A Semantic-Based Framework for Automated Software Functional Size Measurement: A Case Study of Odoo

An Hu
Graduate School of Advanced Science and Technology
Japan Advanced Institute of Science and Technology
Nomi, Japan
shorehu@outlook.com

***Abstract*—Functional Size Measurement (FSM) is a critical procedure in software development that quantifies software functions and enables estimation of workload in software projects. As software development significantly accelerated by AI coding, the high cost and inconsistent drawbacks of manual FSM have become serious. However, there has not been a perfect solution for automated FSM as software requirements are highly unstructured. In this research, a case study is conducted on an open-source software "Odoo" which developed a set of metadata to specify its contents. The correlativity between Odoo metadata and IFPUG (International Function Points Users Group, an FSM standard) functions is identified. Then a semantic-based framework is proposed to represent the metadata and calculate IFPUG function point automatically. The results show strong rank correlation between the calculated functional points and the implementation sizes of Odoo modules which verify the feasibility of the new FSM path.**

Keywords—functional size measurement, function point, IFPUG, semantic, metadata

I. INTRODUCTION

A. Functional Size Measurement

Functional Size Measurement (FSM) [1] is a concept for software functions quantification. It is not a specific method, but an upper seat concept of specific implement practice.

FSM measures functions required by customers and could be recognized by users, independent of technical implementation methods. Software applications following the same requirement would have the same FSM results even if they are developed by different programming languages with different workloads. Contents only used by technical staff and cannot be reached by software interface like system logs are not included in FSM.

The metric of FSM is called Function Point (FP). FP acts as an objective indicator of software functions to help project managers estimate reasonable time, financial and personnel costs in software developing projects.

Conventionally FSM is conducted manually on requirement documents in natural language, but human counters are slow, expensive and there is inconsistency between different counters.

B. IFPUG

International Function Points Users Group (IFPUG) [2] is a widely adopted FSM standard with ISO certification. It measures two types of functions, data function and transaction function.

A data function refers to a group of user-recognizable, logically related data or control information in the software. There are 2 types of data functions: Internal Logical File (ILF) and External Interface File (EIF) which are maintained within and outside the measured software respectively [3].

A transaction function refers to a transaction in the measured software with business data input or output. There are 3 types of transaction functions: an External Input (EI) is a transaction in which one or more ILF functions changed by data or control information outside the measured application; an External Output (EO) is a transaction in which processed data or control information from the software are exported; an external Inquiry (EQ) is a transaction in which original data or control information from the software are exported.

There are 3 attributes in the functions: a Data Element Type (DET) is a user-recognizable field within a data function or delivered by a transaction function; A record Element Type (RET) is a subgroup of a data function; File Type Referenced (FTR) is the number of data functions referenced in transaction functions.

The Function Point of each data & transaction function is calculated by their attributes, and the total FP is the sum of all functions.

In this study, Function Point Counting Practices Manual Release 4.3.1 [3] is taken as measurement standard.

C. Ontology and Knowledge Graph

An ontology is an explicit specification of conceptualization, emphasizing its role in supporting shared and reusable knowledge representation [4]. It is constructed by defining concepts, relations and constraints.

Knowledge graph [5] conveys not only concepts, but also entities in the real world. It provides human-machine readable representations by instantiating ontologies, with entities denoted as nodes and their relations as edges.

D. Odoo Metadata

Odoo is the largest open-sourced ERP (Enterprise Resource Planning) software in the world with a market share of more than 50% [6]. It's a popular, comprehensive software project with high representative. This study is conducted in odoo-19.0 Community Edition.

Odoo metadata is a set of technical vocabulary that specifies the software elements in Odoo. Specifically, it's a family of system models prefixed with ir.*[1] and defined in 'base' module[2], the core system module of Odoo. When a module in Odoo is installed, the framework loads its tables,

[1] '_name' field in .py files in odoo-19.0/addons/base/models/

[2] each .py file in odoo-19.0/addons/base/models/ refers to a piece of metadata

XXX-X-XXXX-XXXX-X/XX/$XX.00 ©20XX IEEE

fields, views, actions, menus and access rules declared into these models, forming a self-describing registry that the system queries at run time to determine the application's data structures, user interface and entry points.

