Poster: UMLx: A UML Diagram Analytic Tool for Software Management Decisions

Kan Qi
University of Southern California
United States
kqi@usc.edu

Barry W. Boehm
University of Southern California
United States
boehm@usc.edu

ABSTRACT

A UML diagram analytic tool called UMLx is proposed, which automatically extracts information from UML diagrams to facilitate decision making in risk management, planning, resource allocation, and system design, based on a set of proposed metrics.

CCS CONCEPTS

• **Software and its engineering → Unified Modeling Language (UML)**; **Software development process management**; **Object oriented development**;

KEYWORDS

Software sizing, effort estimation, UML analysis, unified modeling language (UML), project management, architecture quality evaluation, object-oriented modeling, use case driven development

ACM Reference Format:
Kan Qi and Barry W. Boehm. 2018. Poster: UMLx: A UML Diagram Analytic Tool for Software Management Decisions. In *ICSE '18 Companion: 40th International Conference on Software Engineering Companion, May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3183440.3194969

1 INTRODUCTION

UML diagrams have been the major artifacts of the model-based approach of software engineering. It comprises 14 diagrams to describe the behavioral and structural aspects of a system. Also a subset of the diagrams have been adopted in the attempts to bring light-weight modeling into Agile software development[1]. In addition to their effectiveness in describing the behavioral and structural aspects of the system being developed, system specifications defined using UML diagrams are also effective indicators of software size, an accurate estimate of which can further be used to estimate effort, cost, and schedule[2] and guide decision making in various areas of software project management.

In this paper, we introduce a UML diagram analytic tool called UMLx to derive information from the UML diagrams by the proposed metrics and discuss their effects on various software management decisions.

ICSE '18 Companion, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5663-3/18/05.
https://doi.org/10.1145/3183440.3194969

2 THE AUTOMATED FRAMEWORK

2.1 Model Definition and Analytic Procedures

2.1.1 Parsing XML files. XMI files exported from UML modeling tools are first parsed by the rules mapping the tagged elements of XMI files to the UML elements defined in the UML metamodels. A hierarchy of UML elements is constructed by establishing the associations among the UML elements using the UUIDs of the referenced tagged elements. This hierarchy includes multiple use cases expounded by activity diagrams, sequence diagrams, and object analysis diagrams and a domain model composed of a set of classes to describe a system's behavior and structure[3].

2.1.2 User-System Interaction Model (USIM) Construction. USIM is defined based on the hierarchy of the parsed UML elements, which includes the following key elements: activities (A), precedence relations (PR), system boundary (SB), system scope (SCP), stimuli (STL), and components (CMP). Specifically, activity, object analysis, and sequence diagrams are converted into control flow graphs (CFG) by graph transformation algorithms[4] to PR among A. SB and STL are identified by the rules defined based on the associations with actors. SCP defines the effective transactions. CMP that realize A are identified from class diagrams.

2.1.3 Transaction Identification. The CFGs are traversed (based on DFS) to identify the independent paths with the stimulus nodes as the entry points and the exit points determined by the rules based on the system boundary.

2.1.4 Transaction Classification. The identified transactions characterized by the components and activities they are implemented upon, the interface complexity of the components, the referenced data elements, etc. These properties help classify the transactions into different levels of complexity.

2.1.5 Model Profiling. The information at the different levels: UML elements, transactions, use cases, and models, is profiled based on the proposed metrics.

2.2 Evaluation Metrics

2.2.1 NT, SWT-I, SWT-II. The number of transactions (NT) has been the major factor that defines system functional size metrics[5][6]. Transactions can be classified into different complexity levels by their properties, for example, operational complexity (defined as the number of activities), called transaction length (TL), and interface complexity (defined as the average number of the external methods ($|M|/|I|$) exposed by an interface, where $|M|$ is the total number of methods exposed by $|I|$ interfaces), called the degree of a transaction (TD). Different weights are assigned to the complexity levels

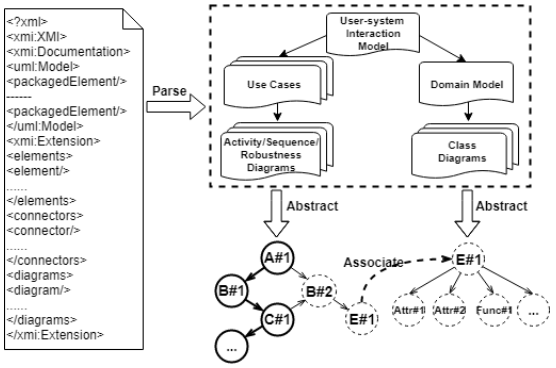

Figure 1: USIM is constructed from XMI files to support automated transaction identification and classification.

to represent their relative influences on software size. The sum of weighted transactions by this method is called SWT-I. Also, if class diagrams are input, the data element types (DETs) referenced by a transaction can be identified. The size metric that takes the number of DETs into consideration is called SWT-II. The relevance of the size metrics in predicting project effort is presented in the top section of Figure 2, which is based on an empirical study of 19 student projects. Early effort estimations help decide a feasible scope to avoid the risks of being over schedule or budget. Also, we can prioritize use cases by the business value per unit of use case complexity - BV/(NT/SWT-I/SWT-II) - to improve return on investment (ROI).

