software (Curtis et al., 1979; Harrison et al., 1982; Rombach, 1987; Kafura e Reddy, 1987). Entre os resultados estão tanto o fato de as métricas de complexidade poderem ser utilizadas de forma eficaz para explicar ou prever a manutenção de software em sistemas distribuídos quanto a relação observada entre 7 diferentes métricas de complexidade para medir as atividades de manutenção em um sistema que evoluiu durante 3 anos. Nesse segundo experimento foram observadas 3 versões diferentes do mesmo software, medindo-se a complexidade interna dos módulos de software e a complexidade das inter-relações entre os módulos de software. Entre as métricas objetivas citadas estão:

- Número de erros detectados no código;
- Número de erros detectados por aplicação de uma bateria de testes;
- Número de erros detectados pelos usuários;
- Tempo médio de resposta na resolução de problemas detectados;
- Número de mudanças de projeto;
- Número de alterações no código fonte.

- *Métricas de Confiabilidade*

São métricas para estimar e dimensionar o esforço de depuração, como, por exemplo, a quantidade de testes a aplicar, partindo das métricas de manutenção corretiva (Ruston, 1979; Basili, 1980; Musa, Iannino e Okumoto, 1987). Métricas típicas desse tipo são:

- MTTF - *Mean Time To Fail*: a probabilidade de uma falha ocorrer num intervalo de tempo especificado;
- MTBF - *Medium Time Between Failures*: o tempo médio entre falhas.

Existem modelos para descrever a ocorrência de erros em função do tempo, permitindo definir a confiabilidade e o MTTF. Partindo dos pressupostos:

- Entradas de teste são exemplos aleatórios de entradas no sistema;
- Todos os erros de software são observados;
- Intervalos de erros são independentes uns dos outros;
- MTBF é exponencialmente distribuído.

Assim, a seguinte equação foi definida:

$$d(t) = D(1 - (e)exp(-bct))$$

onde:

- D é o total de número de erros;
- b e c são constantes determinadas pelo tipo de software ou de um software similar;
- $d(t)$ é o número total de erros detectados no tempo t :

$$MTTF(t) = ((e)exp(bct))/cD$$

Determinar b , c e D , que não é uma tarefa óbvia, pode estabelecer o sucesso da aplicação desse modelo de métrica de confiabilidade (Ruston, 1979; Basili, 1980; Musa, Iannino e Okumoto, 1987).

Também existem estudos relacionando a confiabilidade com métricas de dimensão e complexidade, como as de Halstead e de complexidade ciclomática (Potier et al., 1982; Shen et al., 1985). Os trabalhos mais clássicos nessa área relacionam as métricas de complexidade com a manutenção (Curtis et al., 1979; Harrison et al., 1982; Rombach, 1987), medindo a complexidade em cada módulo e as inter-relações entre eles. A correlação entre a complexidade e a manutenção pode ser assim explorada no sentido de conseguir a redução dos custos de manutenção (Yau e Collofello, 1980, 1985; Kafura e Reddy, 1987; Brown et al., 1989; Takahashi e Kamayachi, 1989; Gorla, Benander e Benander, 1990).

2.3.2 Outras Métricas Tradicionais

Existem ainda outras métricas de software tradicionais razoavelmente difundidas além das mencionadas; elas não serão abordadas neste trabalho em detalhes porque se aproximam conceitualmente de alguma das anteriores detalhadas, ou ainda são correlacionadas como precursoras de algumas métricas que serão apresentadas dentro do contexto de orientação a objetivos na próxima seção. As mais comuns entre elas são:

- Dados Compartilhados (*BAM - Binding Among Modules*): mede o volume de dados compartilhados entre diferentes módulos do programa (Basili e Turner, 1975; Henry e Kafura, 1981).
- Tamanho Médio dos Módulos (*AML - Average Module Length*): mede o tamanho médio dos módulos que compõem o programa (Boehm et al., 1978).
- Condições e Operações (*COC - Conditions and Operations Count*): contabiliza os pares de todas condições e laços nas operações (Hansen, 1978).

- Razão de Coesão (*CRM - Cohesion Ratio Metrics*): mede a relação entre o número de módulos com coesão funcional e o número total de módulos (Yourdon e Constantine, 1979; Macro e Buxton, 1987; Bieman e Ott, 1994).
- Uso de Variável (*LVA - Live Variables*): mede o período em que cada variável é usada (Dunsmore e Gannon, 1979).
- Menor Número de Caminho (*MNP - Minimum Number of Paths*): mede o menor número de caminhos em um programa e seu alcance em qualquer nó (Schneidewind e Hoffmann, 1979).
- Métricas Morfológicas (*MOR - Morphology metrics*): mede características morfológicas de um módulo, como tamanho, profundidade, largura e razão entre aresta e nó. (Yourdon e Constantine, 1979).
- Complexidade de Fluxo de Controle e de Dados (*CDF - Control flow complexity and Data Flow complexity*): combinação de métricas baseadas na definição de variáveis e de referências cruzadas (Oviedo, 1980).
- Número de Funções (*FCO - Function Count*): mede o número de funções e as linhas de código das funções (Smith, 1980).
- Métrica de Complexidade Composta (*SSC - composite metric of Software Science and Cyclomatic complexity*): combina as métricas de Halstead com Complexidade Ciclomática (Baker e Zweben, 1980).
- Instruções no Código (*DSI - Delivered Source Instructions*): Conta declarações separadas numa mesma linha como distintas e ignora linhas de comentários (Boehm, 1981).
- Equivalência entre Módulos (*ESM - Equivalent Size Measure*): mede o percentual de modificações num módulo reutilizado (Boehm, 1981).
- Declarações Executáveis (*EST - Executable Statements*): contabiliza declarações separadas em uma linha de código como distintas, ignorando comentários, declarações de variáveis e cabeçalhos (Boehm, 1981).
- Níveis de Aninhamento (*NLE - Nesting Levels*): mede a complexidade pela profundidade de aninhamentos (Zolnowski e Simmons, 1981).
- Métricas de Peso Específico (*SWM - Specification Weight Metrics*): mede funções primitivas num diagrama de fluxo de dados (DeMarco, 1982).
- Distância de Árvore (*TRI - Tree Impurity*): determina o quanto um grafo se distanciou de uma árvore (Ince e Hekmatpour, 1988).
- Modularidade Global (*GLM - Global Modularity*): descrição da modularidade global em termos de várias visões específicas de modularidade (Hausen, 1989).
- Relação de Acoplamento (*COR - Coupling Relation*): sinaliza uma relação de todo par de módulos segundo o tipo de acoplamento (Fenton e Melton, 1990).

- Extensão de Reuso (*ERE - Extent of Reuse*): classifica um módulo segundo o grau de reuso (Gaffney, Felber e Erling, 1995).

2.4 Métricas de Orientação a Objetos

Além das métricas anteriormente discutidas, existem métricas voltadas especificamente a aspectos relevantes no contexto da programação orientada a objetos. As métricas apresentadas nesta seção foram selecionadas a partir de uma análise das métricas propostas para orientação a objetos especificamente (Xenos et al., 2000). É uma parte das mais conhecidas métricas, divididas 5 em categorias simples para um melhor entendimento do conceito das mesmas: métricas de classe, métricas de métodos, métricas de herança, métricas de acoplamento e métricas do sistema (características gerais do software orientado a objetos).

Métricas de Classes

- Encapsulamento dos Atributos (*AHF - Attribute Hiding Factor*): razão entre a soma de todos os atributos herdados de todas as classes do sistema em consideração ao número total de atributos das classes disponíveis (Morris, 1988).
- Métodos Reusados (*PMR - Percent of Potential Method uses actually Reused*): percentual de reuso dos métodos da classe (Morris, 1988).
- Coesão de Classe (*CCO - Class Cohesion*): mede a relação entre as classes (Chidamber e Kemerer, 1991).
- Número de Ancestrais (*NOA - Number Of Ancestors*): número total de ancestrais de uma classe (Kolewe, 1993).
- Ponderação do Tamanho da Classe (*WCS - Weighted Class Size*): número de ancestrais mais o tamanho total de métodos da classe (Kolewe, 1993).
- Respostas para uma Classe (*RFC - Response For a Class*): número de métodos dentre todos os métodos que podem ser invocados em resposta a uma mensagem enviada por um objeto de uma classe (Sharble e Cohen, 1993).
- Linhas de comentário por Método (*CLM - Comment Lines per Method*): mede o percentual de comentários no método (Lorenz e Kidd, 1994).
- Porcentagem de Métodos Comentados (*PCM - Percentage of Commented Methods*): percentual de métodos com comentários em uma classe (Lorenz e Kidd, 1994).
- Código Orientado a Função (*FOC - Function Oriented Code*): mede o percentual de código não orientado a objeto usado no programa (Lorenz e Kidd, 1994).
- Número de Métodos de Classe em uma Classe (*NCM - Number of Class Methods in a class*): mede os métodos disponíveis em uma classe, mas não em suas instâncias (Lorenz e Kidd, 1994).
- Número de Variáveis de Instância em uma Classe (*NIV - Number of Instance Variables in a class*): mede a relação de uma classe com outro objeto do programa (Lorenz e Kidd, 1994).

- Falta de Coesão entre Métodos (*LCM - Lack of Cohesion between Methods*): indica o nível de coesão entre os métodos (Chidamber e Kemerer, 1994).
- Privacidade Interna (*INP - Internal Privacy*): refere-se ao uso de funções de acesso à classe, incluindo as chamadas da própria classe (Chidamber e Kemerer, 1994) .
- Dados Públicos (*PDA - Public Data*): contabiliza o número de acessos aos dados protegidos e públicos de uma classe (McCabe, Dreyer e Watson, 1994).
- Percentual de Dados Públicos (*PPD - Percentage of Public Data*): é o percentual de dados públicos de uma classe (McCabe, Dreyer e Watson, 1994).
- Métodos ponderados por Classe (*WMC - Weighted Methods per Class*): soma ponderada de todos os métodos da classe (McCabe e Associates, 1994).
- Número de Parâmetros por Métodos (*NPM - Number of Parameters per Method*): número médio de parâmetros por método (Bansiya e Davi, 1997).
- Entropia de Complexidade da Classe (*CEC - Class Entropy Complexity*): mede a complexidade da classe, baseada no conteúdo de suas informações (Bansiya e Davi, 1997).
- Métrica de Acesso aos Dados (*DAM - Data Access Metric*): razão do número de atributos privados em relação ao número total de atributos declarados na classes (Bansiya e Davi, 1997).
- Medida de Abstração dos Atributos (*MAA - Measure of Attribute Abstraction*): razão do número de atributos herdados por uma classe em relação ao número total de atributos na classes (Bansiya e Davi, 1997).
- Medida de Abstração das Funções (*MFA - Measure of Functional Abstraction*): razão entre o número de métodos herdados por uma classe em relação ao número total de métodos acessíveis na classe (Bansiya e Davi, 1997) .
- Número de Tipos de Dados Abstratos (*NAD - Number of Abstract Data types*): número de objetos definidos pelo usuário como atributos de uma classe que são necessários para instanciar um objeto da referida classe (Bansiya e Davi, 1997).
- Número de Atributos Públicos (*NPA - Number of Public Attributes*): contabiliza o número de atributos declarados como públicos em uma classe (Bansiya e Davi, 1997).
- Número de Referências como Atributo (*NRA - Number of Reference Attributes*): contabiliza o número de ponteiros e referências passadas como atributos de uma classe (Bansiya e Davi, 1997).
- Encapsulamento dos Métodos (*MHF - Method Hiding Factor*): razão entre a soma de todos os métodos invisíveis em todas as classes em relação ao número total de métodos definidos em um determinado sistema (Harrison, 1998).

Métricas de Métodos

- Média de Complexidade dos Métodos (*AMC - Average Method Complexity*): soma da complexidade ciclomática de todos os métodos dividido pelo número total de métodos (Morris, 1988).
- MAG (*MAX V(G)*): complexidade ciclomática máxima dos métodos de uma classe (McCabe, Dreyer e Watson, 1994).
- Média do Tamanho dos Métodos (*AMS - Average Method Size*): mede o tamanho médio dos métodos do programa (Lorenz e Kidd, 1994).
- Complexidade do Método (*MCX - Method Complexity*): relaciona a complexidade com o número de mensagens (Lorenz e Kidd, 1994).

Métricas de Acoplamento

- Acoplamento Aferente (*AC - Afferent Coupling*): número total de classes externas de um pacote que dependem de classes de dentro desse pacote. Quando calculada no nível da classe, essa medida também é conhecida como Fan-in da classe, medindo o número de classes das quais a classe é derivada e, assim, valores elevados indicam uso excessivo de herança múltipla (McCabe, Dreyer e Watson, 1994).
- Acoplamento Eferente (*EC- Efferent Coupling ou EC*): número total de classes dentro de um pacote que dependem de classes externas ao pacote. Quando calculada no nível da classe, essa medida também é conhecida como Fan-out da classe, ou como Acoplamento entre Objetos (*CBO - Coupling Between Objects*) na família de métricas de CK (Chidamber-Kemerer) (Chidamber e Kemerer, 1994).
- Acoplamento de Classe (*CP - Class Coupling*): mede as relações entre as classes com base em suas trocas de mensagens (Basili, Briand e Melo, 1996).
- Fator de Acoplamento (*CFA - Coupling Factor*): razão entre o número máximo possível de acoplamentos no sistema e o número atual de acoplamento possíveis por herança (Harrison, 1998).

Métricas de Herança

- Construção Efetiva (*FEF - Factoring Effectiveness*): número de métodos únicos dividido pelo número total de métodos (Morris, 1988).
- Potencial de Métodos Sobrescritos (*PMO - Percent of Potential Method uses Overridden*): percentual de métodos sobrescritos utilizados (Morris, 1988).
- Nível de Aninhamento Hierárquico da Classe (*HNL - Class Hierarchy Nesting Level*): mede a profundidade de hierarquia em que cada classe está localizada (Lorenz, 1991).
- Métricas de Reuso de Método (*MRE - Method Reuse metrics*): indica o nível de reuso dos métodos (Abreu e Carapuça, 1994).
- Número de Métodos Herdados (*NMI - Number of Methods Inherited*): mede o número de métodos que uma classe herda (Lorenz e Kidd, 1994).

- Medida de Polimorfismo (*PFA - Polymorphism Factor*): razão entre o número atual de possibilidades de polimorfismo diferentes de uma classe e o número máximo de possíveis polimorfismos distintos da referida classe (Abreu, 1994).
- Número de Métodos Sobrescritos (*NMO - Number of Methods Overridden*): número de métodos redeclarados pela classe herdeira (Lorenz e Kidd, 1994).
- Tipo de Especialização (*SIX - Specialisation Index*): indica o tipo de especialização (Lorenz e Kidd, 1994).
- Medida de Herança de Atributos (*AIF - Attribute Inheritance Factor*): razão entre a soma dos atributos herdados em todas as classes do sistema e o número total de atributos disponíveis na classe (Basili, Briand e Melo, 1996).
- Profundidade da Árvore de Herança (*DIT - Depth of Inheritance Tree*): mede o número de ancestrais de uma classe (Shih et al., 1997).
- Número de Filhos (*NOC - Number Of Children*): número total de filhos de uma classe (Rosenberg e Hyatt, 1997).
- Proporção de Reuso (*RER - Reuse Ratio*): razão entre o número de superclasses dividida pelo número total de classes (Rosenberg e Hyatt, 1997).
- Proporção de Especialização (*SPR - Specialisation Ratio*): razão entre o número de subclasses dividido pelo número de superclasses (Rosenberg e Hyatt, 1997).
- Medida de Herança de Método (*MIF - Method Inheritance Factor*): razão entre a soma dos métodos herdados em todas as classes e o número total de métodos disponíveis em todas as classes (Harrison, 1998).
- Proporção entre Profundidade e Largura (*RDB - Ratio between Depth and Breadth*): razão entre a profundidade e a largura da hierarquia de classes (Bellin, Tyagi e Tyler, 1999).

Métricas do Sistema

- Granularidade da Aplicação (*APG - Application Granularity*): número total de objetos dividido pelo número total de pontos de função (Morris, 1988).
- Eficiência da Biblioteca de Objeto (*OLE - Object Library Effectiveness*): razão entre o número total de objetos reusados e o número total de objetos da biblioteca (Morris, 1988).
- Complexidade Associada (*ASC - Association Complexity*): mede a complexidade da estrutura associada ao sistema (Kolewe, 1993).
- Número de Hierarquias (*NOH - Number Of Hierarchies*): número de hierarquias distintas do sistema (Kolewe, 1993).
- Reuso de Classe (*CRE - Number of time a Class is Reused*): mede as referências a uma classe e o número de aplicações que reusam tal classe (Abreu e Carapuça, 1994).

- Número de Rejeição de Classe (*NCT - Number of Classes Thrown away*): mede o número de vezes que uma classe é rejeitada até que seja aceita (West, 1992; Lorenz e Kidd, 1994).
- Problemas Relatados por Classe (*PRC - Problem Reports per Class*): mede os relatos de erros da classe (Lorenz e Kidd, 1994).
- Reuso do Sistema (*SRE - System Reuse*): percentual de reuso de classes (Abreu e Carapuca, 1994).
- Categorização (*CAN - Category Naming*): divide as classes em um conjunto semanticamente significativo (Chidamber e Kemerer, 1994).
- Profundidade Média de Herança (*ADI - Average Depth of Inheritance*): divisão entre a soma de níveis de aninhamento de todas as classes pelo número de classes (Bansiya e Davi, 1997).
- Média de Ancestrais (*ANA - Average Number of Ancestors*): determina o número médio de ancestrais de todas as classes (Bansiya e Davi, 1997).
- Densidade Funcional (*FDE - Functional Density*): razão entre o número de linhas de código e pontos de função (Fenton e Pfleeger, 1998).
- Modificação de Objetos Reusados (*PRO - Percent of Reused Objects Modified*): percentual de objetos reusados que foram modificados (Bellin, Tyagi e Tyler, 1999).

2.5 Medidas Aplicadas aos Métodos Ágeis

Foram listadas e definidas algumas medidas para auxiliar o acompanhamento de uma equipe ágil (Sato e Goldman, 2007), de acordo com abordagens para escolha das melhores métricas possam ser utilizadas nas metodologias ágeis. Entre métricas apresentadas estão exemplos das que podem ser coletadas de forma automatizada, por serem compostas de medidas quantitativas e objetivas. Dessas, algumas já foram explicadas neste trabalho, como:

- Linhas de Código;
- Complexidade Ciclométrica;
- Métodos Ponderados por Classe;
- Falta de Coesão dos Métodos;
- Profundidade da Árvore de Herança;
- Número de Filhos;
- Acoplamento Aferente;
- Acoplamento Eferente.