Compared with ambiguous requirement documents in natural language, Odoo metadata is explicit, formalized and covers most items that IFPUG functions measure. So, it's an ideal testbed to determine IFPUG rules.

TABLE I. ODOO METADATA IN XML FILES

Source: odoo-19.0 Community Edition [7]
odoo-19.0\addons\sale\views\ sale_order_views.xml

```
...
1237 <record
id="mail_followers_edit_action_from_sale"
model="ir.actions.act_window">
1238  <field name="name">Add/Remove
Followers</field>
1239  <field
name="res_model">mail.followers.edit</field>
1240  <field name="view_mode">form</field>
1241  <field name="target">new</field>
1242  <field name="context">
{'default_res_model': 'sale.order',
'default_res_ids': active_ids}</field>
1243  <field name="binding_model_id"
ref="sale.model_sale_order"/>
1244    <field
name="binding_view_types">list,kanban</field>
1245  </record>
```

TABLE II. ODOO METADATA IN PYTHON FILES

Source: odoo-19.0 Community Edition [7]
odoo-19.0\addons\sale_management\models\sale_order_template_line.py

```
...
21   sale_order_template_id = fields.Many2one(
22     comodel_name='sale.order.template',
23     string='Quotation Template Reference',
24     index=True, required=True,
25     ondelete='cascade')
26     sequence = fields.Integer(
27     string="Sequence",
28   help="Gives the sequence order when displaying a
list of sale quote lines.",
29     default=10)
...
```

Table I is an example of Odoo metadata in XML files. The record "ir.actions.act_window" is a metadata denoting an output of data in the interface and the fields below are attributes of it. They specify the implement details like data source and display format.

Table II is an example of Odoo metadata in python files. Metadata locates in declarations at the beginning of python files, separated from the method bodies. "sale_order_template_id" is a metadata denoting a data field, and the items below are attributes of it. They specify the storage details like "which table the field belongs to".

As shown in Tables I and II, Odoo metadata is not just a scheme to present software contents for human readers which may not accord with actual developing job, but it plays a critical role in the actual codes and directly regulates software functions' implementation.

II. RELATED RESEARCH

A. Automated FSM

There have been studies exploring automated FSM by IFPUG standard to address this problem.

The first research route is to conduct FSM on complete codes, not requirement documents. Klusener presented a method to calculate Function Point on Cobol language by mapping FP concepts to technical objects [8]. Kusumoto et al. [9] and Edagawa et al. [10] conducted measurements on Java language respectively with more sophisticated analyses. A relevant standard for code measurement is proposed in 2019 which is consistent with IFPUG [11].

However, although codes are more machine readable than natural language, a wide variety of language types and implementation methods lead to inconsistency. A more serious problem is that these methods can only be performed after development and are not helpful to project management.

More research adopts a different approach: developing formalized models from requirement documents at first and then conducting automated measurement based on the models.

UML models are the most frequently used. Uemura et al. [12] and Živkovič et al. [13] utilized UML model to represent software requirement respectively. In 2024, Malanowska and Zabuski updated Uemura et al.'s method to the new version of UML [14]. However, UML models do not fit software development precisely so that there may be inconsistency between requirement and UML model or between UML model and the actual implement. Any of them could lead to inaccuracy of the measurement.

Abrahão and Poels [15] made use of OO-Method which allows for automatic software development without coding manually [16]. By representing requirement documents with OO-Method the alignment problem between requirement and implement is perfectly solved. However, applying OO-Method is very costly, which is verified in the paper. This is equivalent to advancing coding process to model generation and the measurement is still conducted after development.

Adem and Kasirun developed an automated tool whose input is requirements documents processed in goal and scenario modeling [17]. de Freitas et al. developed a model specifically for representing software requirements and measured on it [18].