2.2.2 CT. Seven operational characteristics are identified to characterize a transaction: interface management (INT), external input (EI), external inquiry (EQ), data management (DM), control (CTRL), external call (EXTCLL), and external invocation (EXTIVK). Each category essentially represents a type of developers needed to implement the transactions of that category. Here we provide an example of calculating the percentage distribution of development effort of different types based on the categorized transactions (CT) in the middle section of Figure 2.

2.2.3 ATC, ATD. The average number of components of a transaction (ATC) represents the expected number of system components a developer needs to understand, create, modify, and test when implementing a transaction of a use case. Similarly, the average transaction degree (ATD) indicates the expected interface complexity when implementing a transaction of a use case. Therefore, the product of ATC and ATD (ATL-ATD) represents the architectural difficulty a developer needs to deal with when implementing a transaction of a use case.

3 CONCLUSIONS

In this paper, we introduce UML diagram analytic tool called UMLx that is capable of analysing commonly used UML diagrams to derive the information that facilitate decision making in various areas of software management, including project scope management, task prioritization, resource allocation, and architecture sufficiency assessment. To enable the automated analyses, we propose USIM that fills the gap between the sizing model and UML metamodels by

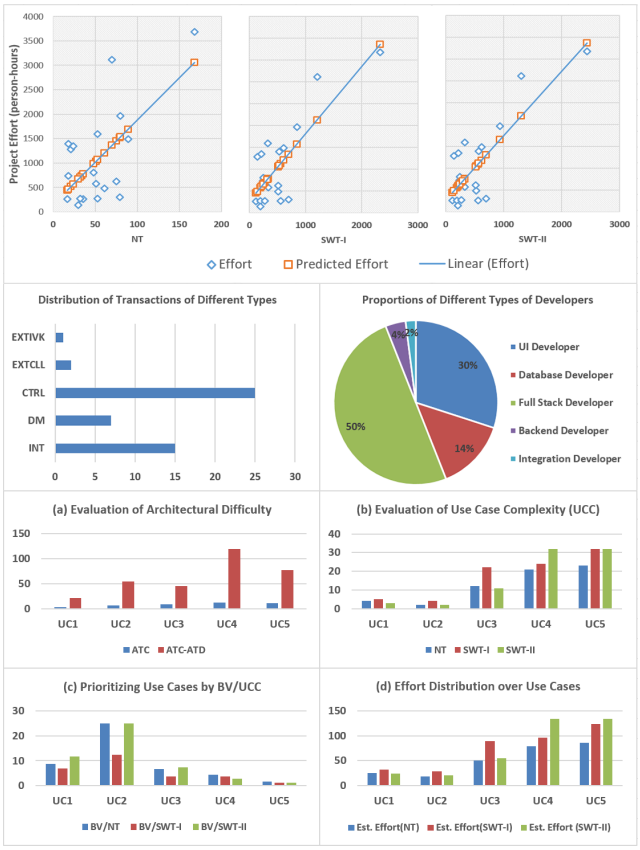

Figure 2: Transaction related statistics for software PM.

synthesizing information from the UML diagrams to computationally support transaction identification and classification. Examples of how the derived data affect decision making are provided.

3.1 Future Directions

To extend the capability of the automated framework to analyzing other types of diagrams would extend the use of the tool. More data points are needed to further evaluate the relevance of the proposed metrics in making the software management decisions. To support data collection, the tool needs to be implemented compatible with the XMI files exported from different UML modeling tools.

REFERENCES

[1] Doug Rosenberg, Barry Boehm, Bo Wang, and Kan Qi. Rapid, evolutionary, reliable, scalable system and software development: The resilient agile process. In *Proc. of the ICSSP'17*, 2017.
[2] Barry W. Boehm. *Software engineering economics.* Prentice-Hall, Englewood Cliffs, N.J, 1981.
[3] Ivar Jacobson. *Object-oriented software engineering: a use case driven approach.* ACM Press ;Addison-Wesley Pub, [New York] :Wokinghams, 1992.
[4] Debasish Kundu, Debasis Samanta, and Rajib Mall. An approach to convert xmi representation of uml 2. x interaction diagram into control flow graph. *ISRN Software Engineering*, 2012, 2012.
[5] International Function Point Users Group (IFPUG). *Function Point Counting Practices Manual*, 4.1.1 edition, 2002.
[6] G. Karner. *Metrics for Objectory. Diploma thesis.* PhD thesis, University of Linkoping, 1993.