Outras medidas objetivas, não citadas neste trabalho, foram apresentadas em (Sato e Goldman, 2007) como úteis para prover o uso eficaz de métricas de software no desenvolvimento com metodologias ágeis, como a seguir:

- Total de Linhas de Código de Teste (TLCT): representa o número total de pontos de teste do sistema (Dubinsky et al., 2005). Um ponto de teste considerado como um passo do cenário de um teste de aceitação, automatizado ou como uma linha de código de teste de unidade automatizado, descartando linhas em branco e comentários.
- Número de Linhas Alteradas: representa o número de linhas (não apenas código-fonte) adicionadas, removidas e atualizadas no Repositório de Código Unificado.
- Número de Commits representa o número de commits efetuados no Repositório de Código Unificado.
- Estimativas Originais: representa o total de pontos (ou horas) originalmente estimadas pela equipe para todas as Histórias da iteração.
- Estimativas Finais: representa o total de pontos (ou horas) efetivamente reportadas como gastas para implementar as Histórias da iteração.
- Número de Histórias Entregues: representa o número total de Histórias implementadas e aceitas pelo cliente.
- Número de Pontos Entregues: representa o número total de pontos implementados e aceitos pelo cliente.
- Tempo: utilizado no cálculo de diversas métricas, pode ser utilizado como duração (em meses, semanas, dias, etc.) ou como número de iterações.
- Tamanho da Equipe: geralmente utilizado no cálculo de métricas, representa a quantidade de pessoas na equipe.
- Esforço: uma medida que leva em conta o tamanho da equipe durante um certo intervalo de tempo, geralmente medido em pessoas-mês ou pessoas-ano.

Capítulo 3

Aspectos Relevantes e Métricas de Qualidade para Adoção de um Software Livre

Com o objetivo de realizar uma ampla pesquisa foram elencados 106 aspectos e métricas que podem influenciar na avaliação da qualidade de um software livre no momento de adotá-lo em uma organização. A seleção dessas métricas foi baseada no estudo do estado da arte de métricas de software (apresentado no capítulo 2), em relatórios de pesquisas sobre métricas e artefatos que influenciam a confiabilidade da indústria de software européia na adoção de software de código aberto (Bianco et al., 2007, 2008,?,?) e também uma triagem inicial com especialistas em desenvolvimento de software, mestres e doutores na área de desenvolvimento de sistemas, principalmente os envolvidos com as metodologias ágeis e o software livre.

3.1 Métricas e Aspectos Selecionados

Para que a medição seja efetiva, torna-se necessário focá-la nos objetivos a serem alcançados (Basili, Caldiera e Rombach, 1994), sendo preciso deixar claras as necessidades para poder melhor conhecer os objetivos das medições (Pfleeger, 2000). O uso de modelos pode guiar a definição de aplicação de medições a serem realizadas (Pfleeger, 2000), um modelo comumente utilizado é o GQM – *Goal Question Metrics* (Basili, Caldiera e Rombach, 1994). A abordagem do método GQM é baseada na premissa de que para medir deve primeiro especificar os seus próprios objetivos e os objetivos de seus projetos, então deve-se traçar os objetivos para os dados que os definem operacionalmente, e finalmente prover um arcabouço para interpretação dos dados, respeitando os objetivos estabelecidos (Basili, Caldiera e Rombach, 1994). O GQM considera um modelo com 3 níveis:

- Conceitual – nível no qual são definidos os objetivos de medição. O objetivo é definido para um objeto (produto, processo ou recurso).
- Operacional – nível no qual são definidas questões para caracterizar um caminho para alcançar um determinado objetivo.
- Quantitativo – nível no qual são definidas as métricas que usam um conjunto de dados (objetivos ou subjetivos) associados com cada questão de forma quantitativa.

A fases de definição do GQM são (Soligen e Berghout, 1999):

- Planejamento: fase na qual é definida a equipe GQM, escolhida a área de melhoria e são preparadas e motivadas as pessoas envolvidas.
- Definição: fase na qual são definidos e documentados os objetivos, questões e métricas;
- Coleta de dados: fase na qual os dados são coletados;
- Interpretação: fase na qual os dados coletados são processados em relação às métricas definidas, gerando os resultados da medição, que provêem respostas para as questões definidas e posteriormente os objetivos podem ser avaliados.

O método GQM foi usado pelos trabalhos de levantamento de métricas para adoção de software de código aberto (Bianco et al., 2007, 2008,?,?), a abordagem não foi replicada nesse trabalho, uma vez que foi detalhadamente descritas nos relatórios citados. As demais métricas elencadas fazem parte do contexto das métricas software tradicionais e novos aspectos considerados relevantes por experientes especialistas e colaboradores de projeto de software livre.

Na próximas seções serão apresentados os aspectos e métricas selecionados para se obter opinião de especialistas em desenvolvimento de software. Serão apresentados em 16 grupos, de acordo com suas características.

Código-Fonte

Métricas de código-fonte, em sua maioria, são objetivas e passivas de automatização. Elas têm o intuito de medir os detalhes de implementação do software e as relações entre seus módulos (classes etc). Foram elencadas 24 métricas de código-fonte neste trabalho:

1. Número total de linhas de código
2. Número de classes (ou arquivos ou módulos)
3. Número de métodos ou funções
4. Número de linhas de teste
5. Número de testes por linha de código
6. Número de testes por método ou função
7. Cobertura dos testes
8. Nível de acoplamento (dependência entre os módulos/classes)
9. Nível de coesão entre módulos/classes do sistema
10. Complexidade Ciclomática
11. Existência de código duplicado: existindo código duplicado no fonte do sistema, pode indicar problemas na modelagem do sistema, definição das classes, métodos ou externar a falta de experiência do desenvolvedor.

12. Separação em Módulos
13. Estruturas de dados utilizadas
14. Estilo uniforme de indentação
15. Número de colunas por linha não muito grande: por exemplo, ausência de linhas com mais de 120 colunas.
16. Número de linhas em cada método
17. Número de linhas em cada classe
18. Número de métodos em cada classe
19. Número de atributos em cada classe
20. Número de variáveis locais em cada método
21. Padrões de nomenclatura usados uniformemente
22. Uso de bons nomes para classes, métodos, variáveis, etc.
23. Existência de comentários no código (cobertura e distribuição no código)
24. Módulo para configurações (sem configurações *hard-coded* e limites espalhados pelo código): o sistema deve possuir um módulo de configuração de maneira a agrupar e possibilitar configurações. Deixando o software mais flexível e de mais fácil manutenção, por permitir reconfigurações.

Testes

Testes é um conjunto de atividades já bem difundidas nos processos de desenvolvimento de software, ganhando mais ênfase nos métodos ágeis. A natureza dessas atividades buscam mecanismos de automização para melhor garantir a segurança dos resultados dos testes. Foram selecionados 11 métricas e aspectos de testes neste trabalho:

1. Existência de testes automatizados
2. Uso de um arcabouço de testes (ex: JUnit, CppUnit, Selenium, TestNG, DejaGNU, etc.)
3. Publicação dos resultados dos testes
4. Existência de um subgrupo (comunidade ou pessoa) especializado em testes
5. Tipos de teste disponíveis (unidade, funcional, desempenho, estresse, estrutural, etc.)
6. Testes de mutação: são construídas versões do software com falhas para saber se uma determinada bateria de testes vai identificar tais falhas.
7. Testabilidade do código (facilidade de escrever testes): capacidade de observar e controlar os testes.
8. Existência de testes de desempenho (benchmarks)

9. Estudo sistemático sobre o tempo de resposta
10. Estudo sobre o consumo de recursos (disco, memória, etc.)
11. Artigos e relatos dos experimentos de desempenho

Manutenibilidade

A manutenibilidade é a facilidade de se dar manutenção em um software. A manutenção é composta por um conjunto de atividades requeridas para prover suporte de custo efetivo para um sistema de software. As métricas envolvem aspectos que podem ser automatizados e outros que necessitam de uma abordagem subjetiva. Assim, foram elencados 11 aspectos e métricas de manutenibilidade neste trabalho:

1. Orientações documentadas para adaptação/extensão
2. Documentação/descrição da arquitetura
3. Uso de padrões de projeto e arquiteturais
4. Uso de boas práticas de desenvolvimento (documentadas ou difundidas)
5. Existência de versões/atualizações de manutenção
6. Existência de organização/empresa fornecedora de suporte
7. Tipo de manutenção/suporte oferecida
8. Linguagem de programação usada
9. Paradigma de programação (procedimental, orientado a objetos, etc.)
10. Número de linguagens usadas
11. Ambiente (plataforma) de desenvolvimento

Interoperabilidade

A interoperabilidade é uma característica que atribui a capacidade do sistema em se integrar e comunicar com outros sistemas e criação de novos e/ou adaptação de novos componentes que se integram com o mesmo software em questão. Neste trabalho foram pensados em 3 aspectos ligados à interoperabilidade.

1. Capacidade de comunicação/integração com outros sistemas
2. Suporte a extensão através de plug-ins
3. Uso de padrões da indústria e protocolos bem disseminados

Portabilidade

A portabilidade é a possibilidade e capacidade de um software funcionar em diferentes ambientes ou plataformas. O conceito no contexto de software livre pode ganhar mais amplitude uma vez que a própria comunidade regula as suas necessidades, mantendo determinado software para as plataformas requeridas pelos próprios membros da comunidade. Foram elencados 2 aspectos de portabilidade:

1. Lista de ambientes para os quais é oferecido suporte
2. Linguagem facilmente portátil

Usabilidade

Neste trabalho conceituamos a usabilidade com o esforço necessário do usuário utilizar o software, desde a sua instalação até como obter ajuda para aprender a lidar com as funcionalidades do software. Foram selecionados 5 aspectos relacionados com a usabilidade:

1. Facilidade de instalação e configuração
2. Clareza da interface do usuário
3. Assistentes e ajuda ao usuário embutidos no software
4. Diferentes opções de uso (linha de comando, modo gráfico, acesso Web, etc.)
5. Suporte a localização/internacionalização (adição de novas línguas)

Ferramentas

A comunidade de software livre é distribuída e necessita de ferramentas de apoio ao desenvolvimento de software. Foram elencados 4 aspectos referentes as características de ferramentas:

1. Integração com ferramentas automatizadas de desenvolvimento
2. Scripts para configuração do processo de build
3. Possibilidade de personalização já integrada
4. Controle de versões com commits pequenos e documentados (comentários nos commits)

Independência

Um conceito aplicado aos sistemas de software livre é a separação em módulos e a criação de pacotes de software. Os mesmos mantêm um nível de dependências entre seus outros módulos e/ou outras aplicações. Da mesma forma, um pacote de software deve se integrar ao ambiente onde será inserido. Foram pensados 2 aspectos relacionados à independência de um software livre.

1. Dependências bem documentadas
2. Não requer instalação de outros pacotes/sistemas

Comunidade

Um grupo de usuários, desenvolvedores, mantenedores entre outros formam uma comunidade de um software livre. Essas comunidades trabalham de maneira que é possível se observar características e medidas para se medir as atividades de uma determinada comunidade de software livre, podendo ser aspectos importantes na escolha de adoção de um determinado sistema. Foram pensados 12 aspectos e métricas relacionadas a comunidade:

1. Tamanho atual da comunidade de usuários do software
2. Número total de componentes de atualizações (patches) e versões (releases)
3. Número de componentes de atualizações (patches) e versões (releases) recentes
4. Tempo de vida do projeto
5. Número de contribuições da comunidade
6. Existência de diferentes grupos na comunidade
7. Existência de lista de e-mail ou Fórum de usuários
8. Atividade recente na lista de e-mail ou Fórum da comunidade
9. Lista de e-mail ou Fórum de desenvolvedores
10. Número de mensagens que já foram mandadas para fóruns, listas, blogs, etc.
11. Número de desenvolvedores
12. Reputação dos desenvolvedores

Aspectos de Confiança

O conceito de confiança neste trabalho está vinculado às dinâmicas das comunidade de software livre na solução de problemas e disponibilização de informações. Foram pensados em 6 aspectos de confiança neste trabalho.

1. Estágio do desenvolvimento/maturidade
2. Informações sobre a complexidade do software
3. Disponibilização de atualizações críticas (patch releases)
4. Números de erros (bugs) críticos reportados
5. Tempo de solução de erros críticos (bugs)
6. Reputação do software

Funcionalidades

A idéia de funcionalidade neste trabalho não está restrita a uma lista de requisitos de software, mas sim aos mecanismos de validação do software e uso em ambiente de produção. Foram pensados 5 aspectos ligados ao quão funcional está um determinado software livre:

1. Disponibilidade de uma lista bem definida de funcionalidades
2. Informações/descrições sobre cada nova versão (p.ex., changelog)
3. Protótipo/exemplo/demo do software
4. Existência de comparações documentadas com produtos similares
5. Existência de produtos derivados

Satisfação de Usuários/Clientes

O retorno das opiniões dos usuários de um determinado de software livre interfere diretamente na reputação e recomendação do sistema. Assim, foram pensados 4 aspectos relacionados com a satisfação dos usuários/clientes do software:

1. Existência de lista de organizações que o utilizam e testemunho de usuários
2. Existência de estudos de casos documentados e histórico de uso
3. Prêmios ganhos pelo software
4. Empresas oferecendo serviços tendo o software como base

Documentação

A documentação no contexto deste trabalho não está vinculado aos artefatos do processo de desenvolvimento de software, estando mais relacionadas aos manuais e documentação técnica do software. Assim, foram elencados 6 tipos de documentação de software.

1. Manual do usuário
2. Manual técnico
3. FAQ para usuários
4. FAQ técnico
5. Documentação técnica (p.ex., Javadoc)
6. Guia de instalação

Suporte

O suporte está vinculado aos aspectos de correção do erros. Assim, foram pensados 4 métricas ligadas ao suporte de um software livre:

1. Número de correções de erros (*bugs*) disponibilizadas
2. Acompanhamento de erros (*bug tracking*)
3. Processo definido para correção de erros (*bug workflow*)
4. Tempo de resposta quando detectado um erro pelo usuário (comunidade e/ou desenvolvedores)

Treinamento

Foram pensando 2 aspectos de treinamento vinculados a um determinado software livre:

1. Disponibilidade de treinamentos, tutoriais e outros materiais de aprendizagem
2. Curso oficial de treinamento

Canais de Distribuição e Licença

Os meios de acesso ao software e ao código-fonte e as licenças de um determinado software podem influenciar na decisão de adoção de um determinado software livre. Assim, foram selecionados 5 aspectos ligados aos canais de distribuição e as licenças de software:

1. Acesso ao repositório
2. Download do código-fonte e binários
3. Disponibilização em formato físico (CD/DVD)
4. Principal licença usada
5. Possibilidade de identificar uma lista de todas as licenças de todos os componentes (árvore de licenças)

Capítulo 4

Resultados do Levantamento de Métricas de Avaliação de Projetos de Software Livre

A pesquisa sobre os aspectos e métricas para adoção de software livre obteve, além dos dados sobre relevância dos aspectos e métricas, um levantamento do perfil dos entrevistados, da atuação dos mesmos em projetos de software livre e participação e relação das organizações e empresas, que os entrevistados pertencem, com os sistemas livres disponíveis. Neste capítulo serão apresentados o perfil dos especialistas em desenvolvimento de software, vinculados ou não aos projetos de software livre, e o resumos de suas opiniões sobre os 106 aspectos e métricas elencados no questionário disponível na íntegra no Anexo A deste trabalho. Nem todas as perguntas e itens eram indicados como obrigatórios de respostas, podendo ocorrer variações no número absolutos de respostas.

4.1 Perfil dos Entrevistados

Os especialistas em desenvolvimento de software convidados para responder o questionário são desenvolvedores reconhecidos por sua atuação em projeto de software livre; mestres, doutores e pesquisadores na área de sistemas de software das mais reconhecidas universidades brasileiras; e profissionais de empresas de desenvolvimento de software. Foi levada em consideração para o convite a reputação dos desenvolvedores perante os demais profissionais e comunidade de software livre e acadêmica, dependendo do contexto de atuação de cada indivíduo.

Atuação Profissional em Relação ao Desenvolvimento de Software

Foi perguntado aos especialistas em desenvolvimento de software qual era o tipo de atuação profissional na área de desenvolvimento de software, independente de ser livre ou proprietário, para se ter conhecimento das diversas experiências de atuação dos entrevistados. A Figura 4.1 mostra o resumo das respostas, sendo possível assinalar mais de um tipo de atuação. Destacam-se como as mais indicadas a atuação como desenvolvedor, usuário, líder de projeto e mantenedor.

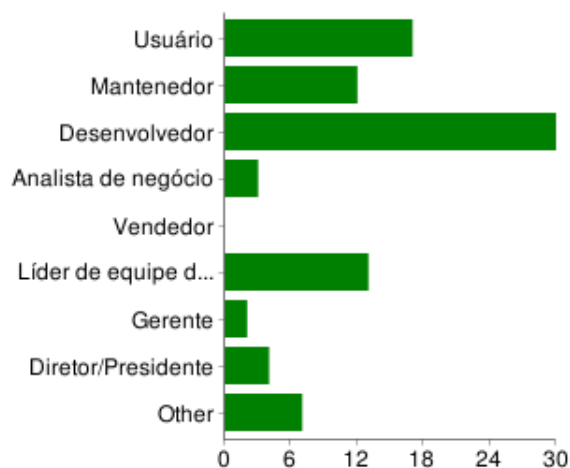


Figura 4.1: Atuação profissional em relação à software

Foram 34 pessoas que responderam este item, resultando nos seguintes números absolutos e relativos listados abaixo:

- Usuário foi assinalado por 17 pessoas, correspondendo a 50%;
- Mantenedor marcado por 12 pessoas, correspondendo a 35%;
- Desenvolvedor indicado por 30 pessoas, correspondendo a 88%;
- Analista de negócio marcado por 3 pessoas, correspondente a 9%;
- Vendedor com nenhuma indicação de resposta;
- Líder de equipe de desenvolvimento assinalado por 13 pessoas, corresponde a 38%;
- Gerente indicado por 2 pessoas, corresponde a 6%;
- Diretor/Presidente marcado por 4 pessoas, corresponde a 12% das respostas;
- Outros foi indicado por 7 pessoas, corresponde a 21%. Entre as outras opções de resposta estão Consultor e Analista de Testes.

Participação em Projetos de Software Livre

Também foi perguntando aos especialistas em desenvolvimento de software de qual maneira que já participou de algum projeto de software livre. A Figura 4.2 mostra o resumo das respostas, podendo ser assinalada mais de uma resposta. Usuário e Desenvolvedor foram os tipos de participação mais indicados como resposta.