Quesada-López et al. conducted automated measurement on a company's software with a framework called FastWorks. All applications in the company were developed by the framework so that no extra work is needed for software measurement [19]. But this method relied on a specific developing paradigm.

Lavazza et al. [20] optimized the calculation of Function point by machine learning, but the identification of functions still relied on manual work.

To the best of the author's knowledge, there is no existing research conducting FSM by metadata of software. To address this gap, this study conducts a case study on Odoo software and explores to represent software requirement by Odoo metadata.

B. Semantic Methods in Software Management

Wen and Katt [21] used ontology and SPARQL statements to represent and calculate assurance metrics in software. In the field of FSM, Bagriyanik and Karahoca [22] proposed a semantic method by COSMIC, another FSM standard. They

designed an ontology to represent the software elements and COSMIC functions, generated knowledge graph with human help and then calculate function points automatically by java programs on the knowledge graph.

To the best of the author's knowledge, there is no existing research that uses semantic method to conduct automated FSM with IFPUG standard. To address the gap, this study developed an ontology to represent metadata and IFPUG functions, then calculate function point in a knowledge graph. By adopting semantic approach, developing a new model with high learning cost is avoided, and the automation and traceability level are improved.

C. AI-Assisted FSM

The advent of AI brought changes to FSM. Zhao et al. [23] used LLM to read the requirement documents and realized fully automatic counting. However, the measure procedure was black box.

In this study, AI-assistance is used as a complementary route for FSM based on metadata by estimating on the codes that metadata does not cover.

III. Conceptual Framework

The theoretical ground of this study is SECI model [24] which proposed the knowledge spiral of transformation between tacit knowledge and explicit knowledge. 2 processes are defined in this model:

- Externalization, the process that tacit knowledge is interpreted in a clear and understandable form and transformed to explicit knowledge
- Combination, the process that different explicit knowledge of different types and sources is reorganized and merged into integrated explicit knowledge

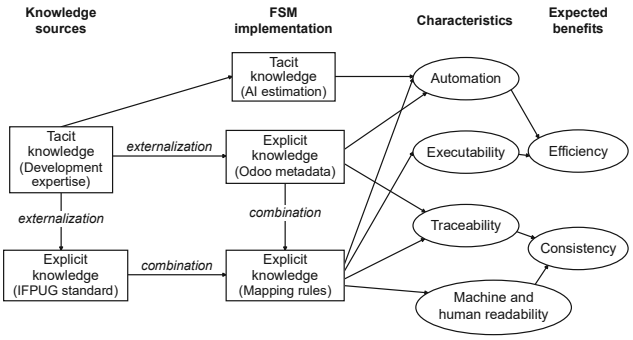


Fig. 1. Conceptual framework.

As Fig. 1 shows, traditionally FSM knowledge locates in software development expertise of humans. The tacit knowledge is externalized by industrial community to be measurement standards like IFPUG. It is also externalized by software developers like Odoo as metadata to assist in development. However, these two parts of explicit knowledge are not associated with each other.

Most existing research focused on externalization by proposing requirement models, but combination is largely ignored. Omitting this crucial step in knowledge creation led to difficulties in automated FSM.

This study makes use of metadata which is a set well developed explicit knowledge so that the measurement is fully automated and traceable. By combining the two parts of explicit knowledge with semantic methods which are executable and machine & human readable, efficiency and consistency in FSM is improved.

IV. Measurement Case on Odoo Software

A. Data Collection

The data source of this study is the source code of Odoo 19.0 community edition [7]. This is to ensure data accuracy; it does not mean that the framework should be conducted on complete source code. In practical applications, metadata generated before completed code development is sufficient.

The target population is user-recognizable modules while system modules are excluded because FSM focuses on user visible functions. The sampling frame is all 34 application modules[3] in Odoo and their dependency modules, which is complete enumeration.

