UNIDADE 03

Medidas de Tamanho de Produto de Software

META

Apresentar as principais medidas de tamanho de software, suas aplicações práticas e limitações

OBJETIVOS

Geral

Compreender diferentes formas de medir o tamanho de produtos de software e sua importância na engenharia de software.

Específicos

- Identificar os principais tipos de medidas de tamanho aplicáveis ao código-fonte, projeto, requisitos e funcionalidades de software;
- Compreender a métrica de Pontos de Função e seu processo de cálculo;
- Reconhecer aplicações práticas das medidas de tamanho, como normalização de métricas, quantificação de reúso e avaliação de atributos de teste.

RESUMO

Nesta unidade é abordado as diferentes formas de medir o tamanho de produtos de software, considerando desde código-fonte até documentos de requisitos e funcionalidades. São apresentadas medidas como LOC, NCLOC e CLOC, discutindo suas limitações e usos. Também são exploradas medidas aplicáveis ao projeto de software e aos documentos de requisitos, com base em elementos como casos de uso e cenários. Pontos de Função, metodologia que mede o tamanho funcional do software, também é discutido. Por fim, discute-se a aplicação das medidas

de tamanho.

1. INTRODUÇÃO

Os produtos gerados no desenvolvimento de software são concretos e podem ser tratados de maneira similar a entidades físicas. Assim como em entidades físicas, os produtos de software podem ser caracterizados em termos de seu ***tamanho***. A medida de tamanho de uma entidade indica o quanto se tem dessa entidade, portanto somente o tamanho sozinho não é uma medida direta de atributos externos como esforço, produtividade ou custo. No entanto, o tamanho normalmente é um dos componentes de atributos indiretos, sendo um atributo relevante a ser medido.

Seguindo a intuição da noção de “tamanho” das entidades do mundo físico e relações empíricas consistente com a teoria da medida, qualquer medida de tamanho de produtos de software deve satisfazer as seguintes propriedades:

1. Não-negatividade;
2. Valor nulo;
3. Aditividade.

A propriedade da não-negatividade estabelece que qualquer entidade de software tem um tamanho não negativo. Por sua vez, a propriedade de valor nulo diz que um sistema com ausência de elementos deve ter tamanho nulo, isto é, zero. A aditividade afirma que o tamanho da união de dois sistemas deve ser a soma do tamanho de cada um separadamente, descontado a interseção do tamanho dos elementos em comum.

Nesta unidade são apresentadas diversas medidas de tamanho para produtos de software. Desde o tamanho do código-fonte ao tamanho de documentos de especificação. Por fim, fechamos o estudo com uma discussão das aplicações das medidas de tamanho.

2. MEDIDAS DE TAMANHO DO CÓDIGO-FONTE

A medida mais utilizada para avaliar o tamanho de um código-fonte é o ***número de linhas de código (LOC, do inglês Lines of Code)***. No entanto, algumas linhas são diferentes de outras. Por exemplo, muitos programadores utilizam espaçamentos e linhas em brancos para tornar seus programas mais legíveis. Assim se LOCs forem utilizadas para estimar o esforço de programação, então uma linha em branco não contribui para o esforço do mesmo modo que uma linha que implementa um algoritmo complexo.

Da mesma forma, linhas de comentários aumentam a legibilidade do programa e, certamente, exigem algum esforço para serem escritas. Entretanto, podem não demandar tanto esforço quanto o próprio código funcional. Diversos modelos diferentes já foram propostos para contar linhas, cada um com um propósito específico em mente, resultando em muitas formas diferentes de calcular LOCs para um mesmo programa. Portanto, é necessário esclarecer o que está sendo contato e como está sendo contado. Em particular, é preciso explicar como cada um dos seguintes elementos está sendo tratado:

- Linhas em branco
- Linhas de comentários
- Declarações de tipos de dados
- Linhas contendo vários comandos

A definição de LOC mais aceita e usada é a de Conte el al. (1986) Eles definem uma ***linha de código*** como toda linha do programa que não é um comentário nem linha em branco, independente se a linha do número de comando ou fragmento que contém na linha. Em outros termos, uma linha de código é uma linha não comentada. Para deixar isso evidente, usaremos a abreviação ***NCLOC (do inglês, NonCommented Line Of Code)*** para se referir a essa definição.