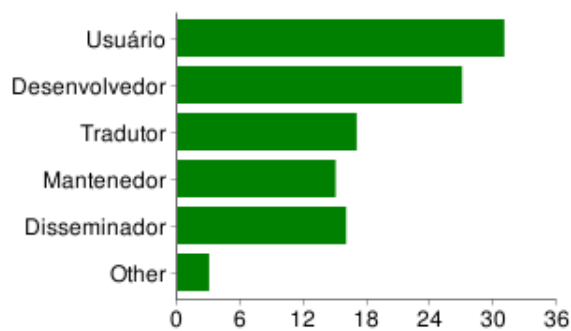


Figura 4.2: Tipo de participação em projeto de software livre

O total de pessoas que responderam este item foram 34, sendo os valores absolutos e relativos conforme mostrado abaixo:

- Usuário foi indicado por 31 pessoas, correspondendo a 91%;
- Desenvolvedor marcado por 27 pessoas, correspondendo a 79%;
- Tradutor indicado por 17 pessoas, corresponde a 50%;
- Mantenedor assinalado por 15 pessoas, correspondente a 44%;
- Disseminador indicado por 16 pessoas, corresponde a 47%;
- Outros foi marcado por 3 pessoas, corresponde a 9%. Entre os outros tipo de participação está o de Gerente de Projeto e Testador.

Formação

O nível formação escolar e acadêmica dos entrevistados também foi levantado. A Figura 4.3 apresentada o resumo das respostas, indicando uma variação de formação entre superior incompleta ao doutorado.

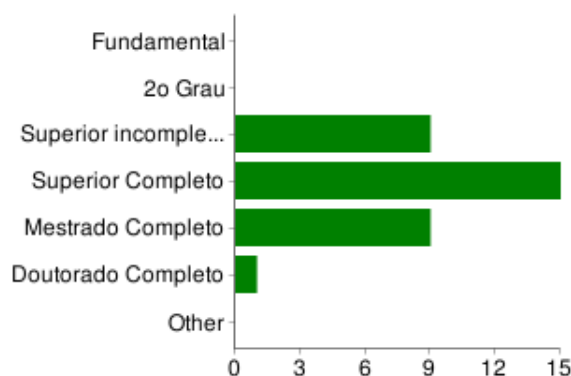


Figura 4.3: Nível de formação dos entrevistados

As 34 respostas em relação à formação apresentam os seguintes valores absolutos e relativos:

- Fundamental, Segundo Grau e Outras Opções não tiveram indicações.
- Superior incompleto foi marcado por 9 pessoas, corresponde a 26%;

- Superior Completo indicado por 15 pessoas, corresponde a 44%;
- Mestrado Completo indicado por 9 26%, corresponde a 26%;
- Doutorado Completo foi marcado 1 pessoa, corresponde a 3%;

Tempo de Experiência Profissional com Software

Também foi perguntado aos especialistas em desenvolvimento de software qual é a experiência profissional na área de desenvolvimento de software, independente de ser livre ou proprietário, para se ter conhecimento das diversas experiências de mercado dos entrevistados. A Figura 4.4 mostra o resumo das respostas, sendo possível a indicação de um único intervalo de tempo em anos. Todos indicam pelo menos 3 anos de experiência com desenvolvimento de software, mas a maioria com mais de 5 anos.

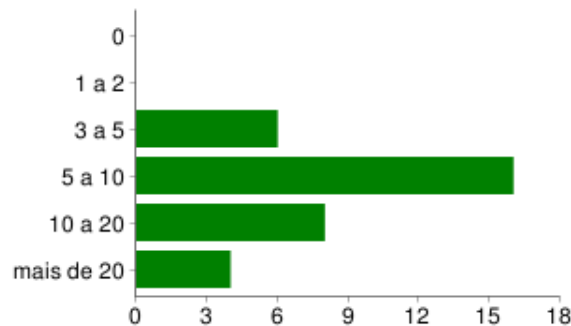


Figura 4.4: Experiência profissional (em anos)

Os valores absolutos e relativos das 34 respostas indicadas foram como a seguir:

- Nenhuma experiência e 1 a 2 anos não foram marcadas como resposta;
- 3 a 5 anos de experiência foi indicado por 6 pessoas, corresponde a 18%;
- 5 a 10 anos de experiência foi indicado por 16 pessoas, corresponde a 47%;
- 10 a 20 anos de experiência foi indicado por 8 pessoas, correspondente a 24%;
- mais de 20 anos de experiência foi indicado por 4 pessoas, corresponde a 12%.

Número de Projetos de Software Livre Participante

Também foi perguntado aos especialistas em desenvolvimento de software qual é o número de projetos de software livre que participa ou participou até o momento. A Figura 4.5 mostra o resumo das respostas, sendo possível a indicação de um único intervalo de unidades. Alguns indicaram nenhuma participação direta, mas a maioria indicou que participou de pelo menos 3 projeto.

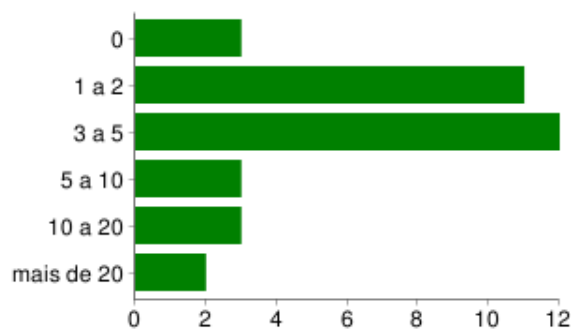


Figura 4.5: Número de projeto de software livre participante

Foram 34 resposta para este item, sendo os valores absolutos e relativos como detalhados abaixo:

- 3 pessoas indicaram nenhuma participação direta em projetos de software livre, ou seja 9%;
- 1 a 2 projetos foi indicado por 11 pessoas, corresponde a 32%;
- 3 a 5 projetos foi indicado por 12 pessoas, corresponde a 35%;
- 5 a 10 projetos foi indicado por 3 pessoas, corresponde a 9%;
- 10 a 20 projetos foi indicado por 3 pessoas, corresponde a 9%;
- mais de 20 projetos foi indicado por 2 pessoas, correspondente a 6%.

Tempo de Atuação em Projetos de Software Livre

Também foi perguntado aos especialistas em desenvolvimento de software qual é o tempo de experiência e atuação em projetos de software livre. A Figura 4.6 mostra o resumo das respostas, sendo possível a indicação de um único intervalo de tempo em anos. Alguns indicaram nenhuma participação direta, mas a maioria indicou pelo menos 3 anos de participação.

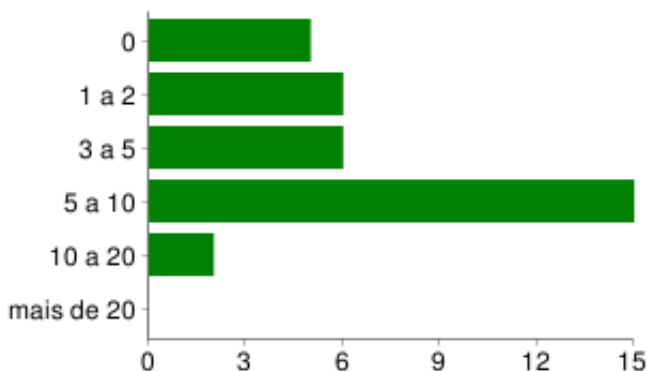


Figura 4.6: Tempo de participação em projetos de software livre (em anos)

Foram obtidas 34 respostas, conforme detalhadas abaixo:

- 5 pessoas nunca participaram de um projeto de software livre diretamente, correspondem a 15%;
- 1 a 2 anos de participação em projetos foi indicado por 6 pessoas, corresponde a 18%;

- 3 a 5 anos de participação em projetos foi indicado por 6 pessoas, correspondente a 18%;
- 5 a 10 anos de participação em projetos foi indicado por 15 pessoas, corresponde a 44%;
- 10 a 20 anos de participação em projetos foi indicado por 2 pessoas, corresponde a 6%;
- mais de 20 anos de participação em projetos não foi assinalada como resposta.

Tempo de Existência do Projeto de Software Livre e Número de Participantes

Aos especialistas em desenvolvimento de software que atuam em um projeto de software livre, foi solicitado a indicação do nome do principal projeto que colabora, o tempo de existência do projeto, o número total de participantes e o número de desenvolvedores que o especialistas interage. A Figura 4.7 mostra o resumo das respostas em relação ao tempo de existência do projeto que participa.

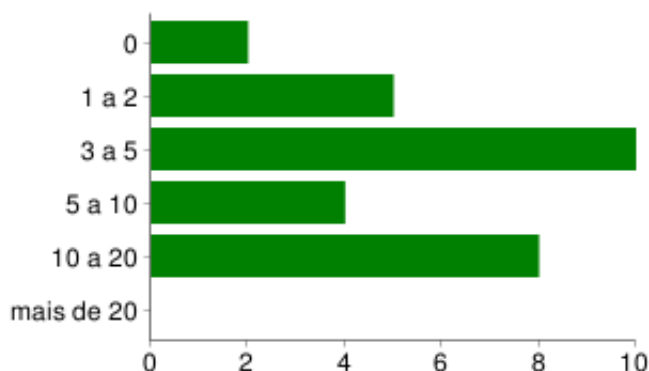


Figura 4.7: Tempo de existência do software livre participante

Foram 29 respostas para este item, com valores absolutos e relativos conforme abaixo:

- menos de 1 ano de existência foi indicado por 2 pessoas, corresponde a 7%;
- 1 a 2 anos de existência foi indicado por 5 pessoas, corresponde a 17%;
- 3 a 5 anos de existência foi indicado por 10 pessoas, corresponde a 34%;
- 5 a 10 anos de existência foi indicado por 4 pessoas, corresponde a 14%;
- 10 a 20 anos de existência foi indicado por 8 pessoas, corresponde a 28%;
- mais de 20 anos de existência não foi assinalado como resposta.

O número de participantes do projeto e o número de desenvolvedores que os especialistas interagem foram resumidos na Tabela 4.1.

Participantes do Projeto	Total (votos)	%	Interagem (votos)	%
1 a 2	3	10%	5	17%
3 a 10	8	28%	12	41%
11 a 50	5	17%	7	24%
51 a 100	1	3%	1	3%
101 a 500	5	17%	3	10%
mais de 500	7	24%	1	3%

Tabela 4.1: Números correspondentes aos participantes do projeto

A Tabela 4.1 indica que a maioria dos entrevistados colaboram com projetos com 3 a 10 participantes e também interagem em média com esse intervalo de participantes.

Número de usuários

Foi perguntado aos entrevistados o número de usuários do principal projeto de software livre que colabora. A Figura 4.8 resume as respostas para este item, destacando a indicação de milhares de usuários para a maioria dos projetos em questão.

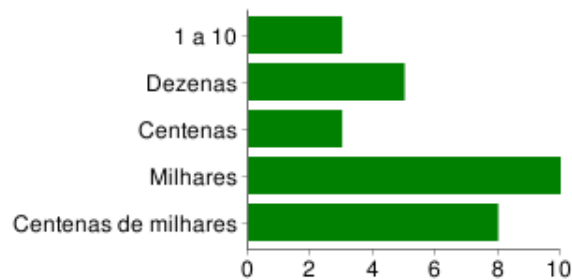


Figura 4.8: Número de usuários do software

Os valores absolutos e relativos das 29 respostas para este item foram:

- 1 a 10 usuários do projeto foi indicado por 3 pessoas, corresponde a 10%;
- Dezenas de usuários do projeto foi indicado por 5 pessoas, corresponde a 17%;
- Centenas de usuários do projeto foi indicado por 3 pessoas, corresponde a 10%;
- Milhares de usuários do projeto foi indicado por 10 pessoas, corresponde a 34%;
- Centenas de milhares de usuários do projeto foi indicado por 8 pessoas, corresponde a 28%.

Número de Downloads

Também foi perguntado aos entrevistados o número de downloads do principal projeto de software livre que colabora. A Figura 4.9 resume as respostas para este item, destacando a indicação de centenas de milhares de downloads para a maioria dos projetos em questão.

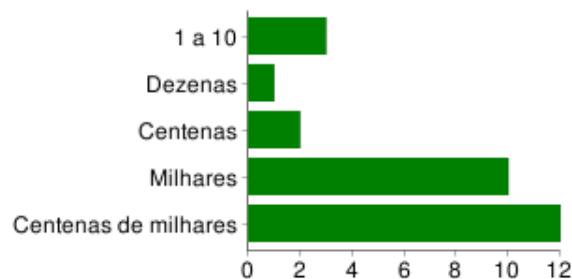


Figura 4.9: Número de downloads

Foram 28 respostas para este item, conforme detalhado abaixo:

- 1 a 10 downloads foi indicado por 3 pessoas, correspondendo a 11%;

- Dezenas de downloads foi indicado por 1 pessoa, corresponde a 4%;
- Centenas de downloads foi indicado por 2 pessoas, correspondendo a 7%;
- Milhares de downloads foi indicado por 10 pessoas, corresponde a 36%;
- Centenas de milhares de downloads foi indicado por 12 pessoas, corresponde a 43%.

Tempo de Participação no Projeto de Software Livre

Foi pedido aos especialistas a indicação do tempo de participação, em anos, do principal projeto de software livre que colabora. A Figura 4.10 mostra as respostas, destacando a opção de pelo menos 3 anos de participação no referido projeto.

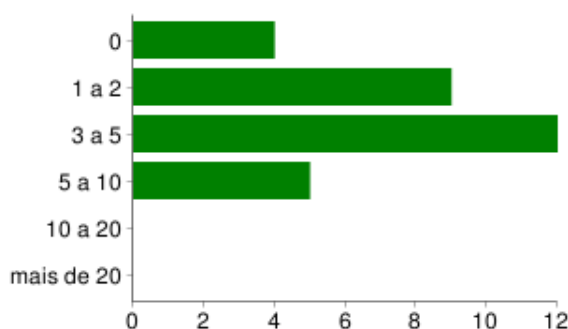


Figura 4.10: Tempo participação (em anos)

As 30 respostas para este item tiveram os seguintes valores relativos e absolutos:

- menos de 1 ano de participação foi indicado por 4 pessoas, corresponde a 13%;
- 1 a 2 anos de participação foi indicado por 9 pessoas, corresponde a 30%;
- 3 a 5 anos de participação foi indicado por 12 pessoas, corresponde a 40%;
- 5 a 10 anos de participação foi indicado por 5 pessoas, corresponde a 17%;
- 10 a 20 anos de participação não foi assinaladas como resposta.
- mais de 20 anos de participação não foi assinalado com resposta.

Número de Horas Semanais Dedicadas ao Projeto

Também foi pedido aos especialistas a indicação do número horas dedicadas ao principal projeto de software livre que colabora. A Figura 4.11 mostra as respostas, destacando a opção de pelo menos 20 horas semanais com a segunda mais indicada.

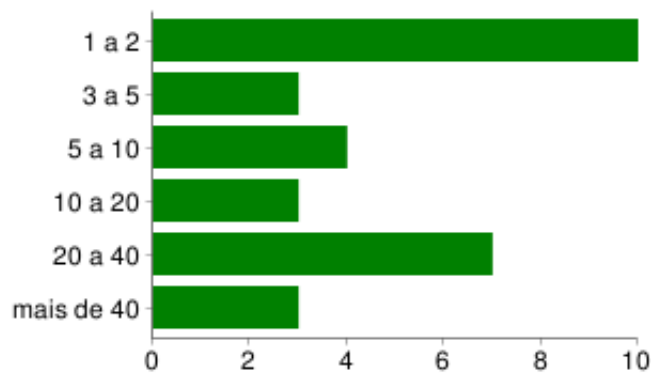


Figura 4.11: Média de horas semanais dedicadas

Foram 30 respostas para este item, sendo os seguintes valores absolutos e relativos:

- 1 a 2 horas indicada por 10 pessoas, corresponde a 33%;
- 3 a 5 horas indicada por 3 pessoas, corresponde a 10%;
- 5 a 10 horas indicada por 4 pessoas, corresponde a 13%;
- 10 a 20 horas indicada por 3 pessoas, corresponde a 10%;
- 20 a 40 horas indicada por 7 pessoas, corresponde a 23%;
- mais de 40 horas indicada por 3 pessoas, corresponde a 10%.

Sistema Operacional de Desenvolvimento

Informações sobre o ambiente de desenvolvimento também foram obtidas. A Figura 4.12 indicar o resultado sobre o Sistema Operacional do ambiente de desenvolvimento do principal software livre que os especialistas colaboram.

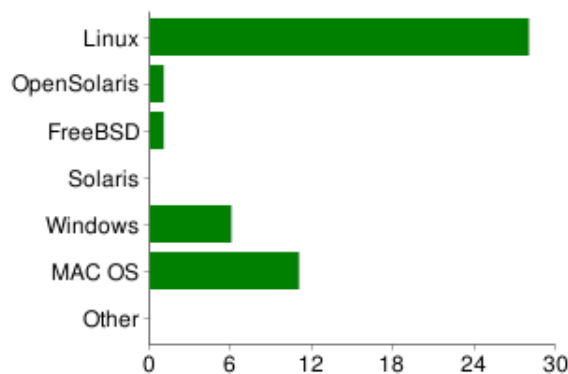


Figura 4.12: S.O. do ambiente de desenvolvimento do software

Dentre as respostas para este item, podendo ser indicada mais de um sistema operacional, os valores absolutos e relativos são:

- Linux indicado por 28 pessoas, corresponde a 97%;

- OpenSolaris indicado por 1 pessoa, corresponde a 3%;
- FreeBSD indicado por 1 pessoa, corresponde a 3%;
- Solaris não foi indicado como resposta;
- Windows indicado por 6 pessoas, corresponde a 21%;
- MAC OS indicado por 11 pessoas, corresponde a 38%;

Linguagem de Programação Usada

Informações sobre a linguagem de programação usada para o desenvolvimento do software livre também foram obtidas, conforme os resultados resumidos na Figura 4.13.

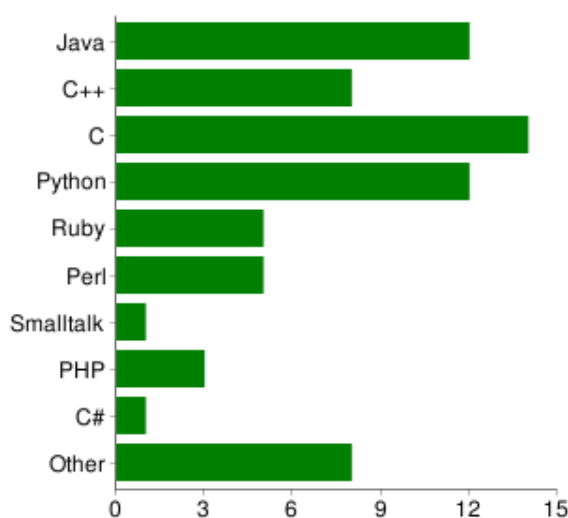


Figura 4.13: Linguagem de programação usada na implementação

Os detalhes para as respostas deste item, podendo ser assinalada mais de uma opção de linguagem de programação, está como a seguir:

- Java obteve 12 respostas, correspondendo a 40%;
- C++ obteve 8 respostas, correspondendo a 27%;
- C obteve 14 respostas, correspondendo a 47%;
- Python obteve 12 respostas, correspondendo a 40%;
- Ruby obteve 5 respostas, correspondendo a 17%;
- Perl obteve 5 respostas, correspondendo a 17%;
- Smalltalk obteve 1 resposta, correspondendo a 3%;
- PHP obteve 3 respostas, correspondendo a 10%;
- C# obteve 1 resposta, correspondendo a 3%;
- Outras opções foram indicadas por 8 pessoas, correspondendo a 27%. Entre as outras citadas estão *Shell Script* e *Lua*.