TABLE III. Application modules of Odoo 19.0 Community

Category	Application modules
Sales	**crm, point_of_sale**, pos_restaurant, sale_management, contacts
Marketing	**mass_mailing, survey**, marketing_card, website_event, mass_mailing_sms
Supply Chain	**stock, mrp**, purchase, maintenance, repair
Human Resources	**hr, fleet**, hr_holidays, hr_expense, hr_recruitment, hr_skills, hr_attendance, lunch
Accounting	account
Website	website, website_sale, website_slides, im_livechat, website_hr_recruitment
Services	project
Productivity	mail, calendar, data_recycle, project_todo

As shown in Table III, 8 modules in bold text form the developing set for generating parameters while other 26 modules form the test set for validation. 2 largest modules from each of 4 typical categories are selected as purposive sampling.

B. Mapping Rules Between Metadata and IFPUG Functions

1) Judging Criteria of IFPUG Functions

According to the definitions mentioned in Section I-B, the criterion of IFPUG functions could be refined as the following elements and features,

Elements:

1) Data group
2) Subgroup of data groups
3) The data group a data field belongs to
4) Data fields
5) Transaction
6) Data fields on transactions
7) Transactions to which data fields belong
8) The data groups referenced by a transaction function

Features:

1) Whether a data group is user-recognizable

[3] where 'application' item in '__manifest__.py' file is 'true'

2) Whether a data group is maintained within or outside the boundary of the application

3) Whether a data field is user-recognizable

4) Whether a data field is unique and non-repeated

5) Does a transaction process data or control information which comes from outside the application, maintain one or more ILFs, and/or alter the behavior of the system?

6) Does a transaction process data or control information which sends data or control information outside the application and processes, calculates, or derives the data before sending?

7) Does a transaction process data or control information which sends data or control information unchanged outside the application?

2) Related Odoo Metadata

The following 11 related metadata and attributions are used to identify IFPUG functions,

1) ir.module.module
name: module name
application: application module
dependency: the modules it depends on

2) ir.model: record data groups
model: data groups' name
transient: is it a temporary data group
abstract: is it a technical data group

3) ir.model.inherit: record model inherit[4] relationship
model_id: successor model
parent_id: predecessor model
parent_field_id: foreign key

4) ir.model.fields: record data fields
model: the data group the field belongs to
name: field name
ttype: field type
relation: the target model's name of a relational field
relation_field: foreign key
related: associated path
store: does it stored in a database
required: is it an obligatory field
on_delete: would it be deleted with its target data group
compute: is it computed by other fields

5) ir.ui.view: contents and formats of an interface
model: the data group it present
type: view type
arch_db: XML structure of the view
inherit_id: is it an inherit view
mode: is it primary or extension
priority: the priority of view adoption
key: foreign key
active: is it being used

6) ir.actions.act_window: an operation triggers an interface
res_model: which data group the present data come from
binding_model_id: links to menu of a certain data group
context: behavioral parameters

7) ir.actions.act_window.view
act_window_id: an action in ir.actions.act_window
view_mode: the type of view
view_id: which view it present

8) ir.ui.menu: record menus in user interface
action: the action it links to
parent_id: its parent menu

9) ir.actions.report: record printable reports
model: the data group it present
report name: template of its contents

10) ir.actions.server: record actions executed on server side
model_id: the data group it executes on
state: action type

11) ir.model.data: external identifiers of system elements
module: the module that declares the element
name: identifier of the element
model: the data group the element relates to
res_id: id of the element

3) Mapping Logics

According to IFPUG counting practices [3] and the meanings of Odoo metadata, judging logic of all elements and features summarized in Section IV-B1 is as follows,

Elements:

1) A data group should not be transient or abstract. While measuring a module, data groups maintained by it are counted while the data maintained by other modules are not. Also, data groups that are referenced by data groups maintained by it are counted. This convention is set by this study, not IFPUG standard.
involved metadata: ir.model, ir.model.fields, ir.actions.act_window, ir.ui.view

2) Subgroup is a data group that "is inherited by another data group" or "associates with another data group by a foreign key and would be deleted along with it". Subgroup should not user-recognizable independently.
involved metadata: ir.model.inherit, ir.model.fields