NCLOC é útil para comparação da linguagem de implementação de subsistemas, componentes, etc. Por exemplo, um estudo da distribuição Debian 2.2 identificou que 71% de NCLOC do código foi escrito em C e o restante em outras 10 linguagens diferentes (González-Barahona et al. 2001). Outra aplicação é o uso do NCLOC para avaliar o

crescimento do código de um sistema ao longo do tempo.

De certo modo, informações valiosas sobre o tamanho do programa são perdidas quando as linhas de comentário não são contadas. Pois, em muitas situações, o tamanho do programa é importante para decidir quanto espaço de armazenamento será necessário para o código-fonte ou quantas páginas será exigida para impressão. Nesses casos, o tamanho do programa também deve refletir as linhas em branco e as linhas de comentário. Assim, NCLOC não é uma medição válida do tamanho do programa nesses casos. O tamanho não comentado é um atributo razoável a ser medido quando está relacionado a perguntas e objetivos adequados. Se estiver relacionando o tamanho ao esforço, do ponto de vista de avaliação de produtividade, então o tamanho não comentado pode ser uma entrada válida. Porém, mesmo nesse caso, há margem para dúvidas, pois o tamanho em NCLOC carrega consigo uma suposição implícita de que comentários não exigem esforço real de programação e, portanto, não devem ser considerados. Como uma forma de amenizar essa questão, recomenda-se medir o número de linhas de comentários (CLOC, do inglês Comment Line Of Code). Com isso, definimos:

$$\text{Tamanho (LOC)} = \text{NCLOC} + \text{CLOC}$$

Por consequência, podemos medir a ***densidade*** de comentários de um programa como a razão entre CLOC e NCLOC.

Outras medidas de tamanho do código levam em consideração a forma que o código é desenvolvido e executado. Nesse sentido, algumas medidas desconsideram elementos do código que não geram código executável, como declarações de tipos e código de cabeçalho. O tamanho em termos do número de ***comando executáveis (ES, do inglês Executable Statements)*** conta o número de comando no programa, desconsiderando declarações de tipos, cabeçalhos e linhas de comentários. Nessa definição, vários comandos em uma mesma linha contam separadamente.

EXPLICA ATIVO

Outra medida de tamanho de código é DSI (Delivery Source Instruction). É uma medida de tamanho que conta o número de instruções do código que vai para produção. Essa medição é útil quando se deseja fazer distinção entre o tamanho do código produzido e o entregue em produção.

3. MEDIDAS DE TAMANHO DO PROJETO DE SOFTWARE

Podemos medir o tamanho de um projeto de software de forma semelhante à utilizada para medir o tamanho do código. Em vez de contar linhas de código (LOCs), contaremos os elementos do projeto. Os elementos que serão considerados na contagem dependerão do nível de abstração utilizado para expressar o projeto e dos aspectos de interesse que se deseja medir. Assim, a medida de tamanho apropriada dependerá da metodologia do projeto, dos artefatos desenvolvidos e do nível de abstração.

Para medir o tamanho de um projeto procedural, pode-se contar o número de procedimentos ou funções. Também pode-se medir o tamanho das interfaces desses procedimentos e funções em termos da quantidade de argumentos. Tais medições podem ser feitas mesmo sem o código-fonte, por exemplo, por meio da análise das APIs do sistema. Por outro lado, em níveis mais alto de abstração, pode-se contar o número de pacotes e subsistemas. O tamanho de um pacote ou subsistema pode ser medido em função da quantidade de funções e procedimentos que ele contém.

Projetos orientados a objetos adicionam novos mecanismos de abstração: classes, interfaces, objetos, operações, métodos, atributos, heranças, dentre outros. Além disso, podem incluir implementações de padrões de projetos (Gamma el al., 1994). Quando quantificamos o tamanho de projetos orientados a objetos, o foco geralmente está nas entidades estáticas, em vez das conexões entre elas ou das entidades em tempo de execução. Assim, tamanho é medido em termos de pacotes, padrões de projeto, classes, interfaces, atributos, operações e métodos. As medições mais comumente usadas são:

- NOP (Number Of Packages): números de pacotes;
- NOC (Number Of Classes): número de classes definidas no projeto;
- NOM (Number Of Operations): número de operações definidas no projeto, incluído métodos e funções globais;
- Padrões de projeto
    - Quantidade de padrões usados no projeto;
    - Quantidade de realizações por cada tipo de padrão no

EXPLICA ATIVO

O número de métodos e de atributos servem como medidas de tamanho de uma classe. No entanto, alguns estudos indicam que o número de métodos prediz melhor a chance de uma classe sofrer alterações que o de atributos (Bieman et al. 2001, 2003).

projeto;

- Número de classes e interfaces que participam da realização de cada padrão.