Sistema Operacional de Funcionamento

Informações sobre o ambiente de funcionamento do software também foram obtidas. A Figura 4.12 indicar o resultado sobre o Sistema Operacional do ambiente de funcionamento do principal software livre que os especialistas colaboram.

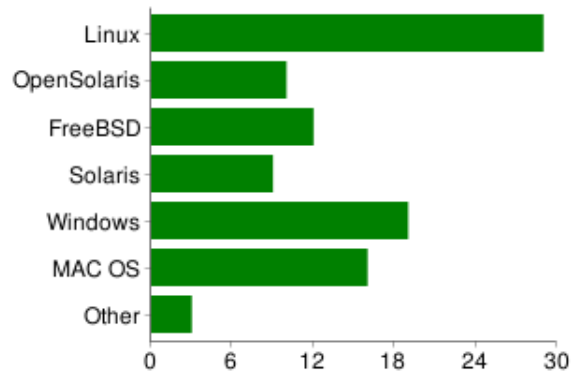


Figura 4.14: S.O. em que funciona o software

Dentre as respostas para este item, podendo ser indicada mais de um sistema operacional, os valores absolutos e relativos são:

- Linux foi indicado por 29 pessoas, correspondendo a 100%;
- OpenSolaris foi indicado por 10 pessoas, corresponde a 34%;
- FreeBSD foi indicado por 12 pessoas, corresponde a 41%;
- Solaris foi indicado por 9 pessoas, corresponde a 31%;
- Windows foi indicado por 19 pessoas, corresponde a 66%;
- MAC OS foi indicado por 16 pessoas, corresponde a 55%;
- Outras opções foram indicadas por 3 pessoas, correspondendo a 10%. Entre os outros indicados estão Palm OS, OLPC e Symbian.

Ferramentas de Desenvolvimento

Informações sobre as ferramentas usadas no desenvolvimento de software também foram obtidas, conforme os resultados resumidos na Figura 4.15. Destacando o Vi como o principal ambiente de desenvolvimento usado pelos especialistas.

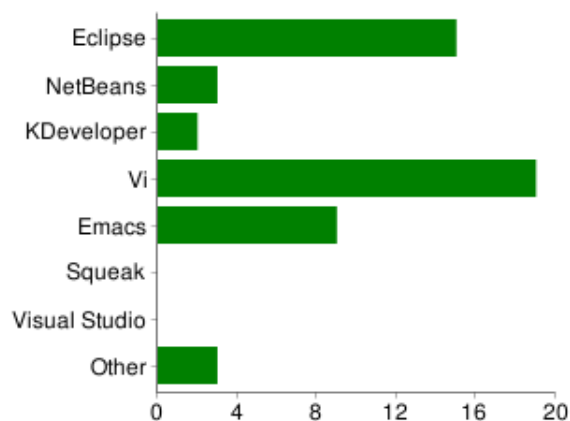


Figura 4.15: Ferramentas usada no desenvolvimento

Os detalhes para as respostas deste item, podendo ser assinalada mais de uma opção de ambiente de desenvolvimento, está como a seguir:

- Eclipse foi indicado por 15 pessoas, corresponde a 54%;
- NetBeans foi indicado por 3 pessoas, corresponde a 11%;
- KDevelope foi indicado por 2 pessoas, corresponde a 7%;
- Vi foi indicado por 19 pessoas, corresponde a 68%;
- Emacs foi indicado por 9 pessoas, corresponde a 32%;
- Squeak e Visual Studio não foi assinalado como respostas;
- Outros ambientes foram indicados por 3 pessoas, correspondendo a 11%. Entre os outros estão Dit e o TextMate.

Perfil de Organização que Trabalham

Os especialistas em desenvolvimento de software também responderam sobre o perfil da organização na qual tem vínculo profissional. O tipo de organização, número de funcionários, área de atuação e relações com a produção e uso de software livre foram levantados. A Figura 4.16 ilustra um gráfico que mostra qual o tipo de organização que a maioria dos especialistas trabalham, destacando as empresas privadas.

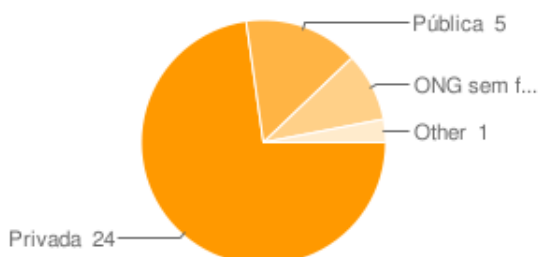


Figura 4.16: Tipo de organização que têm vínculo

Os detalhes das respostas para este item são:

- Privada foi o tipo indicado por 24 pessoas, correspondendo a 73%;
- Pública foi o tipo indicado por 5 pessoas, correspondendo a 15%;
- ONG sem fins lucrativos foi o tipo indicado por 3 pessoas, corresponde a 9%;
- Outros tipos foi indicado por 1 pessoa, correspondendo a 3%. O outro tipo indicado foi organização privada sem fins lucrativos.

Dentre as áreas de negócio das organizações estão o desenvolvimento de software, educação a Distância, gestão de projetos e consultoria, fertilizantes, CMS (*Content Management System*), Internet, serviços de TI, turismo, ensino, indústria de telecom, Ensino etc.

Em relação ao número de funcionários os valores obtidos foram:

- 1 a 5 funcionários foi indicado por 8 especialistas, correspondendo a 24%;
- 6 a 15 funcionários foi indicado por 5 especialistas, correspondendo a 15%;
- 16 a 50 funcionários foi indicado por 5 especialistas, correspondendo a 15%;
- 51 a 100 funcionários foi indicado por 2 especialistas, correspondendo a 6%;
- 101 a 500 funcionários foi indicado por 3 especialistas, correspondendo a 9%;
- mais de 500 funcionários foi indicado por 10 especialistas, correspondendo a 30%.

A indicação do tipo de relação das organizações com os sistemas e aplicações livre estão resumidas na Tabela 4.2, destacando o fato de todas as organizações usarem software livre e quando colaboram a maioria indicou a personalização como atividades.

Relação	Sim	%	Não	%
<i>Produz software livre?</i>	19	58%	14	42%
<i>Usa software livre?</i>	33	100%	0	0%
<i>Colabora com um software livre? (sugere modificações, relata problemas, etc.)</i>	25	76%	8	24%
<i>Adiciona funcionalidades em um software livre?</i>	24	73%	9	27%
<i>Personaliza/configura um software livre existente para o seu contexto?</i>	26	79%	7	21%

Tabela 4.2: Relação das organizações com projetos de software livre

Em relação ao uso de software livre pelas organizações dos especialistas em desenvolvimento, foram levantados alguns dados que estão resumidos na Tabela 4.3. Chamando a atenção o uso de software livre para auxílio no desenvolvimento de software e ao processo interno do negócio da empresa.

Relação	Sim	%	Não	%
Usa para auxílio no desenvolvimento de software?	34	100%	0	0%
Como parte de outro produto?	30	88%	4	12%
Personalização/configuração do software livre como forma de serviço?	19	58%	14	42%
Para auxílio ao processo interno do negócio da empresa (ou seu)?	32	94%	2	6%
Usado para prover serviços para clientes?	26	76%	8	24%

Tabela 4.3: Relação das organizações ao uso de software livre

4.2 Resultados das Métricas e Aspectos Pesquisados

Os especialistas em desenvolvimento de software indicaram quais aspectos e métricas levam em consideração para avaliar a qualidade de um software a fim de determinar se deve adotá-lo em sua empresa/organização. Tais aspectos e métricas foram apresentadas em sub-grupos, como definidos no capítulo anterior. Os especialistas indicaram a relevância de cada um dos 106 itens listados como métricas no questionário (Anexo A) numa escala de 1 a 10, onde 1 é considerado *irrelevante* e 10 é *relevante*. Nesta seção serão apresentados os gráficos com as médias obtidas e as tabelas comparativas com os valores das médias e desvio padrão de cada uma das métricas.

Código-Fonte

A Figura 4.17 apresenta o gráfico com as médias obtidas da relevância que os entrevistados deram aos aspectos e métricas de código-fonte. Dentre as 24 métricas as que mais se destacaram foram *módulo para configuração*, *separações em módulos* e *uso de bons nomes* como as mais significativas. Já métricas clássicas como *complexidade ciclomática* e *número de linhas de código* não foram consideradas muito significativas.

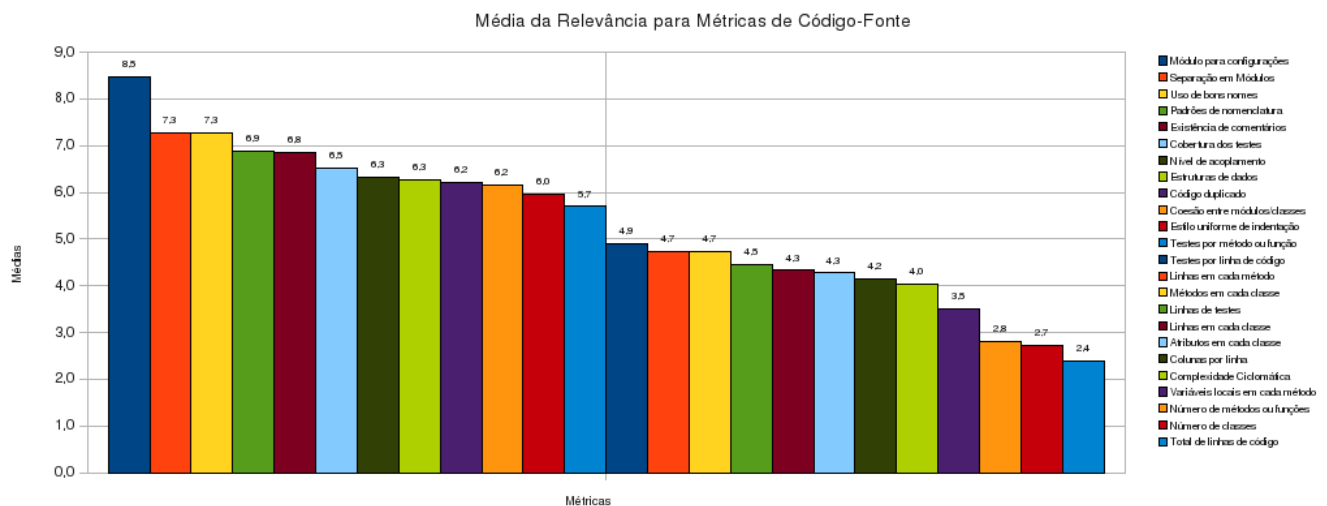


Figura 4.17: Gráfico com as médias obtidas em relação à relevância das métricas de código-fonte

A Tabela 4.4 apresenta os resultados das médias e valores de desvio padrão em relação à relevância considerada para as métricas de código-fonte. A tabela, assim como o gráfico, está ordenada em ordem decrescente, de acordo com os valores das médias obtidas. Observando os valores de desvio padrão,

atribuindo um desvio padrão igual a 3, têm-se 10 métricas apresentando desvio padrão maior que 3, assim podendo considerar que houve discordância entre as opiniões nestes casos. Porém, as 3 primeira, indicadas como mais significativas, e as 5 últimas, consideradas menos significativas, não indicam um alto grau de discordância de acordo com esse critério de desvio padrão aceitável.

Métrica	Média	Desvio Padrão
Módulo para configurações	8,5	2,1
Separação em Módulos	7,3	2,9
Uso de bons nomes	7,3	2,9
Padrões de nomenclatura	6,9	3,1
Existência de comentários	6,8	2,8
Cobertura dos testes	6,5	3,4
Nível de acoplamento	6,3	3,2
Estruturas de dados	6,3	3,2
Código duplicado	6,2	3,1
Coesão entre módulos/classes	6,2	3,0
Estilo uniforme de indentação	6,0	3,2
Testes por método ou função	5,7	3,3
Testes por linha de código	4,9	3,5
Linhas em cada método	4,7	2,9
Métodos em cada classe	4,7	2,9
Linhas de testes	4,5	3,2
Linhas em cada classe	4,3	2,8
Atributos em cada classe	4,3	2,8
Colunas por linha	4,2	3,0
Complexidade Ciclomática	4,0	2,5
Variáveis locais em cada método	3,5	2,5
Número de métodos ou funções	2,8	2,1
Número de classes	2,7	2,2
Total de linhas de código	2,4	1,9

Tabela 4.4: Médias e desvio padrão das métricas de código-fonte

Testes

A Figura 4.18 apresenta o gráfico com as médias obtidas da relevância que os entrevistados deram aos aspectos e métricas de testes. Dentre as 11 métricas as que mais se destacaram foram *testes automatizados*, *teste de desempenho* e *testabilidade do código* como as mais significativas. Já teste de mutação não foi considerado como significativo, conforme se observa no gráfico.

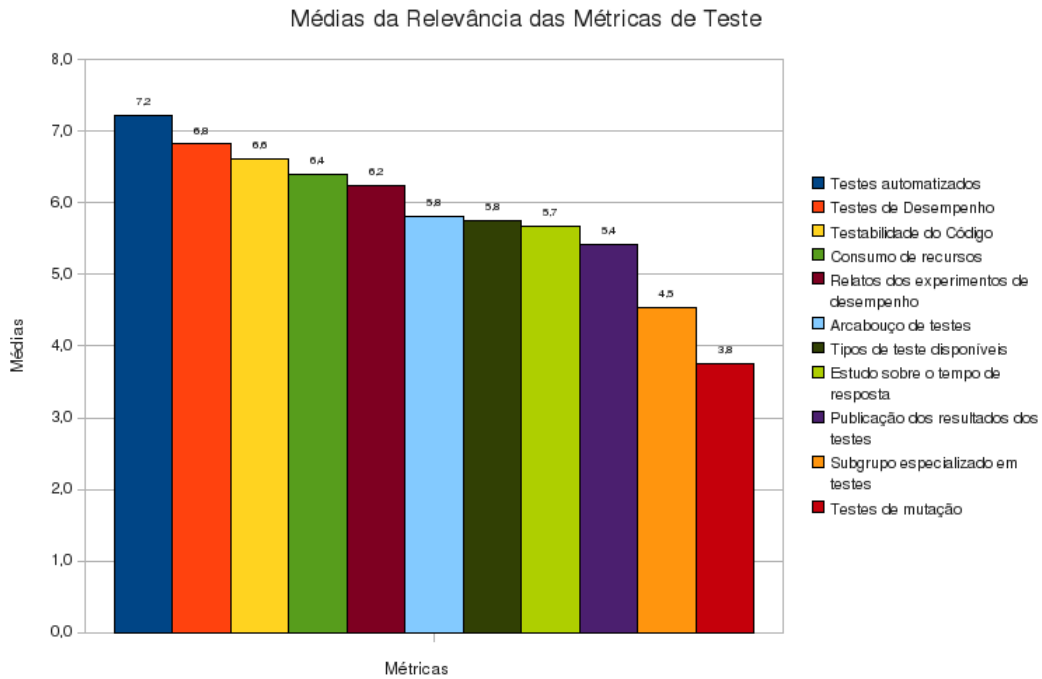


Figura 4.18: Gráfico com as médias obtidas em relação à relevância das métricas de testes

A Tabela 4.5 apresenta os resultados das médias e valores de desvio padrão em relação à relevância considerada para as métricas de testes. A tabela, assim como o gráfico, está ordenada em ordem decrescente, de acordo com os valores das médias obtidas. Observando os valores de desvio padrão, atribuindo um desvio padrão igual a 3, têm-se 5 métricas apresentando desvio padrão maior que 3, assim podendo considerar que houve discordância entre as opiniões nestes casos. Dessas, com um desvio padrão alto, está *testes automatizados*, que tem maior média de relevância. Outras métricas consideradas não muito significativas, como *publicação dos resultados dos testes* e *sub-grupo especializado em testes*, também têm um desvio padrão maior que 3.

Métrica	Média	Desvio Padrão
Testes automatizados	7,2	3,3
Testes de Desempenho	6,8	2,3
Testabilidade do Código	6,6	2,8
Consumo de recursos	6,4	2,6
Relatos dos experimentos de desempenho	6,2	2,6
Arcabouço de testes	5,8	3,7
Tipos de teste disponíveis	5,8	3,4
Estudo sobre o tempo de resposta	5,7	2,9
Publicação dos resultados dos testes	5,4	3,3
Subgrupo especializado em testes	4,5	3,2
Testes de mutação	3,8	2,6

Tabela 4.5: Médias e desvio padrão das métricas de testes

Manutenibilidade

A Figura 4.19 apresenta o gráfico com as médias obtidas da relevância que os entrevistados deram aos aspectos e métricas de manutenibilidade. Dentre as 11 métricas as que mais se destacaram foram

versões/atualização de manutenção, documentação/descrição da arquitetura, uso de boas práticas de desenvolvimento e linguagem de programação como as mais significativas. O aspecto de ter uma *organização/empresa fornecedora de suporte* foi considerada a menos significativa, mas com uma média de relevância quase 5.

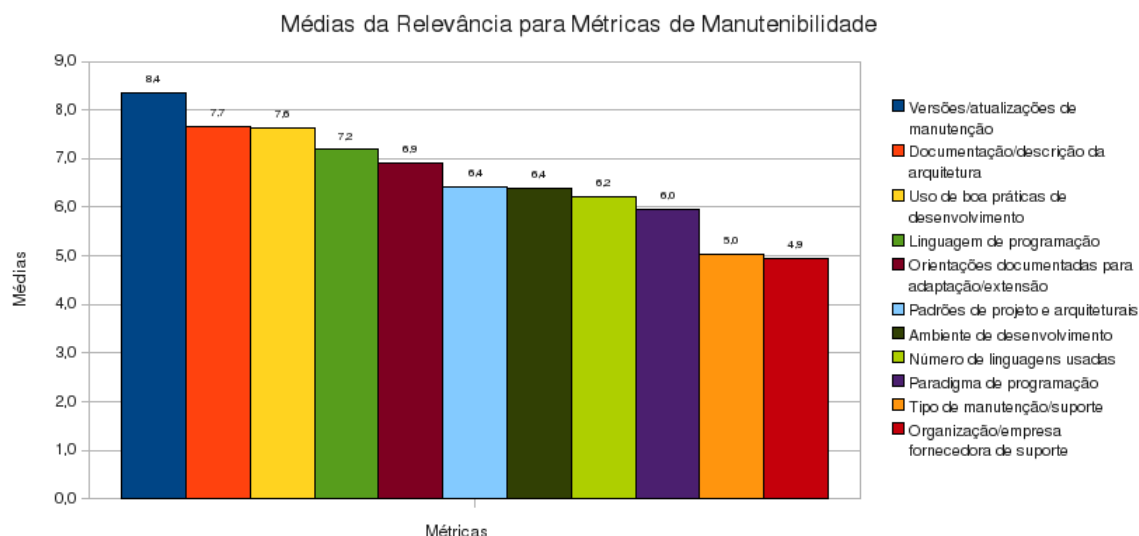


Figura 4.19: Gráfico com as médias obtidas em relação à relevância das métricas de manutenibilidade

A Tabela 4.6 apresenta os resultados das médias e valores de desvio padrão em relação à relevância considerada para os aspectos e métricas de manutenibilidade. A tabela, assim como o gráfico, está ordenada em ordem decrescente, de acordo com os valores das médias obtidas. Observando os valores de desvio padrão, atribuindo um desvio padrão igual a 3 como o limite de discordância, têm-se apenas 3 métricas apresentando desvio padrão maior que 3, assim podendo considerar que houve discordância entre as opiniões nestes casos. De qualquer forma, os primeiros aspectos e métricas apresentam um desvio padrão menor que 3, assim como *organização/empresa fornecedora de suporte*, considerada a menos significativa.