3) The data group to which a data field belongs is recorded in ir.model.fields. The data group to which it belongs is the data group or the parent model of the data group.
involved metadata: ir.model.fields

4) Defined in ir.model.fields

5) Views in interfaces triggered by a button or a menu item are transactions; all the output reports are transactions. While measuring a module, transactions referenced derived from Interfaces containing views or output reports that declared by the module is counted. This convention is set by this study, not IFPUG standard.
involved metadata: ir.ui.menu, ir.ui.view, ir.actions.report, ir.actions.act_window, ir.model.data, ir.actions.server

6) Data fields in transactions are the fields located in views and reports.
involved metadata: ir.ui.view, ir.actions.report

[4] based on an existing model, reuse and extend its definitions

7) Data fields belong to transactions that referenced views or output reports the fields located in.

8) The data groups referenced by a transaction function are the functions to which all data groups in the transaction belong.
involved metadata: ir.model.fields

Features:

1) A data group is user-recognizable if it is presented by a reachable action, or at least one of its field shows on a transaction.
involved metadata: ir.ui.menu, ir.model.data, ir.model.fields

2) Data groups are maintained by the model declares it and models declare actions whose form view allows creating its records. While measuring on a module, data groups maintained by it are considered maintained within the application. The model maintained by other Odoo modules are considered maintained outside the application. This convention is set by this study, not IFPUG standard.
involved metadata: ir.model.data, ir.actions.act_window

3) Data fields that are presented in views or reports are user-recognizable.
involved metadata: ir.ui.view, ir.actions.report

4) Exclude fields that align with a field in another data group, fields that do not store in database and foreign key.
involved metadata: ir.model.fields

5) Transactions with form type view opened by a button or a menu item and temporary data groups triggered by a button or a menu item.
involved metadata: ir.ui.view, ir.model, ir.actions.act_window.view

6) Transactions with pivot, graph type view opened by a button or a menu item and transactions with output reports.
involved metadata: ir.ui.view, ir.actions.act_window.view, ir.actions.report

7) Transactions with list type view opened by a button or a menu item.
involved metadata: ir.ui.view, ir.actions.act_window.view

4) Identification of IFPUG Functions

In summary, IFPUG functions are judged as following statements by elements and features in Section IV-B3,

- ILF: element1 accord with features 1&2
- EIF: element1 accord with feature1 but does not accord with feature2
- DET of data function: element4 accord with features 3&4 that belongs to element3
- RET: element2
- EI: element5 that accord with feature5
- EO: element5 that accord with feature6
- EQ: element5 that accord with feature7
- DET of transaction function: element6 accord with features 3&4 that belong to element7
- FTR: element8

So far, the mapping rules between Odoo metadata and is generated and are utilized in the principal path of this study.

C. Residual Estimation

There are residual elements in Odoo software that metadata does not cover. 5 types of residuals are identified in this study:

1) FTR of an EO transaction in reports

2) FTR of an EI transaction in wizard view

3) Functions invoked by a button

4) Functions in website pages

5) Server actions

Values of the former 2 types are represented by 2 parameters calculated from the code. The parameters are set as the average values of all such elements in developing set modules. The corresponding values of elements in test set modules are all considered equal to the parameters.

Contents of the latter 3 types could not be inferred from the relevant code directly as the distinguishment between transaction functions is concerned. To address this problem, an AI-assisted protocol is developed to estimate the functions in the residual. It's assumed that the UFP (unadjusted function point) of each type is the same in all modules. The relevant code pieces are extracted from the developing set.

A pair of parameters 'p' and 'm' are designed in this study to represent the estimation of function point. Samples were stratified by residual-type * transaction-type combination. 'p' refers to the admitted rate of code pieces in percentage. 'm' refers to the mean UFP of admitted code pieces.

In the protocol, the source code of all 3 types of residuals would be upload to an AI with task specification.