- Número de métodos públicos das classes;
- Número de atributos das classes.

4. MEDIDAS DE TAMANHO DE DOCUMENTOS DE REQUISITOS DE SOFTWARE

Documentos de requisitos e especificação geralmente combinam texto, gráficos, diagramas e símbolos especiais. A forma de apresentação depende do estilo, método ou notação utilizados. Ao medir o tamanho de um código ou projeto, é possível identificar entidades atômicas para contar (como linhas, instruções, bytes, classes e métodos). No entanto, um documento de requisitos pode ser composto por uma mistura de texto e diagramas. Por exemplo, uma análise de casos de uso pode incluir um diagrama de casos de uso em UML acompanhado de um conjunto de cenários que podem ser expressos em texto ou por meio de diagramas de atividades UML. Como uma análise de requisitos frequentemente envolve diferentes tipos de documentos, é difícil gerar uma medida única de tamanho.

Os elementos que são comumente usados como medidas de tamanho de documentos de requisitos são:

- Números de casos de usos, atores e tipos de relacionamentos;
- Números de cenários e comprimento do cenário, em número de passos;
- Número de classes, operações e atributos (do diagrama UML).

5. MEDIDAS E ESTIMATIVA DE TAMANHO DE FUNCIONALIDADES

EXPLICA ATIVO

FP mede a funcionalidade de documentos de especificação, mas também podem ser aplicadas a produtos de fases posteriores do ciclo de vida, com o objetivo de refinar a estimativa de tamanho e, consequentemente, a estimativa de

Muitos engenheiros de software argumentam que o tamanho pode ser enganoso e que a quantidade de funcionalidade presente em um produto oferece uma visão mais precisa do seu porte. Em especial,

custo ou produtividade.

aqueles que fazem estimativas de esforço e duração a partir de artefatos iniciais do desenvolvimento costumam preferir avaliar a quantidade de funcionalidade requerida, em vez do tamanho do produto (que ainda não está disponível nesse estágio). Como um atributo distinto, a funcionalidade requerida capta uma noção intuitiva da quantidade de funções contidas em um produto entregue ou em uma descrição de como o produto deve se comportar.

CURIOSIDADE

Pontos de função foi inicialmente proposto por Allan Albrecht, da década de 70, quando trabalha na IBM.

Pontos de função (FP, do inglês Function Points) é uma abordagem desenvolvida com o objetivo de fornecer informações sobre o tamanho de um sistema de software a um modelo de custo ou produtividade, com base em produtos iniciais mensuráveis, em alternativa a estimativa de linhas de código (LOCs).

Para computar o número de pontos de funções, deve-se primeiro computar o ***UFC (Unadjusted Function point Count)***. Para tal, deve-se determinar, a partir da especificação do software, o número de itens dos seguintes tipos:

- **Entradas Externas (EE)** – dados ou comandos inseridos no sistema pelo usuário;
- **Saídas Externas (SE)** – dados processados enviados para fora do sistema, incluindo cálculos e formatações;
- **Consultas Externas (CE)** – interações que solicitam e recebem dados sem alterar o sistema;
- **Arquivos Lógicos Internos (ALI)** – grupos de dados mantidos e gerenciados pelo próprio sistema;
- **Arquivos de Interface Externa (AIE)** – dados usados pelo sistema, mas mantidos por sistemas externos.

Para cada um dos itens, um valor de complexidade (subjetivo) é atribuído usando uma escala ordinal: baixa, médio e complexo. O UFC é calculado pelo somatório dos itens ponderados conforme peso definido pelo padrão da *International Function Point Users Group:*

Tipo de Componente	Complexidade		
	Baixa	**Média**	**Alta**
Entrada Externa (EE)	3	4	6
Saída Externa (SE)	4	5	7
Consulta Externa (CE)	3	4	6
Arquivo Lógico Interno (ALI)	5	7	10
Arquivo de Interface Externa (AIE)	7	10	15

O cálculo do FP é feito multiplicando-se o UFC por um *fator de complexidade técnica (TCF, do inglês Technical Complexity Factor)*. O TCF envolve a contribuição dos seguintes 14 fatores:

- F1 – Confiabilidade de backup e recuperação;
- F2 – comunicação de dados;
- F3 – processamento distribuído;
- F4 – desempenho;
- F5 – uso intenso de configuração;
- F6 – entrada de dados on-line;
- F7 – facilidade de operação;
- F8 – atualização on-line;
- F9 – interface com o usuário;
- F10 – complexidade de processamento;
- F11 – reusabilidade;
- F12 – facilidade de instalação;
- F13 – múltiplos locais;
- F14 – facilidade de mudança.

