Métricas de Software

Brito, Rui
PG22781
Universidade do Minho
ruibrito666@gmail.com

Silva, António
PG22820
Universidade do Minho
anolsi@live.com

Resumo

1 Introdução

O processo de validação e certificação de softtware é cada vez mais importante, assim como necessário para a construção do mesmo. Mas como podemos afirmar se um programa é bom ou mau? Só existem bons programas se existirem maus programas, e são os conhecimentos, a experincia da interpretação e leitura de vários programas, que nos fornece da noção de um bom ou mau programa. No entanto é importante saber quais as caracteristicas que podem ser utilizadas, e que ao mesmo tempo têm uma posição relevante na decisão de um bom ou mau software.

2 Métricas de Software

Métricas de software são um conjunto de medidas de alguma propriedade do software ou da sua especificação. São, no fundo, um tentativa de munir a Engenharia de Software de ferramentas de medição, transversais a todas as ciências e engenharias. Estas métricas podem ser uteis na planificação de um projecto, na estimativa de custo, em debugging, optimização de performance e também garantias de qualidade.
Existem, então, diversas métricas capazes de contribuir com feedback quanto à qualidade de um software, no entanto, nem todas são objectivas ou podem ser usadas sem conhecimento humano. Estamos então a falar de métricas que devem ser calculáveis, facilmente compreensiveis, que possam ser testadas, que sejam passiveis de estudos estatisticos e que consequemente se expressem em alguma unidade, por ultimo, deverão ser repetiveis e independentes do observador.

3 LOC - Lines of Code

O numero de linhas de código é provavelmente a métrica mais vulgar e comum entre as diferentes linguagens, assim como a mais evidente. Pretende-se com o número de linhas de código analisar a dimensão do código, pois como é do senso comum da engenharia de software, "um bom código não é um código grande".

3.1 Como é medida

Como medir as LOC é algo debatível, pois usar esta métrica para comparar um código com 10000 linhas de código com um de 100000 é bastante mais útil do que comparar um de 10000 com um de 11000. Diferenças de pelo menos um ordem de magnitude são indicadores clássicos da complexidade software bem como das horas/homem dispendidas. Existem dois tipos de LOC, físicas (SLOC) e lógicas (LLOC). SLOC são uma medida pura das linhas de um código (contanto com linhas em branco, a não ser que estas sejam mais de 25% do código) enquanto as LLOC tentam apenas medir o número de linhas executáveis, algo que se prende obviamente com a linguagem em questão. Por exemplo:

```
for (i = 0; i < 100; i++) printf("sou
    ␣uma␣linha"); /* comentario */
```

No exemplo acima temos[1]:

- 1 SLOC
- 2 LLOC (for e printf)
- 1 linha de comentário

```
    /* comentario */
for (i = 0; i < 100; i++)
{
    printf("sou␣uma␣linha");
}
```

Neste exemplo temos:

- 5 SLOC
- 2 LLOC (for e printf)
- 1 linha de comentário

4 Métricas de Complexidade Ciclomática de McCabe

É usada para medir o número de caminhos linearmente independentes no código de um software.

4.1 Como é medida

Se um programa não tivesse instruções de salto (if) nem ciclos (for,while,etc...), a complexidade do mesmo seria

[1] O código apresentado é apenas uma linha corrida de código

1, uma vez que só existe um caminho a percorrer. Se o código tivesse uma única instrução de salto, entao, haveriam 2 caminhos distintos que o programa poderia eventualmente tomar. Um, onde a condição é avaliada como verdadeira, outro onde é avaliada como falsa. Obviamente, isto pode ser representado através de um grafo direccionado, com uma aresta a conectar dois vértices se houver a possibilidade de um passar o controlo ao outro. Assim, a complexidade M é definida como: $M = E - N + 2P$ onde

- E = número de arestas do grafo;
- N = número de vertices do grafo;
- P = número de componentos ligados (vértices de saída).

5 Métricas de Halstead

São métricas que se baseiam na complexidade do problema e das suas funções/métodos, estas métricas têm em conta por exemplo o número de operadores distintos, ou numero de operandos distinitos. O objectivo é identificar propriedades quantificáveis do sofware, bem como a relação entre as mesmas. Esta métrica, não é apenas uma métrica de complexidade.

5.1 Como é medida

Para um dado problema, sejam:

- n_1 = Número de operadores distintos;
- n_2 = Número de operandos distintos;
- N_1 = Número total de operadores distintos;
- N_2 = Número total de operandos distintos;

Destes números, várias métricas podem ser derivadas, por exemplo:

- Vocabulário do programa: $n = n_1 + n_2$;
- Comprimento do programa: $N = N_1 + N_2$;
- Comprimento do programa calculado: $\widehat{N} = n_1 \log_2 n_1 + n_2 \log_2 n_2$;
- Volume: $V = N \times \log_2 n$;
- Dificuldade: $D = \frac{n_1}{2} \times \frac{N_2}{n_2}$;
- Esforço: $E = D \times V$;
- Tempo necessário para programar: $T = \frac{E}{18}$;
- Número de bugs entregues: $B = \frac{E^{\frac{2}{3}}}{3000}$;

6 Tempo de Execução

Esta é uma metrica que indirectamente se encontra relacionada com a complexidade do problema, no entanto é uma métrica bastante importante na avaliação da qualidade de um software. O que é importante? O que devemos medir?
Num produto de software podemos não estar interessados no tempo de execução da sua totalidade mas sim de apenas uma pequena zona critica, identificar esta zona critica é, então, algo importante. Ainda mais, um bom conhecimento do comportamento de um produto de software podemos levar a encontrar bugs de run-time muito mais facilmente (erros lógicos ou limites de arrays, por exemplo).

7 Número de erros

A existência de erros em qualquer programa é um indicador por si só de má qualidade, ou melhor, de que no exacto momento a que se submeteu à avaliação é um software inválido. Os erros podem ter várias origens, mesmo que o programa seja de qualidade. Erros podem ser induzidos pelo SO, por versões divergentes de um compilador, por falta de uma biblioteca, etc... É, portanto, algo a ter em conta no desenho de uma peça de software.