AI's tasks:

a) to recognize the transaction types

b) to judge the admission of each code piece

c) to identify the tables and fields in the codes

The average UFP of each residual-type * transaction-type combination is calculated by the functions recognized by AI. Thus the 'p' and 'm' parameters are determined.

The parameters are used for all modules, not only developing modules. The function point of a test module for each residual-type * transaction-type is the production of the p, m and the number of corresponding residual-type * transaction-type elements in the module. The number of website pages could not be extracted from Odoo metadata. It's the only function element that metadata does not provide in this study.

D. Semantic Methods[5]

An ontology is developed to represent, store and calculate the information items in this study.

Table IV shows the concepts in the ontology and their related Odoo metadata in principal path. Each concept correlates to a core element in the metadata related to IFPUG

[5] the codes: https://github.com/syrzdl-oss/odoo-ifpug-counting

functions, as summarized in Section IV-B2. Properties of the concepts in the ontology are designed to represent the attributions of metadata and the critical intermediate results while calculating IFPUG functions.

TABLE IV. PRINCIPAL PATH CONCEPTS IN THE ONTOLOGY

Concept	Meaning and source metadata	Property number
Module	An Odoo module (ir.module.module)	1
OdooModel	A data group (ir.model)	
PersistentModel	A data group not transient or abstract	5
TransientModel	A transient data group	2
AbstractModel	An abstract data group	2
OdooField	A data field (ir.model.fields)	8
OdooView	A view (ir.ui.view)	11
QWebTemplate	A report template (ir.ui.view)	6
WindowAction	An action that opens views (ir.actions.act_window)	5
ReportAction	An action that prints a report (ir.actions.report)	3

Table V shows the concepts in the ontology in complementary path. The numbers of elements for each residual-type*transaction-type combination and the p/m parameters are included.

TABLE V. ESTIMATION PATH CONCEPTS IN THE ONTOLOGY

Concept	Meaning	Property number
ServerAction	An action that runs code on the server (ir.actions.server)	5
ObjectButton	A button in a view (ir.ui.view)	3
HttpRoute	A website or portal entry point	1
CalibrationStat	A calibrated parameter it's properties include: residual type, transaction type, p value, m value	5
EstimatedFunctionGroup	Code-side estimate result for a module and a residual type: n × p × m.	7

Table VI shows the concepts representing IFPUG functions.

TABLE VI. IFPUG FUNCTION CONCEPTS IN THE ONTOLOGY

Concept	Properties	Property number
DataFunction		
ILF	Including det, ret, ufp	9
EIF	Including det, ret, ufp	9
TransactionFunction		
EI	Including det, ftr, ufp	11
EO	Including det, ftr, ufp	10
EQ	Including det, ftr, ufp	10

By incorporating metadata items, estimation parameters and IFPUG functions into the same ontology, the subsequent calculation could be held altogether.

To implement the mapping rules generated in Section IV-B, they are transformed into SPARQL statements that could be executed in knowledge graphs.

E. Measurement Processes

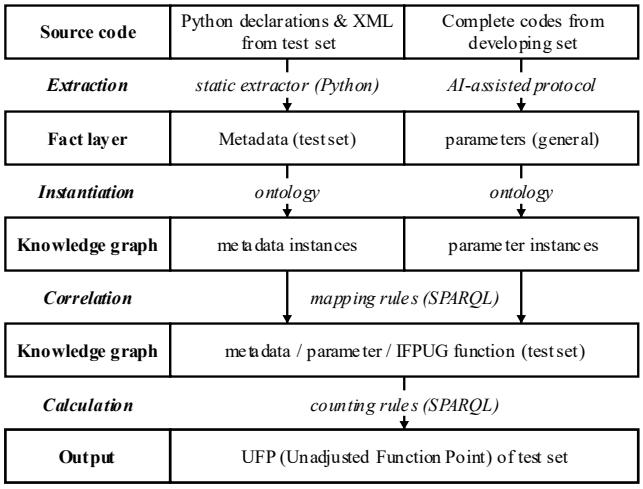


Fig. 2. System architecture.