Cada um desses 14 fatores é avaliado com numa escala ordinal, de 0 a 5, em que 0 significa que o fator é irrelevante, 3 médio e 5 que o fator é essencial para o desenvolvimento do sistema. Em seguida, o valor do TCF é calculado pela seguinte fórmula:

$$\text{TCF} = 0{,}65 + 0{,}01\sum_{i=1}^{14} F_i$$

O TCF varia de 0,65 (se for atribuído 0 para todos os 14 fatores) a 1,35 (se todos os fatores receberem valor 5). O valor final dos FP é calculado pela multiplicação do UFC pelo TCF:

$$FP = UFC \times TCF$$

Para ilustrar o processo do cálculo de pontos de funções para um sistema, considere um sistema simples de cadastro de clientes para uma empresa com as seguintes funcionalidades:

- O usuário pode cadastrar, alterar e excluir dados de clientes;
- O sistema permite consultar os dados dos clientes por nome ou CPF;
- Os dados dos clientes são armazenados em um banco de dados;
- O sistema se integra com serviço externo de verificação de CPF; e
- O sistema gera um relatório com estatísticas básicas de clientes cadastrados.

Na primeira etapa do cálculo deve-se identificar cada um dos 5 itens do UFC e classificá-los conforme a sua complexidade. Nesse sistema há 3 entradas externas (EE) – operações de cadastrar, alterar e excluir clientes; 2 consultas externas (CE) – as consultas por nome e CPF; 1 saída externa (SE) – o relatório; 1 arquivo lógico interno (ALI) – a tabela do banco de dados; e 1 arquivo de interface externa (AIE) – o sistema de verificação do CPF. Suponha uma avaliação de complexidade média para EE, SE e ALI, e baixa para CE e AIE. Assim, conforme os valores para cada complexidade, o UFC para esse sistema é:

$$UFC = 3 \times 4 + 1 \times 5 + 2 \times 3 + 1 \times 7 + 1 \times 7 = 37$$

Após calcular o UFC, deve-se atribuir um grau de relevância para o sistema para cada um dos 14 fatores para calcular o TCF. Considerando o valor 3 (grau mediano) para cada um dos 14 fatores, o TCF é:

$$TCF = 0,65 + 0,01(14 \times 3) = 1,07$$

Por fim, o valor final dos pontos de função é

EXPLICA ATIVO

No contexto de orientação a objetos, há o conceito de *object points* e *use case points* cujo objetivo é estimar funcionalidades diretamente dos diagramas de classes / uso, ao invés de mapear as construções para FP.

MULTIMÍDIA

FP = UFC x TCF = 37 x 1,07 = 39,59

Function Points or Use Case Point?

Apesar da popularidade e aplicação prática de pontos de funções, essa medida apresenta diversas limitações técnicas e conceituais que comprometem sua precisão e validade como medida científica e rigorosa. Uma das críticas mais recorrentes diz respeito à subjetividade do TCF, pois os valores atribuídos aos fatores são feitos a partir do julgamento do analista e é altamente subjetiva. Portanto, diferentes analistas podem atribuir valores distintos para o mesmo sistema, dependendo de suas experiências, interpretações ou contextos organizacionais. Essa variabilidade impacta diretamente no cálculo do TCF, afetando o resultado da contagem de pontos de função.

Outro problema é a possibilidade de dupla contagem da complexidade. A complexidade funcional já está, em parte, considerada nos próprios pesos atribuídos aos tipos de componentes funcionais (por exemplo, entradas, saídas, arquivos), que variam conforme a complexidade (baixa, média ou alta). No entanto, essa complexidade é novamente incorporada indiretamente por meio dos fatores de ajuste. Isso pode levar a uma superestimação da complexidade do sistema, distorcendo o valor final da medida.

Além disso, a métrica pode apresentar valores contraintuitivos. Em alguns casos, sistemas com maior complexidade real podem receber menos pontos de função do que sistemas mais simples, dependendo da forma como os elementos funcionais são mapeados e classificados. Isso levanta dúvidas sobre a consistência interna da métrica e sua capacidade de refletir corretamente o esforço necessário para implementar determinado software.