Métrica	Média	Desvio Padrão
Versões/atualizações de manutenção	8,4	1,9
Documentação/descrição da arquitetura	7,7	2,3
Uso de boas práticas de desenvolvimento	7,6	2,2
Linguagem de programação	7,2	2,8
Orientações documentadas para adaptação/extensão	6,9	2,4
Padrões de projeto e arquiteturas	6,4	2,7
Ambiente de desenvolvimento	6,4	3,4
Número de linguagens usadas	6,2	3,2
Paradigma de programação	6,0	3,1
Tipo de manutenção/suporte	5,0	3,0
Organização/empresa fornecedora de suporte	4,9	2,7

Tabela 4.6: Médias e desvio padrão das métricas de manutenibilidade

Interoperabilidade

A Figura 4.20 apresenta o gráfico com as médias obtidas da relevância que os entrevistados deram aos aspectos e métricas de interoperabilidade. Todas as 3 métricas tiveram médias altas de relevância, destacando os *padrões da indústria e protocolos bem disseminados* como o mais significativo.

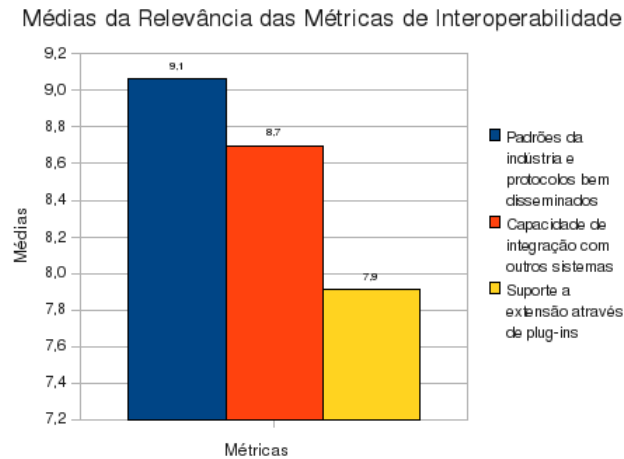


Figura 4.20: Gráfico com as médias obtidas em relação à relevância das métricas de Interoperabilidade

A Tabela 4.7 apresenta os resultados das médias e valores de desvio padrão em relação à relevância considerada para os aspectos e métricas de interoperabilidade. A tabela, assim como o gráfico, está ordenada em ordem decrescente, de acordo com os valores das médias obtidas. Os valores de desvio padrão, de acordo com a convenção usada neste trabalho de desvio padrão até 3, podem ser considerados baixo, principalmente os dois primeiros aspectos apresentados na tabela - *padrões da indústria e protocolos bem disseminados* e *capacidade de integração com outros sistemas* - com desvio padrão de até 1,5.

Métrica	Média	Desvio Padrão
Padrões da indústria e protocolos bem disseminados	9,1	1,3
Capacidade de integração com outros sistemas	8,7	1,5
Suporte a extensão através de plug-ins	7,9	2,3

Tabela 4.7: Médias e desvio padrão das métricas de interoperabilidade

Portabilidade

A Figura 4.21 apresenta o gráfico com as médias obtidas da relevância que os entrevistados deram aos aspectos e métricas de portabilidade. As 2 métricas tiveram médias boas de relevância, destacando *lista de ambientes suportados* como a mais significativa.

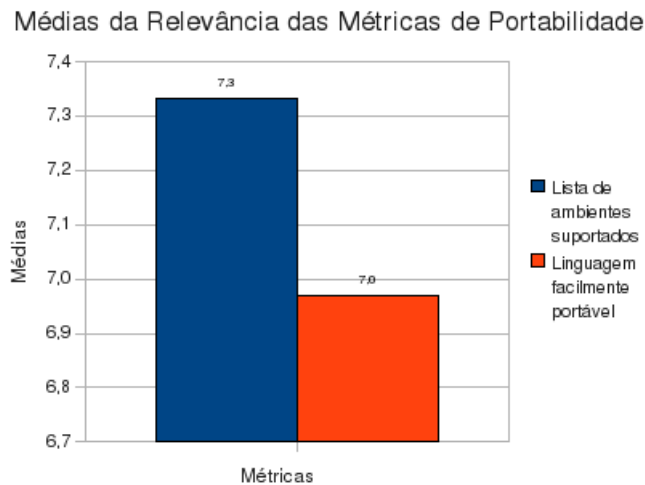


Figura 4.21: Gráfico com as médias obtidas em relação à relevância das métricas de Portabilidade

A Tabela 4.8 apresenta os resultados das médias e valores de desvio padrão em relação à relevância considerada para os aspectos e métricas de portabilidade. A tabela, assim como o gráfico, está ordenada em ordem decrescente, de acordo com os valores das médias obtidas. Os valores de desvio padrão, de acordo com a convenção usada neste trabalho de desvio padrão até 3, podem ser considerados normais por estarem um pouco abaixo de 3.

Métrica	Média	Desvio Padrão
Lista de ambientes suportados	7,3	2,8
Linguagem facilmente portátil	7,0	2,9

Tabela 4.8: Médias e desvio padrão das métricas de portabilidade

Usabilidade

A Figura 4.22 apresenta o gráfico com as médias obtidas da relevância que os entrevistados deram aos aspectos e métricas de usabilidade. Todas as 5 métricas tiveram boas médias de relevância, destacando *clareza da interface do usuário e facilidade de instalação e configuração* como as mais significativas, com médias acima de 8.

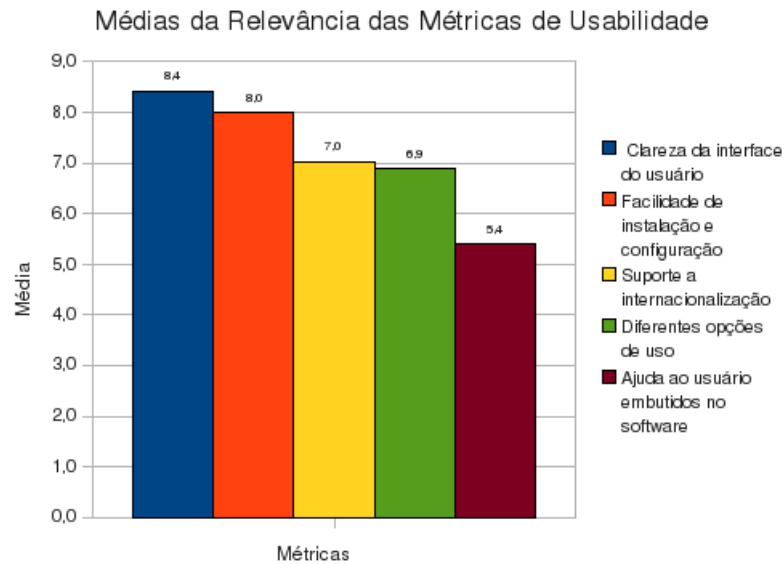


Figura 4.22: Gráfico com as médias obtidas em relação à relevância das métricas de usabilidade

A Tabela 4.9 apresenta os resultados das médias e valores de desvio padrão em relação à relevância considerada para os aspectos e métricas de usabilidade. A tabela, assim como o gráfico, está ordenada em ordem decrescente, de acordo com os valores das médias obtidas. Os valores de desvio padrão, de acordo com a convenção usada neste trabalho de desvio padrão até 3, podem ser considerados normais por variarem entre 2,2 a 2,6.

Métrica	Média	Desvio Padrão
Clareza da interface do usuário	8,4	2,2
Facilidade de instalação e configuração	8,0	2,4
Suporte a internacionalização	7,0	2,6
Diferentes opções de uso	6,9	2,3
Ajuda ao usuário embutidos no software	5,4	2,5

Tabela 4.9: Médias e desvio padrão das métricas de usabilidade

Ferramentas

A Figura 4.23 apresenta o gráfico com as médias obtidas da relevância que os entrevistados deram aos aspectos e métricas de ferramentas. Os 4 aspectos tiveram boas médias de relevância, com médias maior que 5, destacando *scripts para configuração do processo de build e controle de versões de commits* como os mais significativos.

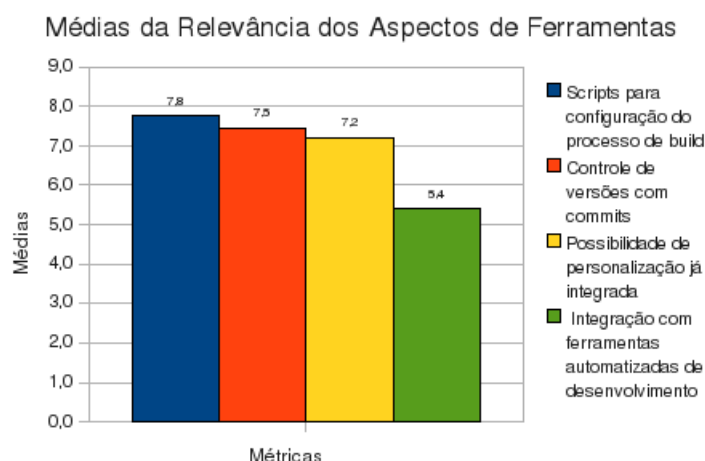


Figura 4.23: Gráfico com as médias obtidas em relação à relevância das métricas de ferramentas

A Tabela 4.10 apresenta os resultados das médias e valores de desvio padrão em relação à relevância considerada para os aspectos e métricas de ferramentas. A tabela, assim como o gráfico, está ordenada em ordem decrescente, de acordo com os valores das médias obtidas. Os valores de desvio padrão, de acordo com a convenção usada neste trabalho de desvio padrão até 3, podem ser considerados normais por variarem entre 2,3 a 2,9.

Métrica	Média	Desvio Padrão
Scripts para configuração do processo de build	7,8	2,7
Controle de versões com commits	7,5	2,7
Possibilidade de personalização já integrada	7,2	2,3
Integração com ferramentas automatizadas de desenvolvimento	5,4	2,9

Tabela 4.10: Médias e desvio padrão das métricas de ferramentas

Independência

A Figura 4.24 apresenta o gráfico com as médias obtidas da relevância que os entrevistados deram aos aspectos e métricas de independência. Os 2 aspectos tiveram boas médias de relevância com médias maior que 5, destacando *dependências bem documentadas* como os mais significativos, com média acima de 8.

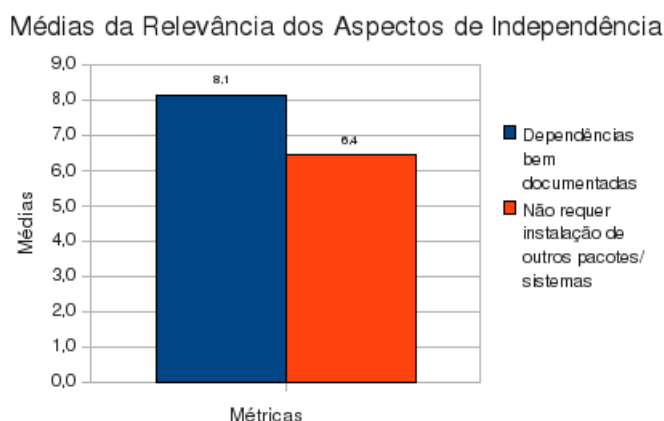


Figura 4.24: Gráfico com as médias obtidas em relação à relevância das métricas de independência

A Tabela 4.11 apresenta os resultados das médias e valores de desvio padrão em relação à relevância considerada para os aspectos e métricas de independência. A tabela, assim como o gráfico, está ordenada em ordem decrescente, de acordo com os valores das médias obtidas. Observando os valores de desvio padrão, atribuindo um desvio padrão igual a 3 como o limite de discordância, tem-se apenas 1 aspecto - *não requer instalação de outros pacotes/sistemas* - apresentando desvio padrão um pouco maior que 3, assim podendo considerar que houve discordância entre as opiniões neste caso.

Métrica	Média	Desvio Padrão
Dependências bem documentadas	8,1	2,5
Não requer instalação de outros pacotes/sistemas	6,4	3,1

Tabela 4.11: Médias e desvio padrão das métricas de independência

Comunidade

A Figura 4.25 apresenta o gráfico com as médias obtidas da relevância que os entrevistados deram aos aspectos e métricas de comunidade. Dentre as 12 métricas as que mais se destacaram foram *lista de e-mail ou fórum de usuários*, *lista de e-mail ou fórum dos desenvolvedores* e *atividades recentes na lista de e-mail* como as mais significativas. De qualquer forma, todos os aspectos e métricas foram considerados significativos, apresentando médias maiores que 5.

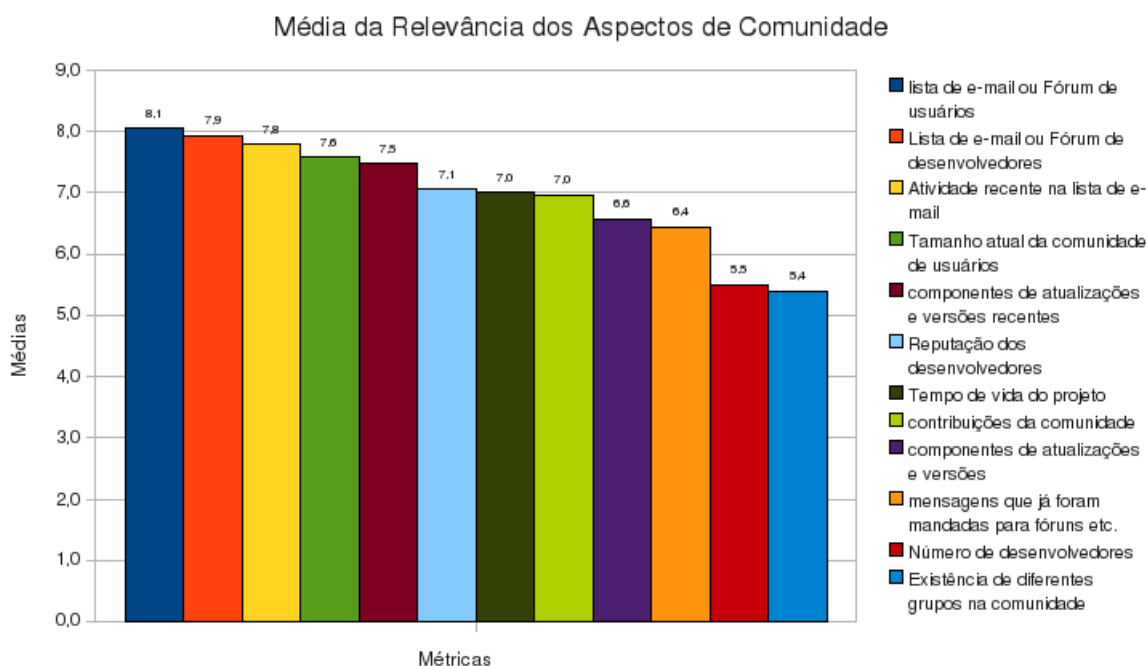


Figura 4.25: Gráfico com as médias obtidas em relação à relevância das métricas de comunidade

A Tabela 4.26 apresenta os resultados das médias e valores de desvio padrão em relação à relevância considerada para os aspectos e métricas de comunidade. A tabela, assim como o gráfico, está ordenada em ordem decrescente, de acordo com os valores das médias obtidas. Observando os valores de desvio padrão, atribuindo um desvio padrão igual a 3 como o limite de discordância, têm-se apenas 2 aspecto, os dois indicados com menos significativos, apresentam desvio padrão um pouco maior que 3, assim podendo considerar que houve discordância entre as opiniões nestes casos.

Métrica	Média	Desvio Padrão
lista de e-mail ou Fórum de usuários	8,1	2,4
Lista de e-mail ou Fórum de desenvolvedores	7,9	2,5
Atividade recente na lista de e-mail	7,8	2,8
Tamanho atual da comunidade de usuários	7,6	2,5
componentes de atualizações e versões recentes	7,5	2,7
Reputação dos desenvolvedores	7,1	2,6
Tempo de vida do projeto	7,0	2,7
Contribuições da comunidade	7,0	2,8
Componentes de atualizações e versões	6,6	3,0
Mensagens que já foram mandadas para fóruns etc	6,4	3,0
Número de desenvolvedores	5,5	3,1
Existência de diferentes grupos na comunidade	5,4	3,1

Figura 4.26: Médias e desvio padrão das métricas de comunidade

Aspectos de Confiança

A Figura 4.27 apresenta o gráfico com as médias obtidas da relevância que os entrevistados deram aos aspectos e métricas de confiança. Dentre as 6 métricas, as que mais se destacaram foram *tempo de solução de erros críticos*, *reputação do software* e *disponibilização de atualizações críticas*, como as mais significativas, com médias maiores que 8. De qualquer forma, todos os aspectos e métricas foram considerados significativos, apresentando médias maiores que 5.