There are two parallel paths in this measurement as shown in Fig. 2.

The principal path was based on Odoo metadata. First, the metadata related to measurement was extracted from xml files and declarations of python files in the source code of test set modules by python program. The extracted metadata was instantiated with the ontology generated in Section IV-D into a knowledge graph. Then the IFPUG function instances were generated in the knowledge graph with SPARQL statements in Section IV-D executing on the metadata instances. The function point was calculated by the function instances.

The complementary path was for residual functions that Odoo metadata does not cover. The AI-assisted protocol developed in Section IV-C estimated parameters from source code of developing set modules. Then the parameters were also instantiated into the knowledge graph. The function point was calculated by the parameter instances. Table VII shows the estimated p/m parameters.

TABLE VII. PARAMETERS CALIBRATED ON THE DEVELOPING SET

Residual type	Category	p	m	Mean FTR
Button	EI	0.2222	3.519	—
	EO	0.0082	4.000	—
Webpage	EI	0.3846	3.450	—
	EO	0.2885	4.733	—
	EQ	0.1731	3.111	—
Server action	EI	0.1961	3.000	—
Report	EO	—	—	3.571
Wizard	EI	—	—	2.412

The final function point of test set modules was the sum of the 2 values. Table VIII shows the calculated UFP values of the 26 test modules.

TABLE VIII. UFP OF THE TEST MODULES

Module	UFP	Module	UFP
mail	1049	hr_attendance	156
account	849	repair	139
project	445	calendar	127
website	444	website_event	113
website_sale	422	hr_skills	110
hr_recruitment	415	mass_mailing_sms	108

Module	UFP	Module	UFP
website_slides	400	marketing_card	70
hr_holidays	361	data_recycle	55
im_livechat	360	sale_management	54
purchase	318	contacts	50
lunch	252	project_todo	44
hr_expense	204	website_hr_recruitment	38
maintenance	185	pos_restaurant	37

V. Validation

A. Correlation Between Functional Size and Implementation Size

The implementation size and functional size of software modules should be correlated with each other, although may not be strictly proportional.

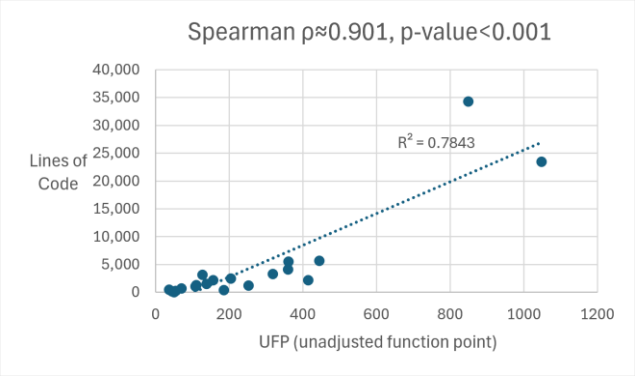


Fig. 3. Correlation coefficient, n=26.

Fig. 3 presents the correlation coefficient between implementation size in lines of codes [25] and functional size in UFP. The sample size is all 26 test set modules. The Spearman ρ is more than 0.9 which means these factors are highly correlated with each other. This result supports the correctness of the functional size calculated in this study.

B. Sensitivity Analysis for p/m Parameters

A sensitivity analysis for p/m parameters is conducted by elasticity [26] on the modules' UFP value with respect to p/m parameters. Set U as the total UFP and A as the UFP of the principal path; the elasticity E is given in (1). 1% increase in p*m results in an E% increase in UFP.

$$E = (U - A) / U \tag{1}$$

Fig. 4 shows the E values of residual types aggregated by module categories. The results show that the E values vary among different types and categories. Some values like webpage type in website modules are extremely high. It indicates that p/m parameters are necessary or the results in such modules would deviate seriously.

