Assessing Modeling Languages, metrics and tools

No Author Given

Department of Informatics, University of Minho
Campus de Gualtar, 4710-057 Braga, Portugal

Abstract. Any traditional engineering field has metrics to rigorously assess the quality of their products. From a long time ago, engineers know that the production process is not all; the output must comply with the rules and good-practices, must satisfy the requirements and must be competitive.
Professionals in the new field of software engineering started a few years ago to define metrics to appraise their product: individual programs and software systems.
This concern motivates the need to assess not only the outcome but also the process and tools employed in its development. In this context, assessing the quality of programming languages is a legitimate objective; in a similar way, it makes sense to be concerned will models and modeling approaches, as more and more people starts the software development process by a modeling phase.
In the paper we discuss the quality of modeling languages, introducing and motivating the topic, presenting metrics, and comparing tools.

Keywords: : Modeling Languages, Software/Language Quality, Software/Language Metrics, UML

1 Introduction

A model is a representation of reality aiming for the simplification of some complex object.

Models are built so that we can better understand the system being developed. They help us to visualize a system as it is or as we need it to be. Models allow us to specify the structure and behavior of a system, they provide the guidance lines/blueprints in constructing a system, and finally, models document the decision taken for a given system;

Some models are best described textually, other graphically. All interesting systems exhibit structures that transcend what can be represented in a programming language.

A modeling language is a language whose vocabulary and rules focus on the conceptual and physical representation of a system.

Specifying means building models that are precise, unambiguous and complete. The UML addresses the specification of all the important analysis, design and implementation decision.

That aside, when assessing a modeling language we might infer on its quality.

The software functionality, trustability, usability, efficiency, maintability and portability.

* Importância do uso de métricas num projecto: no que consistem, o que medem, o que ajudam a melhorar, etc. - esta parte liga-se com o que sugiro abaixo - sugiro partir de uma definição geral de métricas;

Nowadays, metrics become increasingly essential for Software Engineering: they are crucial for quality assessment and reengineering processes. In *Forward Engineering* they are used to measure the software quality and estimate cost and effort of software projects [FP98]. In the field of *Software Evolution*, they can be both used to identify stable or unstable parts of a system as to determine where refactoring can be or have been applied [DDN00]. They even can be used for assessing the quality and complexity of software systems in *Software Reengineering* or *Reverse Engineering* [CC90].

When focusing on the field of Object-Oriented (OO) systems, many metrics have been proposed for assessing the design of a software system. However, most of the existing approaches involve the analysis of the source code and cannot be applied in earlier stages of the development process. In fact, it is not always simple to apply the existing metrics in this earlier stages. As the Unified Modeling Language, proposed by Booch, Jacobson and Rumbaugh [BRJ05] has became a standard for expressing OO systems, apply metrics to these models enables an early estimate of development efforts, implementation time, complexity and cost of the system under development.

* Especificar o tipo de métricas sobre o qual nos vamos focar (UML) - o paragrafo acima ja fala 1pc, completar se tiverem+ideias

** estrutura do artigo - proposta a completar

In this paper, we will introduce and discuss the major existing metrics for UML models, and focus on present a set of tools designed for measure UML projects. In what follows, Section 2 is devoted to the metrics assessment process. In Section 3 we describe the principal measurements applicable to the most popular UML diagrams. Then, in Section 4 we present some tools designed for apply metrics to UML models and the results of applying them a real case-study. We conclude in Section 5 with a comparsion between the presented tools (.....).

2 Metrics Assessment

–Entra o assunto da Secção 2 do artigo do Azevedo, JJ e Tiago.

* Explicar aqui que as métricas não podem ser observadas, por si só, fora do contexto. Este problema prende-se essencialmente com o facto deste tipo de medições ser empírica, ou baseada em métodos empíricos.

* Explicar em maior detalhe o que se entendo como validação de métricas (Kaner e Walter Bond).