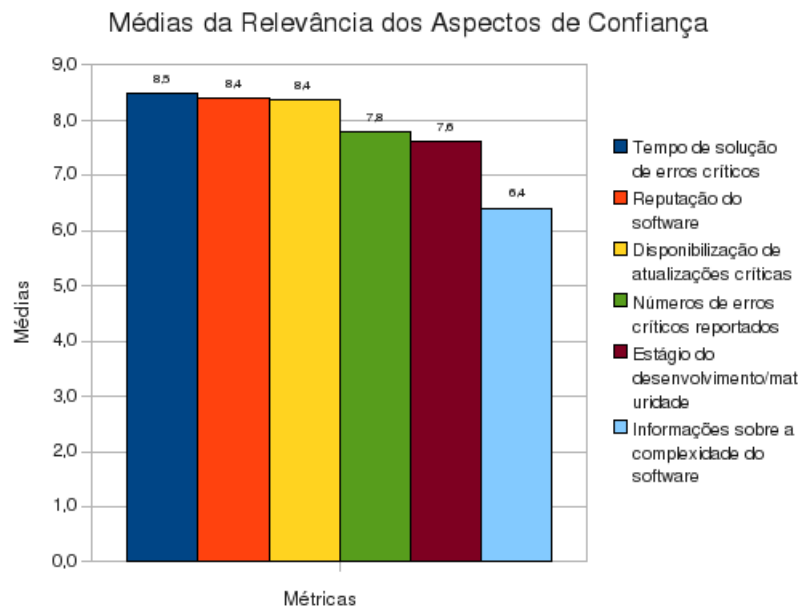


Figura 4.27: Gráfico com as médias obtidas em relação à relevância dos aspectos de confiança

A Tabela 4.12 apresenta os resultados das médias e valores de desvio padrão em relação à relevância considerada para os aspectos e métricas de confiança. A tabela, assim como o gráfico, está ordenada em ordem decrescente, de acordo com os valores das médias obtidas. Os valores de desvio padrão, de acordo com a convenção usada neste trabalho de desvio padrão até 3, podem ser considerados normais por variarem entre 1,5 a 2,4.

Métrica	Média	Desvio Padrão
Tempo de solução de erros críticos	8,5	1,8
Reputação do software	8,4	1,5
Disponibilização de atualizações críticas	8,4	2,1
Números de erros críticos reportados	7,8	2,2
Estágio do desenvolvimento/maturidade	7,6	2,3
Informações sobre a complexidade do software	6,4	2,4

Tabela 4.12: Médias e desvio padrão das métricas de aspectos de confiança

Funcionalidades

A Figura 4.28 apresenta o gráfico com as médias obtidas da relevância que os entrevistados deram aos aspectos e métricas de funcionalidade. Dentre as 5 métricas as que mais se destacaram foram *disponibilidade de uma lista bem definida de funcionalidade e protótipo do software* como as mais significativas. O aspecto considerado menos significativo foi *existência de produtos derivados*, com média de relevância menor que 5.

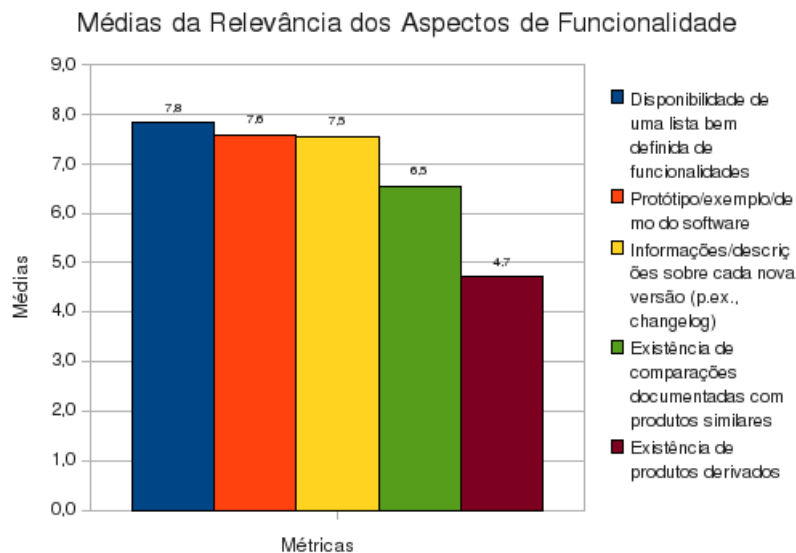


Figura 4.28: Gráfico com as médias obtidas em relação à relevância dos aspectos de funcionalidade

A Tabela 4.13 apresenta os resultados das médias e valores de desvio padrão em relação à relevância considerada para os aspectos e métricas de funcionalidade. A tabela, assim como o gráfico, está ordenada em ordem decrescente, de acordo com os valores das médias obtidas. Os valores de desvio padrão, de acordo com a convenção usada neste trabalho de desvio padrão até 3, podem ser considerados normais por variarem entre 2,5 a 2,7.

Métrica	Média	Desvio Padrão
Disponibilidade de uma lista bem definida de funcionalidades	7,8	2,6
Protótipo/exemplo/demo do software	7,6	2,7
Informações/descrições sobre cada nova versão (p.ex., changelog)	7,5	2,6
Existência de comparações documentadas com produtos similares	6,5	2,5
Existência de produtos derivados	4,7	2,7

Tabela 4.13: Médias e desvio padrão das métricas de funcionalidade

Satisfação do Usuário/Cliente

A Figura 4.29 apresenta o gráfico com as médias obtidas da relevância que os entrevistados deram aos aspectos e métricas de satisfação do usuário. As 4 métricas obtiveram boas médias de relevância, mas nenhuma chegou a relevância 7.

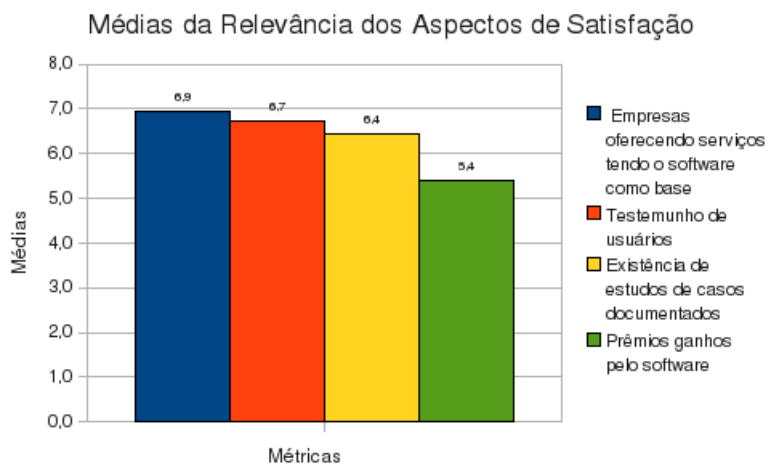


Figura 4.29: Gráfico com as médias obtidas em relação à relevância dos aspectos de satisfação do usuário

A Tabela 4.14 apresenta os resultados das médias e valores de desvio padrão em relação à relevância considerada para os aspectos e métricas de satisfação do usuário. A tabela, assim como o gráfico, está ordenada em ordem decrescente de acordo com os valores das médias obtidas. Os valores de desvio padrão, de acordo com a convenção usada neste trabalho de desvio padrão até 3, podem ser considerados normais por variarem entre 2,3 a 2,8.

Métrica	Média	Desvio Padrão
Empresas oferecendo serviços tendo o software como base	6,9	2,7
Testemunho de usuários	6,7	2,3
Existência de estudos de casos documentados	6,4	2,8
Prêmios ganhos pelo software	5,4	2,8

Tabela 4.14: Médias e desvio padrão das métricas de satisfação do usuário

Documentação

A Figura 4.30 apresenta o gráfico com as médias obtidas da relevância que os entrevistados deram aos aspectos e métricas de documentação. Todas as 6 métricas obtiveram boas médias de relevância, ficando com médias maiores que 7, destacando o *FAQ Técnico* como a mais relevante com média maior que 8.

Métrica	Média	Desvio Padrão
FAQ técnico	8,1	2,5
FAQ para usuários	7,9	2,4
Guia de instalação	7,9	2,6
Manual técnico	7,8	2,4
Manual do usuário	7,6	2,5
Documentação técnica	7,3	2,9

Tabela 4.15: Médias e desvio padrão das métricas de documentação

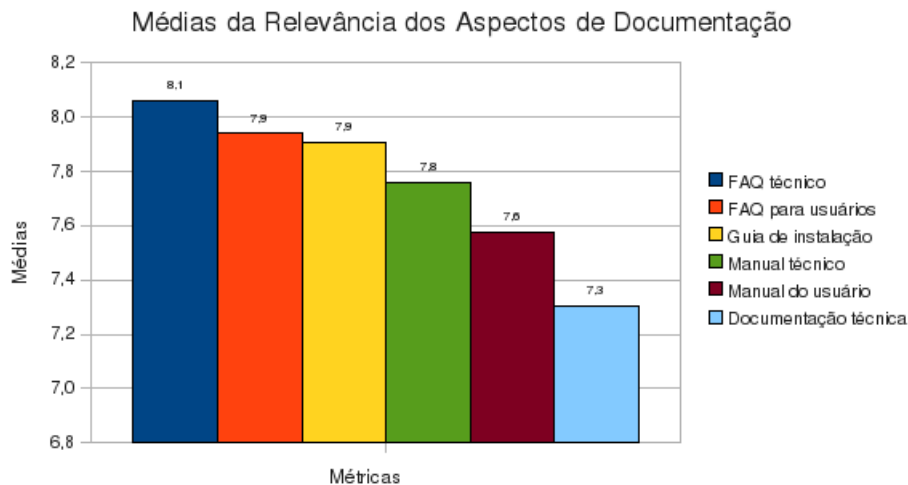


Figura 4.30: Gráfico com as médias obtidas em relação à relevância dos aspectos de documentação

A Tabela 4.15 apresenta os resultados das médias e valores de desvio padrão em relação à relevância considerada para os aspectos e métricas de documentação. A tabela, assim como o gráfico, está ordenada em ordem decrescente, de acordo com os valores das médias obtidas. Os valores de desvio padrão, conforme a convenção usada neste trabalho de desvio padrão até 3, podem ser considerados normais por variarem entre 2,4 a 2,9.

Suporte

A Figura 4.31 apresenta o gráfico com as médias obtidas da relevância que os entrevistados deram aos aspectos e métricas de suporte. As 4 métricas obtiveram boas médias de relevância, destacando o *tempo de resposta quando detectado um erro pelo usuário* e o *acompanhamento de erros*, como as mais significativas, com médias de relevância maiores que 8.

Métrica	Média	Desvio Padrão
Tempo de resposta quando detectado um erro pelo usuário	8,2	2,0
Acompanhamento de erros	8,1	2,2
Correções de erros disponibilizadas	7,1	2,6
Processo definido para correção de erros	6,5	3,2

Tabela 4.16: Médias e desvio padrão das métricas de suporte

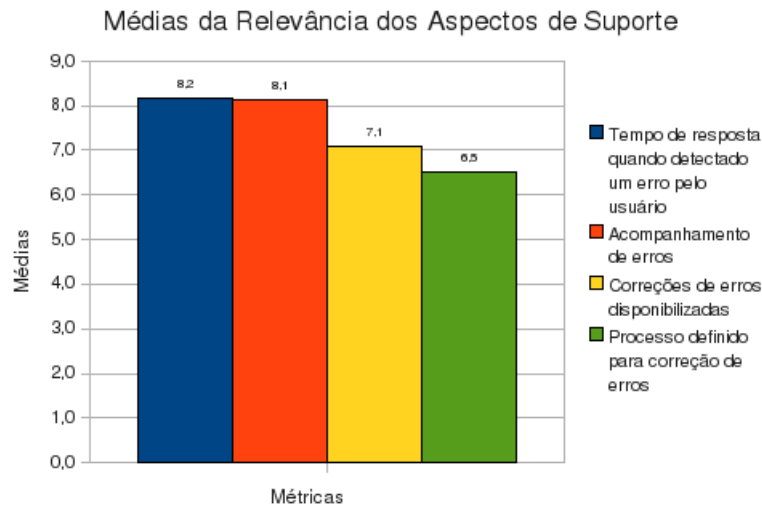


Figura 4.31: Gráfico com as médias obtidas em relação à relevância dos aspectos de suporte ao software

A Tabela 4.16 apresenta os resultados das médias e valores de desvio padrão em relação à relevância considerada para os aspectos e métricas de suporte. A tabela, assim como o gráfico, está ordenada em ordem decrescente de acordo com os valores das médias obtidas. O aspecto menos significativo (*processo definido para correção de erros*), porém com uma boa média de relevância, apresenta discordância das opiniões do entrevistados, de acordo com a convenção usada neste trabalho de desvio padrão até 3.

Treinamento

A Figura 4.32 apresenta o gráfico com as médias obtidas da relevância que os entrevistados deram aos aspectos e métricas de treinamento. Dos 2 aspectos analisados, um (*disponibilidade de treinamentos, tutoriais e outros materiais de aprendizagem*) apresenta uma boa média de relevância; e outro (*curso oficial de treinamento*) é apontado como pouco significativo, por ter média de relevância próxima a 4.

Médias da Relevância dos Aspectos de Treinamento

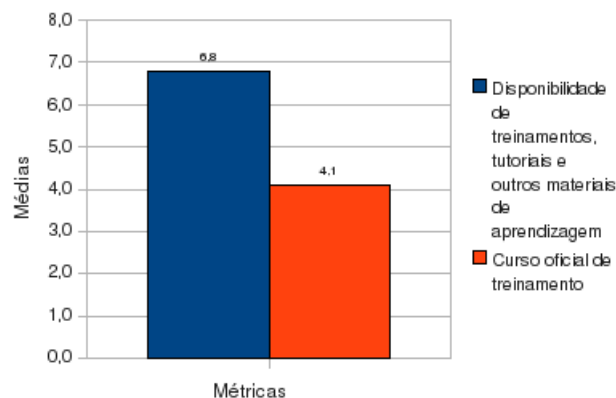


Figura 4.32: Gráfico com as médias obtidas em relação à relevância dos aspectos de treinamento

A Tabela 4.17 apresenta os resultados das médias e valores de desvio padrão em relação à relevância considerada para os aspectos e métricas de documentação. A tabela, assim como o gráfico, está ordenada em ordem decrescente, de acordo com os valores das médias obtidas. Os valores de desvio padrão, de acordo com a convenção usada neste trabalho de desvio padrão até 3, podem ser considerados normais por variarem entre 2,6 a 2,9.

Métrica	Média	Desvio Padrão
Disponibilidade de treinamentos, tutoriais e outros materiais de aprendizagem	6,8	2,9
Curso oficial de treinamento	4,1	2,6

Tabela 4.17: Médias e desvio padrão das métricas de treinamento

Canais de Distribuição e Licenças

A Figura 4.33 apresenta o gráfico com as médias obtidas da relevância que os entrevistados deram aos aspectos e métricas de canais de distribuição e licenças. Dentre as 5 métricas, a que mais se destaca é *download do código-fonte e binários*, como a mais significativa, com média de relevância maior que 9. Já o aspecto sobre a *disponibilização em formato físicos* não foi considerado muito significativo.

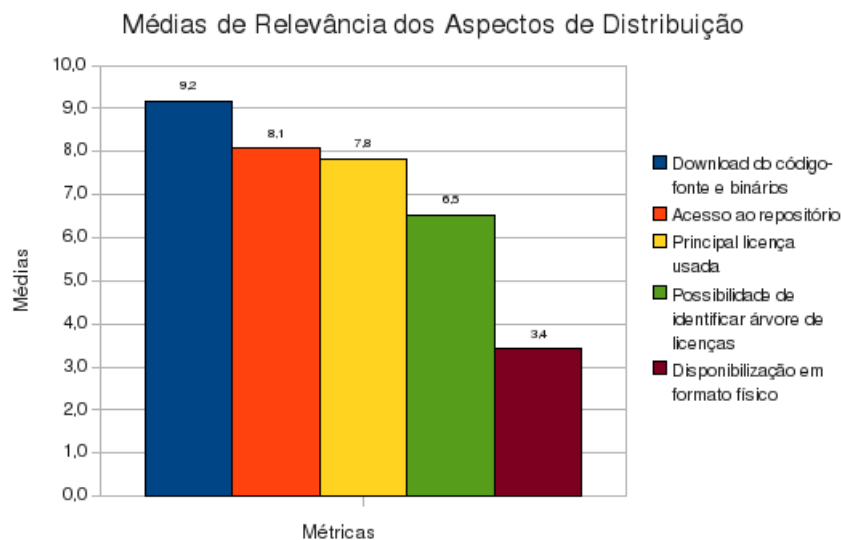


Figura 4.33: Gráfico com as médias obtidas em relação à relevância dos Aspectos de Canais de Distribuição e Licenças

A Tabela 4.18 apresenta os resultados das médias e valores de desvio padrão em relação à relevância considerada para os aspectos e métricas de canais de distribuição e licenças. A tabela, assim como o gráfico, está ordenada em ordem decrescente de acordo com os valores das médias obtidas. Um dos aspectos (*possibilidade de identificar árvore de licenças*), mesmo com uma boa média de relevância, apresenta discordância das opiniões dos entrevistados, de acordo com a convenção usada neste trabalho de desvio padrão até 3.

Métrica	Média	Desvio Padrão
Download do código-fonte e binários	9,2	1,8
Acesso ao repositório	8,1	2,7
Principal licença usada	7,8	2,7
Possibilidade de identificar árvore de licenças	6,5	3,3
Disponibilização em formato físico	3,4	2,9

Tabela 4.18: Médias e desvio padrão das métricas de canais de distribuição e licenças

4.2.1 Aspectos e Métricas mais Relevantes

A Tabela 4.19 apresenta os resultados das médias e valores de desvio padrão considerados para os aspectos e métricas mais relevantes entre todos os pesquisados. A tabela está ordenada em ordem decrescente, de acordo com os valores das médias obtidas. Os valores de desvio padrão, de acordo com a convenção usada neste trabalho de desvio padrão até 3, podem ser considerados normais por variarem entre 1,3 a 2,8, apresentando para todas as 26 métricas listadas pequenos índices de discordância. O objetivo era elencar os 20 aspectos e métricas com melhores médias de relevância, porém do 20º ao 26º na classificação dos mesmos possuíam os mesmo valor de média.

Métrica	Média	Desvio Padrão
Download do código-fonte e binários	9,2	1,8
Uso de padrões da indústria e protocolos bem disseminados	9,1	1,3
Capacidade de comunicação/integração com outros sistemas	8,7	1,5
Tempo de solução de erros críticos	8,5	1,8
Módulo para configurações	8,5	2,1
Clareza da interface do usuário	8,4	2,2
Reputação do software	8,4	1,5
Existência de versões/atualizações de manutenção	8,4	1,9
Disponibilização de atualizações críticas	8,4	2,1
Tempo de resposta quando detectado um erro pelo usuário	8,2	2,0
Dependências bem documentadas	8,1	2,5
Acompanhamento de erros	8,1	2,2
Acesso ao repositório	8,1	2,7
FAQ técnico	8,1	2,5
Existência de lista de e-mail ou Fórum de usuários	8,1	2,4
Facilidade de instalação e configuração	8,0	2,4
FAQ para usuários	7,9	2,4
Lista de e-mail ou Fórum de desenvolvedores	7,9	2,5
Guia de instalação	7,9	2,6
Suporte a extensão através de plug-ins	7,9	2,3
Disponibilidade de uma lista bem definida de funcionalidades	7,8	2,6
Principal licença usada	7,8	2,7
Números de erros críticos reportados	7,8	2,2
Atividade recente na lista de e-mail ou Fórum da comunidade	7,8	2,8
Manual técnico	7,8	2,4
Scripts para configuração do processo de build	7,8	2,7

Tabela 4.19: Médias e valores de desvio padrão das métricas mais relevantes

A Figura 4.34 apresenta o gráfico com as médias obtidas da relevância que os entrevistados deram aos mais significativos aspectos e métricas, segundo eles mesmo. Dentre as 26 métricas apresentadas, as que mais se destacaram foram *download do código-fonte* e *uso de padrões da indústria e protocolos de bem disseminados*, com as maiores diferenças do valor da média em relação às demais.