Por fim, do ponto de vista teórico, os pontos de função não satisfazem os critérios da teoria representacional da medida. Além disso, usa uma escala ordinal como sendo uma escala de razão nos cômputos do UFC e TCF. Assim, a medida não atende aos requisitos formais de

validade e mensurabilidade científica.

Em síntese, embora os pontos de função sejam úteis para propósitos práticos de estimativa e comparação, é essencial reconhecer suas limitações metodológicas e conceituais. Seu uso deve ser cauteloso, sempre complementado por outras fontes de informação e, quando possível, calibrado com dados históricos reais.

6. APLCAÇÕES DAS MEDIDAS DE TAMANHO

Embora muitas vezes associadas apenas à estimativa de esforço ou custo, as medidas de tamanho têm aplicações mais amplas e estratégicas, especialmente quando utilizadas de forma integrada a outras métricas e processos. Dentre suas diversas utilidades, três aplicações se destacam: a normalização de outras medidas, a quantificação do reúso de software e a avaliação de atributos relacionados a testes.

A primeira e talvez mais comum aplicação é a normalização de outras medidas. Em muitas situações, medidas brutas, como número de defeitos, esforço em horas-homem ou custo, não são suficientes para comparações significativas entre projetos. Projetos maiores, naturalmente, tendem a apresentar mais defeitos ou a demandar mais recursos. Por isso, ao dividir essas métricas pelo tamanho do software, obtém-se métricas normalizadas como “defeitos por ponto de função”, “custo por LOC” ou “esforço por unidade funcional”. Essa normalização permite comparações mais justas e significativas entre projetos de diferentes portes, além de fornecer indicadores de produtividade, qualidade e eficiência mais precisos.

Outra aplicação essencial das medidas de tamanho é na quantificação da quantidade de reúso de software. O reúso, seja de componentes internos ou bibliotecas externas, é um dos pilares da engenharia de software moderna, contribuindo para redução de custos, melhoria da qualidade e aumento da produtividade. Para que o reúso possa ser efetivamente mensurado e gerenciado, é necessário conhecer o

tamanho do software reutilizado em relação ao tamanho total do sistema.

Por fim, as medidas de tamanho também são aplicadas na mensuração de atributos relacionados ao teste de software. O esforço de teste, a cobertura de casos de teste e o número de defeitos esperados estão diretamente relacionados ao tamanho funcional do sistema. Por exemplo, ao conhecer o número de pontos de função, é possível estimar o número de casos de teste necessários ou o esforço proporcional a ser investido na atividade de validação. Além disso, métricas como “defeitos por ponto de função testado” ou “cobertura de testes por função” são utilizadas para avaliar a eficácia do processo de teste e orientar melhorias contínuas na qualidade do produto.

7. CONSIDERAÇÕES FINAIS

Nesta unidade, exploramos diferentes abordagens para medir o tamanho de produtos de software, desde métricas baseadas em código-fonte até medidas funcionais como os pontos de função. Compreendemos que o tamanho, embora não esteja diretamente ligado a atributos como esforço ou custo, é um atributo relevante para se medir. Destacamos a importância de escolher a medida adequada ao contexto e aos objetivos do projeto, considerando também as limitações e subjetividades envolvidas. Por fim, reforçamos o papel estratégico das medidas de tamanho na normalização de métricas, na avaliação de reúso e na gestão de testes de software.

REFERÊNCIAS

BIEMAN, J.; JAIN, D.; YANG, H. *Design patterns, design structure, and program changes: an industrial case study*. In: INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE, 2001, Florence.

BIEMAN, J.; STRAW, G.; WANG, H.; MUNGER, P. W.; ALEXANDER, R. *Design patterns and change proneness: an examination of five evolving systems*. In: INTERNATIONAL

SOFTWARE METRICS SYMPOSIUM, 2003, Sydney.

CONTE, S. D.; DUNSMORE, H. D.; SHEN, V. Y. *Software engineering metrics and models*. Menlo Park: Benjamin-Cummings, 1986.

GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. *Padrões de projeto: soluções reutilizáveis de software orientado a objetos*. 1. ed. Porto Alegre: Bookman, 2000.

GONZÁLEZ-BARAHONA, J. M.; PÉREZ, M. A. O.; DE LAS HERAS QUIRÓS, P.; GONZÁLEZ, J. C.; OLIVERA, V. M. *Counting potatoes: the size of Debian 2.2*. Upgrade, v. 2, n. 6, p. 60–66, dez. 2001.