3 Applying Metrics To UML Models

As métricas para avaliação de software estão bem mais desenvolvidas que as métricas para linguagens de modelação. É então interessante ter em conta o estudo já realizado sobre métricas para avaliação de código, como é o caso das CK Metrics, como ponto de partida para métricas de modelação.
As CK Metrics, umas das primeiras métricas para o modelo Orientado a Objectos (OO), foram propostas por Chidamber e Kemerer [CK94]. O conjunto das CK Metrics consiste em seis métricas: Weighted Methods Per Class (WMC), Depth of Inheritance Tree (DIT), Number of Children (NOC), Coupling between Object Classes (CBO), Response For a Class (RFC), e Lack of Cohesion in Methods (LCOM). Estas métricas foram depois adaptadas para linguagens de modelação, tal como é mostrado em [MP06b]. De seguida, será explicado como são calculadas cada uma das métricas referidas.

3.1 Métricas CK

Weighted methods per class (WMC): Esta métrica diz respeito à complexidade que cada método tem para cada classe. Assim, para se obter o valor desta métrica soma-se as complexidades que cada método pertencente a uma classe tem. Se considerarmos a complexidade de cada método como medida unitária então a métrica WMC para um classe é igual ao número de métodos definidos nessa classe, referimo-nos a isto como WMC1. A métrica WMC1 para uma classe pode ser obtida pelo diagrama de classes de um modelo UML, identificando a classe e contando o número de métodos que essa classe implementa. Aternativamente podemos considerar a complexidade de cada método como o McCabe Cyclomatic Complexity, que referimos como WMCcc. Os diagramas de actividade, sequência e comunicação contêm informação relevante para o WMCcc, mas é igualmente plausível que os diagramas de estado possam ser usados para calcular este valor para a classe como um todo.

Depth of inheritance tree (DIT): Esta é uma medida de profundidade da classe relativamente à sua árvore de herança. Esta métrica define-se por ser igual à distância máxima desde a classe até à sua super classe root na árvore de herança. Esta métrica pode ser calculada para uma classe fazendo a união de todos os diagramas de classe num modelo *UML* e atravessando a hierarquia de herança desta classe.

Number of children (NOC): Esta métrica representa o número de decendentes imediatos de uma determinada classe. Esta métrica pode ser obtida ao juntar todos os diagramas de classes numa modelação *UML* e verificar todas as

relações de herança da classe.

Coupling between object classes: Duas classes estão relacionadas se o método de uma classe usa uma variável de instância ou um método da outra classe. Uma estimativa desta métrica pode ser obtida a partir dos diagramas de classes, contando o número de classes relacionadas com a classe em questão e contando todos os tipos de referência dos atributos e todos os parâmetros dos métodos da classe. Para obter um valor mais fidedigno, pode ter em conta informação dada pelos diagramas comportamentais, de forma a obter mais informação sobre o uso das variáveis de instância e de métodos de invocação. O diagrama de sequência, por exemplo, oferece informação directa sobre interacções entre métodos de classes diferentes.

Response for a class (RFC) - esta métrica é a contagem do número de métodos que potencialmente poderão ser invocados por um objecto de uma dada classe. O número de métodos de uma classe pode ser obtido a partir de um diagrama de classes, mas o número de métodos de outras classes que são invocadas por cada um dos métodos da classe requer informação à cerca do comportamento dessa classe. Esta informação pode ser derivada a partir da inspecção de vários diagramas de comportamento (diagramas sequencias/diagramas de actividade), de modo a obter a identidade dos métodos invocados.

Lack of cohesion in methods (LCOM)- ou seja, falta de coesão entre métodos. Calcular esta métrica para uma dada classe envolve descobrir, para cada possível par de métodos, se os conjuntos de variáveis de instância acedidos por cada método têm uma interseção que não um conjunto vazio. Para ser possível computar um valor para esta métrica, informação do modo de uso das variáveis de instância pelos métodos de uma classe é essencial. Esta informação não pode ser obtida através de um diagrama de classes. No entanto, o valor máximo para esta métrica pode ser computado usando o número de métodos na classe. Diagramas contendo essa informação sobre o uso das variáveis, por exemplo, os diagramas sequenciais podem ser usados para calcular esta métrica.