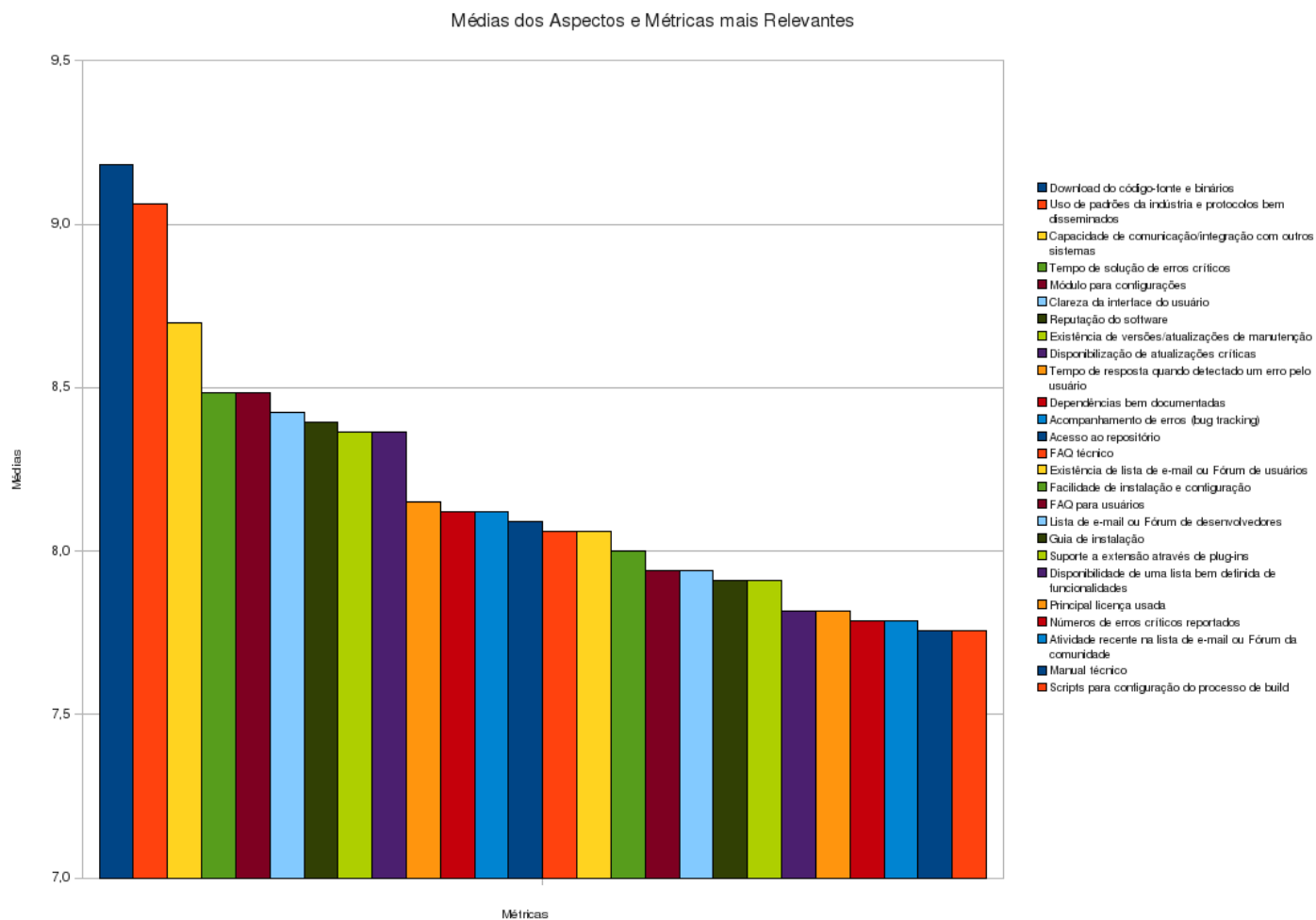


Figura 4.34: Gráfico das média das métricas mais relevantes

Capítulo 5

Considerações Finais

Este trabalho apresentou um amplo estudo sobre o estado da arte em métricas de software que vai guiar as pesquisas de doutorado, em ciência da computação, em avaliação automática da qualidade de código-fonte.

Outro objetivo e etapa do trabalho foi elencar um conjunto de aspectos e métricas com o intuito de realizar uma pesquisa com especialistas em desenvolvimento de software, ajudando também nas escolhas de quais métricas e aspectos serão focados na tentativa de automatização da avaliação de qualidade de um software livre.

A referida pesquisa foi realizada com 34 especialistas em desenvolvimento de software até o momento. Os resultados foram apresentados neste trabalho, mas não analisados com o rigor estatístico necessário, sendo precipitado constar uma conclusão no atual andamento das pesquisas. Uma mestre em estatística já começou a trabalhar na análise dos dados apresentados. Assim, pretende-se em um curto prazo apresentar conclusões efetivas e que possam ajudar na evolução do tema de doutorado proposto.

Esta monografia se concretiza como de extrema importância para os demais passos da mencionada pesquisa de doutorado, ajudando para que em um médio prazo se tenha resultados que possam ser satisfatórios do ponto de vista prático.

Apêndice A

Questionário de Levantamento de Aspectos Relevantes e Métricas de Qualidade para Adoção de um Software Livre

Paulo Meirelles (doutorando)

Prof. Fabio Kon (orientador)

Departamento de Ciência da Computação

IME / USP

Caro especialista em desenvolvimento de software,

Gostaríamos de solicitar cerca de 20 minutos do seu tempo para o preenchimento do questionário a seguir que faz parte de uma pesquisa de doutoramento no Departamento de Ciência da Computação do IME/USP. Assim que a análise dos dados estiver completa, você terá acesso em primeira mão aos resultados obtidos.

Observação: os dados pessoais coletados são para uso exclusivo do pesquisador e seu orientador e serão mantidos em total sigilo. Apenas informações totalizadas e anonimizadas serão divulgadas como resultado desta pesquisa.

1. Informações Pessoais

a)Nome:

- ...

b)E-mail:

- ...

c)Profissão/Ocupação:

- ...

d) Como você caracterizaria sua atuação profissional em relação à software:

- ☐ usuário, ☐ mantenedor, ☐ desenvolvedor, ☐ analista de negócios, ☐ consultor, ☐ vendedor, ☐ líder de equipe de desenvolvimento, ☐ gerente, ☐ diretor/presidente, ☐ outros (especificar):

e) Já participou em projeto(s) de software livre como:

- ☐ usuário, ☐ desenvolvedor, ☐ tradutor, ☐ mantenedor, ☐ disseminador, ☐ outros (especificar):

f) Formação:

- ☐ Fundamental ☐ 2o Grau ☐ Superior incompleto ☐ Superior Completo ☐ Mestrado Completo ☐ Doutorado Completo

g) Anos de experiência profissional com software:

- ☐ 0 ☐ 1 a 2 ☐ 3 a 5 ☐ 5 a 10 ☐ 10 a 20 ☐ mais de 20.

h) De quantos projetos de software livre você participa ou participou?

- ☐ 0 ☐ 1 a 2 ☐ 3 a 5 ☐ 5 a 10 ☐ 10 a 20 ☐ mais de 20.

i) Caso pertinente: tempo, em anos, em que atua em projeto(s) de software livre:

- ☐ 0 ☐ 1 a 2 ☐ 3 a 5 ☐ 5 a 10 ☐ 10 a 20 ☐ mais de 20.

2. Informações sobre o principal projeto de Software Livre de que participa:

a) Nome do software/projeto:

- ...

b) Anos de existência do projeto:

- ☐ 0 ☐ 1 a 2 ☐ 3 a 5 ☐ 5 a 10 ☐ 10 a 20 ☐ mais de 20.

c) Número de participantes:

- ☐ 1-2 ☐ 3-10 ☐ 11 a 50 ☐ 51-100 ☐ 101-500 ☐ mais de 500.

d) Número de participantes que interagem com você:

- ☐ 1-2 ☐ 3-10 ☐ 11 a 50 ☐ 51-100 ☐ 101-500 ☐ mais de 500.

e) Número de usuários:

- ☐ 1-10 ☐ dezenas ☐ centenas ☐ milhares ☐ centenas de milhares

f)Número de Downloads:

- ☐ 1-10 ☐ dezenas ☐ centenas ☐ milhares ☐ centenas de milhares

g)Anos de participação neste projeto:

- ☐ menos de 1 ☐ 1 a 2 ☐ 3 a 5 ☐ 5 a 10 ☐ 10 a 20 ☐ mais de 20.

h)Motivação para participar do projeto:

- ...

i)Número de horas semanais, em média, dedicadas ao projeto:

- ☐ 1 a 2 ☐ 3 a 5 ☐ 5 a 10 ☐ 10 a 20 ☐ 20 a 40 ☐ mais de 40.

j)Em qual sistema operacional o projeto é desenvolvido?

- ☐ Linux ☐ OpenSolaris ☐ FreeBSD ☐ Solaris ☐ Windows ☐ Outros (especificar):

k)Qual linguagem de programação é usada na implementação?

- ☐ Java ☐ C++ ☐ C ☐ Python ☐ Ruby ☐ Perl ☐ Smalltalk ☐ PHP ☐ C# ☐ Outros (especificar):

l)Em qual sistema o software pode ser usado?

- ☐ Linux ☐ OpenSolaris ☐ FreeBSD ☐ Solaris ☐ Windows ☐ Outros (especificar):

m)Qual ferramenta de desenvolvimento é usada no projeto ?

- ☐ Eclipse ☐ NetBeans ☐ KDeveloper ☐ Vi ☐ Emacs ☐ Squeak ☐ Visual Studio ☐ Outros (especificar):

3. Caso esteja atuando profissionalmente, em relação à Empresa/Organização em que trabalha indique:

a)Tipo de organização:

- ☐ Privada ☐ Pública ☐ ONG sem fins lucrativos ☐ Outros (especificar):

b)Área do negócio da organização:

- ...

c)Número de funcionários:

- ☐ 1-5 ☐ 6 a 15 ☐ 16-50 ☐ 51-100 ☐ 101-500 ☐ mais de 500

4. Participação da empresa/organização em projetos de Software Livre:

a) Produz software livre?

- ☐ Sim ☐ Não

b) Usa software livre?

- ☐ Sim ☐ Não

c) Colabora com um software livre? (sugere modificações, relata problemas, etc.)

- ☐ Sim ☐ Não

d) Adiciona funcionalidades em um software livre?

- ☐ Sim ☐ Não

e) Personaliza/configura um software livre existente para o seu contexto?

- ☐ Sim ☐ Não

4.1 De que forma software livre é útil para a empresa/organização (ou para você):

a) Para auxílio no desenvolvimento de software?

- ☐ Sim ☐ Não

b) Como parte de outro produto?

- ☐ Sim ☐ Não

c) Personalização/configuração do software livre como forma de serviço?

- ☐ Sim ☐ Não

d) Para auxílio ao processo interno do negócio da empresa (ou seu)?

- ☐ Sim ☐ Não

e) Usado para prover serviços para clientes?

- ☐ Sim ☐ Não

5. Quais aspectos e métricas você leva em consideração para avaliar a qualidade de um software a fim de determinar se deve adotá-lo em sua empresa/organização?

Por favor, indique a relevância de cada item para você com um valor entre 1 e 10, onde 1=irrelevante e 10=essencial.

CÓDIGO-FONTE:

- () Número total de linhas de código
- () Número de classes (ou arquivos ou módulos)
- () Número de métodos ou funções
- () Número de linhas de teste
- () Número de testes por linha de código
- () Número de testes por método ou função
- () Cobertura dos testes
- () Nível de acoplamento (dependência entre os módulos/classes)
- () Nível de coesão entre módulos/classes do sistema
- () Complexidade Ciclomática
- () Existência de código duplicado
- () Separação em Módulos
- () Estruturas de dados utilizadas
- () Estilo uniforme de indentação
- () Número de colunas por linha não muito grande (p.ex.: ausência de linhas com mais de 120 colunas)
- () Número de linhas em cada método
- () Número de linhas em cada classe
- () Número de métodos em cada classe
- () Número de atributos em cada classe
- () Número de variáveis locais em cada método
- () Padrões de nomenclatura usados uniformemente
- () Uso de bons nomes para classes, métodos, variáveis, etc.
- () Existência de comentários no código (cobertura e distribuição no código)
- () Módulo para configurações (sem configurações hard-coded e limites espalhados pelo código)

TESTES:

- () Existência de testes automatizados
- () Uso de um arcabouço de testes (ex: JUnit, CPPUnit, Selenium, TestNG, DejaGNU, etc.)
- () Publicação dos resultados dos testes
- () Existência de um subgrupo (comunidade ou pessoa) especializado em testes
- () Tipos de teste disponíveis (unidade, funcional, desempenho, estresse, estrutural, etc.)
- () Testes de mutação
- () Testabilidade do código (facilidade de escrever testes)
- () Existência de testes de desempenho (benchmarks)
- () Estudo sistemático sobre o tempo de resposta
- () Estudo sobre o consumo de recursos (disco, memória, etc.)
- () Artigos e relatos dos experimentos de desempenho

MANUTENIBILIDADE:

- () Orientações documentadas para adaptação/extensão
- () Documentação/descrição da arquitetura
- () Uso de padrões de projeto e arquiteturais
- () Uso de boas práticas de desenvolvimento (documentadas ou difundidas)
- () Existência de versões/atualizações de manutenção
- () Existência de organização/empresa fornecedora de suporte
- () Tipo de manutenção/suporte oferecida
- () Linguagem de programação usada
- () Paradigma de programação (procedimental, orientado a objetos, etc.)
- () Número de linguagens usadas
- () Ambiente (plataforma) de desenvolvimento

INTEROPERABILIDADE:

- () Capacidade de comunicação/integração com outros sistemas
- () Suporte a extensão através de plug-ins
- () Uso de padrões da indústria e protocolos bem disseminados

PORTABILIDADE:

- () Lista de ambientes para os quais é oferecido suporte
- () Linguagem facilmente portátil

USABILIDADE:

- () Facilidade de instalação e configuração
- () Clareza da interface do usuário
- () Assistentes e ajuda ao usuário embutidos no software
- () Diferentes opções de uso (linha de comando, modo gráfico, acesso Web, etc.)
- () Suporte a localização/internacionalização (adição de novas línguas)

FERRAMENTAS:

- () Integração com ferramentas automatizadas de desenvolvimento
- () Scripts para configuração do processo de build
- () Possibilidade de personalização já integrada
- () Controle de versões com commits pequenos e documentados (comentários nos commits)

INDEPENDÊNCIA:

- () Dependências bem documentadas
- () Não requer instalação de outros pacotes/sistemas

COMUNIDADE:

- () Tamanho atual da comunidade de usuários do software
- () Número total de componentes de atualizações (patches) e versões (releases)
- () Número de componentes de atualizações (patches) e versões (releases) recentes
- () Tempo de vida do projeto
- () Número de contribuições da comunidade
- () Existência de diferentes grupos na comunidade
- () Existência de lista de e-mail ou Fórum de usuários
- () Atividade recente na lista de e-mail ou Fórum da comunidade
- () Lista de e-mail ou Fórum de desenvolvedores

- () Número de mensagens que já foram mandadas para fóruns, listas, blogs, etc.
- () Número de desenvolvedores
- () Reputação dos desenvolvedores

ASPECTOS DE CONFIANÇA:

- () Estágio do desenvolvimento/maturidade
- () Informações sobre a complexidade do software
- () Disponibilização de atualizações críticas (patch releases)
- () Números de erros (bugs) críticos reportados
- () Tempo de solução de erros críticos (bugs)
- () Reputação do software

FUNCIONALIDADES:

- () Disponibilidade de uma lista bem definida de funcionalidades
- () Informações/descrições sobre cada nova versão (p.ex., changelog)
- () Protótipo/exemplo/demo do software
- () Existência de comparações documentadas com produtos similares
- () Existência de produtos derivados

SATISFAÇÃO DE USUÁRIOS/CLIENTES:

- () Existência de lista de organizações que o utilizam e testemunho de usuários
- () Existência de estudos de casos documentados e histórico de uso
- () Prêmios ganhos pelo software
- () Empresas oferecendo serviços tendo o software como base

DOCUMENTAÇÃO:

- () Manual do usuário
- () Manual técnico
- () FAQ para usuários
- () FAQ técnico
- () Documentação técnica (p.ex., Javadoc)
- () Guia de instalação

SUPORTE:

- () Número de correções de erros (bugs) disponibilizadas
- () Acompanhamento de erros (bug tracking)
- () Processo definido para correção de erros (bug workflow)
- () Tempo de resposta quando detectado um erro pelo usuário (comunidade e/ou desenvolvedores)

TREINAMENTO:

- () Disponibilidade de treinamentos, tutoriais e outros materiais de aprendizagem
- () Curso oficial de treinamento

CANAIS DE DISTRIBUIÇÃO E LICENÇA:

- () Acesso ao repositório
- () Download do código-fonte e binários
- () Disponibilização em formato físico (CD/DVD)
- () Principal licença usada
- () Possibilidade de identificar uma lista de todas as licenças de todos os componentes (árvore de licenças)

Por favor, citar e/ou descrever outros aspectos ou métricas de código-fonte não citadas acima que são usadas por você ao avaliar a qualidade de um software. Use quanto espaço for necessário.