O conjunto de métricas aqui definidas são referidos em vários papers e sem sombra de dúvidas são as mais estudadas, e também as mais utilizadas para avaliar modelos *UML*.
Estas focam-se mais nos diagramas de classes visto estes serem os que mais facilmente se relacionam com código e é preciso ter em consideração que como estas regras derivam directamente do paradigma Orientado-a-Objectos, é mais fácil aplicá-las aos diagramas de classes. Para além disso este tipo de diagramas do ponto de vista da implementação dão uma visão mais geral do sistema que modela.
Estas métricas em particular são detalhadas por McQuillan e Power em [MP06a].
Também existe o SDMetrics software, que avalia além destas, um conjunto mais extenso de métricas, que analisam outros diagramas além do de classes, como

por exemplo os diagramas de estados e de actividades.
Como grande parte das métricas derivam de fórmulas matemáticas, no capítulo seguinte serão apresentadas algumas que são essenciais para o cálculo dos valores das métricas apresentadas anteriormente.

–Junção da Sec.1 do artigo dos Pedros e Ulisses com Sec.3 do artigo do Ismael e Daniela.

* Explicar que as métricas sobre UML derivam directamente do OO.

* Falar em CK metrics e falar apenas nos tipos de CK metrics mais importantes.

* Relacionar as tabelas do paper do Ismael e Daniela com os tipos de CK metrics.

4 Tools for UML Metrics Specification

Nowadays, the most popular UML tools for software application development are Visual Paradigm for UML[1] and Poseidon for UML[2]. They enable a visual environment to model software, which reduces the complexity of software design. However, they not support metrics specification - it is required to use other tools, designed for this task. In the next subsections, we will introduce the leading systems on the quantitative analysis of UML models structural properties, and put them to test for explore their features with a real case-study.

One of the tools that we are going to address is SDMetrics[3], a design measurement tool for UML models. Although its core is open source and available under the GNU Affero General Public License, SDMetrics GUI it is not freely distributed. In order to explore all the system features, SDMetrics staff gently provided us an Academic License which we sincerely would like to thank.

The core functionalities of SDMetrics include:

- the configurable XMI parser for XMI1.0/1.1/1.2/2.0/2.1 input files;
- the metrics engine to calculate the user-defined design metrics;
- the rule engine to check the user-defined design rules.

The other tool we put to test is the Sparx System Enterprise Architect[4], a team-based modeling environment. It embraces the full product development lifecycle, supporting both software design, requirements management, and metrics calculation for Use Case Diagrams. Thus, it allow to estimate the complexity of the project in a earlier stage, as well as the complexity associated with each actor of the system.

The case-study used to explore this tools (.....) ***Apresentar agora o Case study: tipo de projecto, principais características, contexto onde se insere - ou seja, para e pq foi desenvolvido... Se virmos q a descriçao fica mt grande, +vale

[1] Available at `http://www.visual-paradigm.com/product/vpuml/`
[2] Available at `http://www.gentleware.com/products.html`
[3] SDMetrics can be found at `http://www.sdmetrics.com/`
[4] `http://www.sparxsystems.com.au`

manter a ideia inicial de por 1a sec só p o CS, ou até ser a primeira subsecção desta parte - ideia a pensar

***NOTAS: –Referir para cada ferramenta:
* o que são capazes de fazer (algumas métricas que calculam),
* se é proprietário, open source, licença académica,
* que input recebem (XML, formato próprio, etc...)
* o que devolvem (se fazem análise apenas das métricas em separado ou se tentam ir mais longe e dão resultados sobre qualidade, tamanho do projecto, etc...)