Referências Bibliográficas

- ABREU, F. Mood - metrics for object-oriented design. In: *OOPSLA 94*. [S.l.: s.n.], 1994.
- ABREU, F.; CARAPUCA, R. Candidate metrics for object oriented software within a taxonomy framework. *Journal of Systems and Software*, v. 26, n. 1, 1994.
- ALBRECHT, A. J.; GAFFNEY, J. E. Software function, source lines of code, and development effort prediction: A software science validation. *IEEE Transactions on Software Engineering*, v. 9, n. 6, p. 628–648, November 1983.
- ARTHUR, Lowell Jay. *Measuring Programmer Productivity and Software Quality*. New York: Wiley-Interscience, 1985. 292 p.
- BAKER, A.L.; ZWEBEN, S.H. A comparison of measures of control flow complexity. *IEEE Transactions on Software Engineering*, v. 6, n. 6, p. 506–512, 1980.
- BANSIYA, J.; DAVI, C. Using qmood++ for object-oriented metrics. *Dr. Dobb's Journal*, December 1997.
- BASIL, V. R.; CALDIERA, G.; ROMBACH, H. D. *Goal question metric paradigm*. <http://www.cs.umd.edu/projects/SoftEng/ESEG/papers/gqm.pdf>, 1994.
- BASIL, V. R.; ROMBACH, H. D. *TAME: Integrating Measurement into Software Environments*. University of Maryland, Computer Science Department, 1987.
- BASIL, Victor R. Tutorial on models and metrics for software management and engineering. *IEEE Computer Society Press*, EHO-167, p. 7, 1980.
- BASIL, V.R.; BRIAND, L.C.; MELO, W.L. A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering*, v. 10, n. 22, 1996.
- BASIL, V.R.; TURNER, A.J. Iterative enhancement: a practical technique for software development. *IEEE Transactions on Software Engineering*, v. 4, n. 1, p. 390–396, 1975.
- BECK, Kent; ANDRES, Cynthia. *Extreme Programming Explained: Embrace Change*. 2nd edition ed. [S.l.]: Addison-Wesley, 2004.
- BECK, Kent; FOWLER, Martin. *Planning Extreme Programming*. [S.l.]: Addison-Wesley Professional, 2000.
- BEHRENS, C. A. Measuring the productivity of computer systems development activities with function points. *IEEE Transactions Software Engineering*, v. 9, n. 6, p. 648–652, November 1983.

- BELLIN, D.; TYAGI, M.; TYLER, M. *Object Oriented Metrics: An Overview*. 1999. Web Publication.
- BENKLER, Yochai. *The Wealth of Networks: How Social Production Transforms Markets and Freedom*. [S.l.]: Yale University Press, 2006.
- BIANCO, V. del et al. *How European software industry perceives OSS trustworthiness and what are the specific criteria to establish trust in OSS*. [S.l.], October 2007. [Http://qualipso.semanticdesktop.org/xwiki/bin/view/Wiki/WP51](http://qualipso.semanticdesktop.org/xwiki/bin/view/Wiki/WP51).
- BIANCO, V. del et al. *Analysis of relevant open-source projects and artifacts - Working document wd 5.2.1, Version 0.1*. [S.l.], February 2008.
- BIANCO, V. del et al. *Definition of trustworthiness of software products and artefacts - Working document wd 5.3.1 - version 1.0*. [S.l.], June 2008.
- BIANCO, V. del et al. *Identification of factors that influence the trustworthiness of software products and artefacts - Working document wd 5.3.2, Version 2.0*. [S.l.], October 2008.
- BIEMAN, J.M.; OTT, L.M. Measuring functional cohesion. *IEEE Transactions on Software Engineering*, v. 8, n. 20, p. 254–259, 1994.
- BOEHM, B. W. *Software Engineering Economics*. N. J.: Prentice-Hall, 1981.
- BOEHM, B. W.; BROWN, J. R.; LIPOW, M. Quantitative evaluation of software quality. In: *2nd Intl. Conf. on Software Engineering*. Long Beach, California: IEEE Computer Society, 1976. p. 592–605.
- BOEHM, B.W. et al. *Characteristics of Software Quality (TRW series of software technology*. 1st edition ed. [S.l.]: Elsevier, 1978. 210 p.
- BROWN, David B. et al. A cost model for determining the optimal number of software test cases. *IEEE Transactions on Software Engineering*, v. 15, n. 2, p. 218–221, 1989.
- CARD, D. N.; AGRESTI, W. W. Measuring software design complexity. *Jornal of Systems and Software*, v. 8, n. 3, p. 185–197, June 1988.
- CERINO, D. A. Software quality measurement tools and techniques . In: *COMPSAC 86*. Washington, D. C: IEEE Computer Society, 1986. p. 160–167.
- CHIDAMBER, R. S.; KEMERER, C. F. Towards a metrics suite for object oriented design. In: *Proceedings of the OOPSLA 91 Conference*. [S.l.: s.n.], 1991. p. 197–211.
- CHIDAMBER, S.R.; KEMERER, C.F. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, v. 20, n. 6, p. 476–493, 1994.
- CHRISTENSEN, K.; FITSOS, G. P.; SMITH, C. P. A perspective on software science. *BM Systems Journal*, v. 20, n. 4, p. 372–387, 1981.
- CONTE, S. D.; DUNSMORE, H. E.; SHEN, V. Y. *Software Engineering Metrics and Models*. California: Benjamin-Cummings, 1986.

- CURTIS, B.; SHEPPARD, S. B.; MILLIMAN, P. Third time charm: Stronger prediction of programmer performance by software complexity metrics. In: *4th Int. Conf. on Software Engineering*. New York: [s.n.], 1979. p. 356–360.
- CURTIS, B. et al. Measuring the psychological complexity of software maintenance tasks with the halstead and mccabe metrics. *IEEE Transactons Software Engineering*, v. 5, n. 2, p. 96–104, March 1979.
- DEMARCO, T. *Controlling Software Projects: Management, Measurement & Estimation*. New York: Yourdon Press, 1982.
- DUBINSKY, Yael et al. Agile metrics at the israeliair force. In: *Agile 2005 Conference*. [S.l.: s.n.], 2005. p. 12–19.
- DUNSMORE, H.E.; GANNON, J.D. Data referencing: an empirical investigation. *IEEE Computer*, v. 12, p. 50–59, 1979.
- ELSHOFF, J. L. Measuring commercial pl/i programs using halstead’s criteria. *ACM SIGPLAN Notices*, v. 11, n. 5, p. 38–46, May 1976.
- FENTON, N.E.; MELTON, A. Deriving structurally based software measures. *Journal of Systems and Software*, v. 12, p. 177–187, 1990.
- FENTON, Norman E.; PFLEEGER, Shari Lawrence. *Software Metrics: A Rigorous and Practical Approach*. 2 edition ed. [S.l.]: Course Technology, 1998. 656 p.
- GAFFNEY, J. Program control complexity and productivity. In: *IEEE Workshop on Quantitative Software Models*. [S.l.: s.n.], 1979. p. 140.
- GAFFNEY, John E.; FELBER, Henry D.; ERLING, Richard W. *The Software Measurement Guidebook (Management Information Systems) (Paperback)*. [S.l.], January 1995.
- GORLA, N.; BENANDER, A. C.; BENANDER, B. A. Debugging effort estimation using software metrics. *IEEE Transactions on Software Engineering*, v. 16, n. 2, p. 223–231, Febuary 1990.
- GOUSIOS, Georgios et al. Software quality assesment of open source software. In: *Proceedings of the 11th Panhellenic Conference on Informatics*. May: [s.n.], 2007.
- GRADY, R. B.; CASWELL, D. R. *Software Metrics: Establishing a Company-Wide Program*. N. J: Prentice-Hall, 1987.
- HALSTEAD, M. H. Natural laws controlling algorithmic structure? *ACM SIGPLAN Notices*, v. 7, n. 2, p. 19 – 26, Febuary 1972.
- HALSTEAD, M. H. *Elements of Software Science*. New York: Elsevier North-Holland, 1977.
- HANSEN, W.J. Measurement of program complexity by the pair (cyclomatic number, operation count). *ACM SIGPLAN Notices*, v. 3, n. 13, p. 29–33, 1978.
- HARRISON, R. An evaluation of the mood set of object-oriented software metrics. *IEEE Transactions on Software Engineering*, v. 24, n. 6, 1998.

- HARRISON, W.; COOK, C. A micro/macro measure of software complexity. *Jornal of Systems and Software*, v. 7, n. 3, p. 213–219, September 1987.
- HARRISON, W. et al. Applying software complexity metrics to program maintenance. *Computer*, v. 15, p. 65–79, 1982.
- HAUSEN, H.L. Yet another software quality and productivity modeling-yaquapmo system sciences. In: *Proceedings of the Twenty-Second Annual Hawaii International Conference on*. [S.l.: s.n.], 1989. v. 2, p. 978–987.
- HENDERSON-SELLERS, Brian. *Object-Oriented Complexity*. [S.l.]: Prentice-Hall, 1996.
- HENRY, S.; KAFURA, D. Software structure metrics based on information flow. *IEEE Transactions on Software Engineering*, v. 5, n. 7, p. 510–518, 1981.
- HENRY, S.; KAFURA, D. The evaluation of software systems' structure using quantitative software metrics software–practice and experience. *Software - Practice and Experience*, v. 14, n. 6, p. 561–573, June 1984.
- HENRY, S.; KAFURA, D.; HARRIS, K. On the relationships among three software metrics. *Performance Evaluation Rev.*, v. 10, n. 1, p. 81–88, 1981.
- INCE, D.C.; HEKMATPOUR, S. An approach to automated software design based on product metrics. *Software Engineering Journal*, v. 3, n. 2, p. 53–56, 1988.
- ISO/IEC9126-1. *Software engineering - Product quality - Part 1: Quality Model*. 2001.
- JONES, T. C. *Programming Productivity*. New York: McGraw-Hill, 1986.
- JONES, T. C. *Applied Software Measurement: Assuring Productivity and Quality*. New York: McGraw-Hill, 1991.
- KAFURA, D.; CANNING, J. A validation of software metrics using many metrics and two resources. In: *8th Intl. Conf. on Software Engineering*. [S.l.]: IEEE Computer Society Press, 1985. p. 378–385.
- KAFURA, D.; HENRY, S. Software quality metrics based on interconnectivity. *Journal of Systems and Software*, v. 2, n. 2, p. 121–131, June 1981.
- KAFURA, D.; REDDY, G. R. The use of software complexity metrics in software maintenance. *IEEE Transactions Software Engineering*, v. 13, n. 3, p. 335–343, March 1987.
- KEMERER, C. F. An empirical validation of software cost estimation models. *Communications ACM*, v. 30, n. 5, p. 416–429, May 1987.
- KIRAKOWSKI, J.; PORTEUS, M.; CORBETT, M. How to use the software usability measurement inventory: the users view of software quality. In: *3rd European Conference on Software Quality*. Madrid: [s.n.], 1992.
- KNAFL, G. J.; SACKS, J. Software development effort prediction based on function points. In: *COMP-SAC*. Washington, D. C.: IEEE Computer Society Press, 1986. p. 319–325.

- KOLEWE, R. Metrics in object-oriented design and programming. *Software Development*, October 1993.
- LASSEZ, J. L. et al. A critical examination of software science. *Jornal of Systems and Software*, v. 2, n. 2, p. 105–112, June 1981.
- LEVITIN, A. V. How to measure software size, and how not to. In: *COMPSAC 86*. Washington, D. C.: IEEE Computer Society Press, 1986. p. 314–318.
- LI, H. F.; CHEUNG, W. K. An empirical study of software metrics. *IEEE Transactions Software Engineering*, v. 13, n. 6, p. 697–708, June 1987.
- LORENZ, M. Real world reuse. *Journal of object-oriented programming*, v. 6, 1991.
- LORENZ, M.; KIDD, J. *Object-Oriented Software Metrics*. [S.l.]: Prentice Hall, 1994.
- MACRO, A.; BUXTON, J. *The Craft of Software Engineering*. [S.l.]: Addison-Wesley, 1987.
- MANNINO, Phoebe; STODDARD, Bob; SUDDUTH, Tammy. The mccabe software complexity analysis as a design and test tool. *Texas Instruments Technical Journal*, v. 7, n. 2, p. 41–54, 1990.
- MCCABE; ASSOCIATES. *McCabe Object-Oriented Tool Users Instructions*. [S.l.], 1994.
- MCCABE, T. J. A complexity measure. *IEEE Transactions Software Engineering*, v. 2, n. 4, p. 308–320, December 1976.
- MCCABE, T. J. *Structured Testing: A Software Testing Methodology Using the Cyclomatic Complexity Metric*. [S.l.], December 1982.
- MCCABE, T. J.; BUTLER, C. Design complexity measurement and testing. *Communications of the ACM*, v. 32, p. 1415–1425, December 1989.
- MCCABE, T.J.; DREYER, L. A.; WATSON, A. H. Testing an object-oriented application. *Journal of the Quality Assurance Institute*, v. 8, n. 4, p. 21–27, October 1994.
- MCCALL, J. A.; RICHARDS, P. K.; WALTERS, G. F. *Factors in Software Quality*. N. Y.:Rome Air Development Center, 1977. RADC-TR-77-369.
- MILLS, Everaldo E. *Software Metrics*. Software Engineering Institute/SEI - Carnegie Mellon University, 1998.
- MOHANTY, S. N. Software cost estimation: Present and future. *Software-Practice and Experience*, v. 11, n. 2, p. 103–121, February 1981.
- MORRIS, K.L. *Metrics for Object-Oriented Software Development Environments*. Dissertação (Mestrado) — M.I.T., 1988.
- MUSA, J. D.; IANNINO, A.; OKUMOTO, K. *Software Reliability: Measurement, Prediction, Application*. New York: McGraw-Hill, 1987.
- MYERS, G. *Reliable Software Through Composite Design*. New York: Petrocelli/Charter, 1975.

- MYERS, G. J. An extension to the cyclomatic measure of program complexity. *ACM SIGPLAN Notices*, v. 12, n. 10, p. 61–64, October 1977.
- OVIDIO, E.I. Control flow, data flow and program complexity. In: *Proceedings of the IEEE Computer Software and Applications Conference*. [S.l.: s.n.], 1980. p. 146–152.
- PERLIS, A.; SAYWARD, F.; SHAW, M. *Software Metrics: An Analysis and Evaluation*. Cambridge, Mass: MIT Press, 1981.
- PFLEEGER, S. Use realistics, effective software measurement. In: _____. [S.l.]: Constructing Superior Software - Software Quality Institute, 2000. cap. 8.
- POTIER, D. et al. Experiments with computer software complexity and reliability. In: *6th Intl. Conf. on Software Engineering*. New York: IEEE, 1982. p. 94–103.
- PRESSMAN, Roger S. *Engenharia de Software*. [S.l.]: Makron Books, 1995.
- ROCHA, A. R. C.; MALDONADO, J. C.; WEBER, K. C. *Qualidade de software - Teoria e prática*. São Paulo: Prentice Hall, 2001.
- ROMBACH, H. D. A controlled experiment on the impact of software structure on maintainability. *IEEE Transaction Software Engineering*, v. 13, n. 3, p. 344–354, March 1987.
- ROSENBERG, L.H.; HYATT, L.E. Software quality metrics for object-oriented environments. *Crosstalk Journal*, 1997.
- RUBIN, H. A. A comparison of software cost estimation tools. *System Development*, v. 7, n. 5, p. 1–3, May 1987.
- RUSTON, H. Quantitative software models for reliability, complexity and cost: An assessment of the state of the art. In: *Workshop on Quantitative Software Models for Reliability, Complexity and Cost*. New York: IEEE, 1979.
- SATO, Danilo; GOLDMAN, Alfredo. *Uso Eficaz de Métricas em Métodos Ágeis de Desenvolvimento de Software*. Dissertação (Mestrado) — IME-USP, São Paulo, Agosto 2007.
- SATO, Danilo; GOLDMAN, Alfredo; KON, Fabio. Danilo sato, alfredo goldman, and fabio kon. tracking the evolution of object oriented quality metrics. In: *Proceedings of the 8th International Conference on Extreme Programming and Agile Processes in Software Engineering (XP 2007)*. [S.l.: s.n.], 2007. p. 84–92.
- SCHNEIDEWIND, N.F.; HOFFMANN, H. An experiment in software error data collection and analysis. *IEEE Transactions on Software Engineering*, v. 3, n. 5, p. 276–286, 1979.
- SHARBLE, R.; COHEN, S. The object-oriented brewery: A comparison of two object-oriented development methods. *Software Engineering Notes*, v. 18, n. 2, p. 60–73, 1993.
- SHEN, V. Y.; CONTE, S. D.; DUNSMORE, H. E. Software science revisited: A critical analysis of the theory and its empirical support. *IEEE Transactions on Software Engineering*, v. 9, n. 2, p. 155–165, March 1983.

- SHEN, V. Y. et al. Identifying error-prone software - an empirical study. *IEEE Transactions Software Engineering*, v. 11, n. 4, p. 317–324, April 1985.
- SHIH, T.K. et al. Decomposition of inheritance hierarchy dags for object-oriented software metrics. In: *Workshop on Engineering of Computer-Based Systems (ECBS 97)*. [S.l.: s.n.], 1997. p. 238.
- SMITH, C.P. A software science analysis of programming size. In: *Proceedings of the ACM national Computer Conference*. [S.l.: s.n.], 1980. p. 179–185.
- SOLIGEN, R.; BERGHOUT, E. *The goal question metric method - A practical guide for quality improvement of software development*. Cambridge, Great Britain: McGraw-Hill, 1999.
- STETTER, F. A measure of program complexity. *Computer Languages*, v. 9, n. 3-4, p. 203–208, 1984.
- TAKAHASHI, M.; KAMAYACHI, Y. An empirical study of a model for program error prediction. *IEEE Transactions on Software Engineering*, v. 15, n. 1, p. 82–86, January 1989.
- TROY, D. A.; ZWEBEN, S. H. Measuring the quality of structured designs. *J. Syst. and Software*, v. 2, n. 2, p. 113–120, June 1981.
- WEST, M. An investigation of c++ metrics to improve c++ project estimation, ibm internal paper. IBM internal paper. 1992.
- WOLVERTON, R. W. The cost of developing large-scale software. *IEEE Transactions Computers*, C-23, n. 6, p. 615–636, June 1974.
- WOODFIELD, S. N.; SHEN, V. Y.; DUNSMORE, H. E. A study of several metrics for programming effort. *J. Syst. and Software*, v. 2, p. 97–103, June 1981.
- WOODWARD, M. R.; HENNEL, M. A.; HEDLEY, D. A measure of control flow complexity in program text. *EEE Transactions. Software Engineering*, v. 5, n. 1, p. 45–50, January 1979.
- XENOS, M. et al. Object-oriented metrics - a survey. In: *Proceedings of the FESMA 2000 - Federation of European Software Measurement Associations*. Madrid, Spain: [s.n.], 2000. v. 6.
- YAU, S. S.; COLLOFELLO, J. S. Some stability measures for software maintenance. *IEEE Transactions Software Engineering*, v. 6, n. 6, p. 545–552, November 1980.
- YAU, S. S.; COLLOFELLO, J. S. Design stability measures for software maintenance. *IEEE Transactions Software Engineering*, v. 11, n. 9, p. 849–856, September 1985.
- YIN, B.; WINCHESTER, J. The establishment and use of measures to evaluate the quality of software designs. In: *Actas da Software Quality and Assurance Workshop*. New York: Association for Computing Machinery, 1978.
- YOURDON, E.; CONSTANTINE, L.L. *Structured Design*. [S.l.]: Prentice Hall, 1979.
- ZOLNOWSKI, J.C.; SIMMONS, D.B. Taking the measure of program complexity. In: *Proceedings of the National Computer Conference*. [S.l.: s.n.], 1981. p. 329–336.