4.1 SDMetrics

SDMetrics is a very complete design measurement tool, analysing a wide range of UML diagrams, including Class, Usecase, Activity and Statemachine diagrams. For each type of diagram, this tool generates several metrics.
For example, **NumAttr** metric, one of the metrics that has already been adressed in this paper, is calculated from Class diagrams. Other one is **ExtPts**, wich is calculated from Usecase diagrams, and gives us the number of extension points of a given use case.

SDMetrics is written in java, and provides us a graphical user interface. The type of source files it receives to analyse are *XMI*[5] files, most modeling tools support project exportation in XMI.
This tool allows us to access the results from different views. We will approach the ones that seem the most important:

- **Metric Data Tables** provides a table that matches each UML model element analysed (table line) to it's value for each metric (table column);
- **Histograms** provides a graphical distribution for each design metric;
- **Design Comparison** provides us a mean to compare the structural properties of two *UML* designs. It is very useful to compare the same design in different iterations of the development, or to compare an alternative design to the current one.
- **Rule Checker** design rules and heuristics detect potencial problems in the UML design such as:
  - incomplete design such as unnamed classes, states without transitions;
  - violation of naming conventions for classes, attributes, operations, packages;
  - etc.
- **Catalog** this view provides us with the definitions of the metrics, design rules, and relation matrices for the current data set, and provides literature references and a glossary for them.

[5] *XMI* (**X**ML **M**etadata **I**nterchange) is an *OMG* (Object Management Group) standard to generate XML-based representations of UML and other OO data models.

Not a view, but one of the most advanced features in this software is the possibility of defining Custom Design Metrics and Rules. The new metrics are defined in a XML file, with a very particular format, the *SDMetricsML* (SDMetrics Markup Language).
The SDMetrics tool doesn't provide a direct notion of good/bad quality of the design model. Despite that, on the SDMetrics website we can find several indications of how to interpret each output.

Results

4.2 Sparx Systems Enterprise Architect

Sparx Systems Enterprise Architect is another tool that provides modeling of UML diagrams. It supports mind map diagrams and project management, to provide full traceability from requirements specification to deployment end implementation. This tool also provides also some metrics evaluation to compute the complexity of a project only based on Use Case diagrams.

To perform this evaluation, the user needs to provide a level of implementation complexity to each interaction with authors. This task can be done when defining the Use Case descriptions or when performing the metrics evaluation.

To evaluate the metrics, Enterprise Architect have a wizard, that gives also other options to do the complexity analysis. This options manipulates the Technical complexity and also Environment complexity and be used to adapt the evaluation and perform a better result on estimation.

Also can be filtered the Use Cases used in evaluation. To filter can be used manual selection or regular expressions over use case name. This kind of filtering ables the project to be distributed and evaluated individually.

The final result presented is the maximum number of working hours needed to perform the implementation.

Results

4.3 IBM Rational

–Permite modelação + aplicação de métricas

Results

5 Conclusion

Comparação entre as diversas ferramentas.

References

Grady Booch, James Rumbaugh, and Ivar Jacobson. *Unified Modeling Language User Guide, The (2nd Edition)*. Addison-Wesley Professional, 2 edition, May 2005.

E.J. Chikofsky and II Cross, J.H. Reverse engineering and design recovery: a taxonomy. *Software, IEEE*, 7(1):13 –17, jan 1990.

S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Trans. Softw. Eng.*, 20:476–493, June 1994.

Serge Demeyer, Stéphane Ducasse, and Oscar Nierstrasz. Finding refactorings via change metrics. In *In Proceedings of OOPSLA '2000 (International Conference on Object-Oriented Programming Systems, Languages and Applications*, pages 166–177. ACM Press, 2000.

N. E. Fenton and S. L. Pfleeger. *Software Metrics: A Rigorous and Practical Approach*. International Thomson Computer Press, London, UK, 1998.

Jacqueline A. Mcquillan and James F. Power. A definition of the chidamber and kemerer metrics suite for the unified modeling language. Technical report, 2006.

Jacqueline A. Mcquillan and James F. Power. Some observations on the application of software metrics to uml models. Technical report, Department of Computer Science National University of Ireland, Maynooth, 2006.