Module Category	Sales	Marketing	Supply Ch.	HR	Accounting	Website	Services	Productivity
Button(E)	0.0263	0.0575	0.0597	0.0290	0.0542	0.0126	0.0270	0.0322
Webpage(E)	0.0442	0.1936	0.0096	0.0357	0.0459	**0.4014**	0.0652	0.1875
Server(E)	0.0021	0.0011	0.0106	0.0045	0.0082	0.0030	0.0067	0

Fig. 4. E values of Odoo modules.

VI. Discussion

A. Limitations

There are still some limitations on this research.

First, the method relies heavily on the structure of a certain version of Odoo software so that it's vulnerable to version changes.

Second, as Odoo metadata is not adopted by other software, this method is difficult to transfer to other applications.

Also, this study is only conducted on application modules in Odoo. But there are many contents in system modules presented on the interface. These contents are omitted in the measurement.

The usage of AI may also be a problem. Different AI products may bring about diverse outcomes which lead to inconsistency in the measurement.

Elements like the numbers of website pages cannot be extracted from Odoo metadata. In this study they are extracted from completed source code, but in actual software development projects they should be estimated in other approaches.

Last but not the least, this study is performed on a completed software version with entire source code. According to the mechanism this method should be able to conduct on metadata generated before source code is completed. The feasibility in practical software development projects has not yet been verified.

B. Implications

1) Metadata

The largest contribution of the study is identifying software metadata as a new source of automated FSM, apart from complete source code, requirement documents and the models based on them.

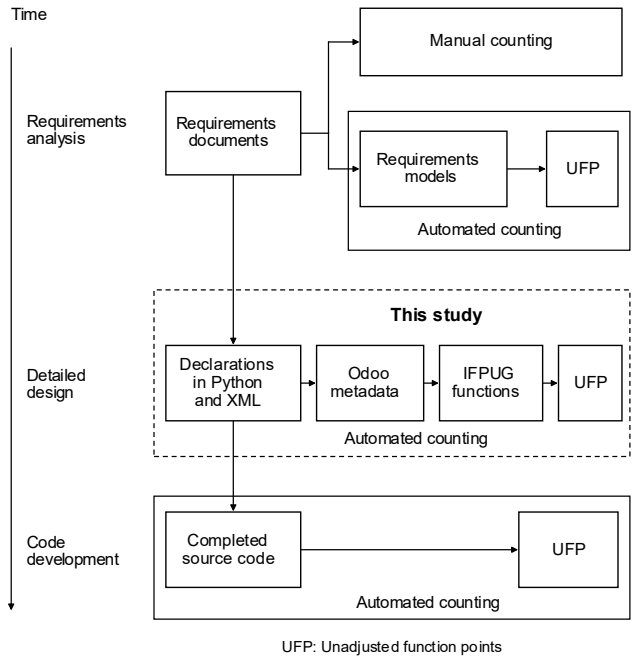


Fig. 5. FSM routes.

Fig. 5 shows the difference between this study and existing automated FSM route. Most studies conduct automated FSM based on requirement models to avoid the high cost and inconsistency of manual FSM, but the models themselves may also be costly and inconsistent, while FSM on complete codes could only be conducted after development.

In terms of accuracy, as Odoo metadata locates in the codes and specifies the subsequent software contents, the measurement based on it is naturally consistent.

In terms of timeliness, with the widespread adoption of automated software design technology like GUI, most metadata would be able to finalize in detailed design phase of software development. Indeed, metadata would finalize later than requirement models which generated in requirement analysis phase. Here a tradeoff is made to improve the degree of automation and consistency while sacrificing timeliness to some extent.

In terms of workload, in existing research manual work is needed for measurement on every single application, while in this framework the job is finished all at once for one set of metadata. In the era of AI as coding has sped up significantly, the speed of FSM has become bottleneck. As automated FSM would be a great advantage, more software projects would adopt metadata, and different software may adopt common metadata to save cost.

In terms of automation, the principal path counts for about 83.3% of UFP while complementary path counts for 16.7%. It's another tradeoff to ensure that the major measurement is conducted explicitly but avoid exponential workload increase for a small portion of functions.

2) Semantic Methods

Semantic methods in measurement is another contribution of this study. A former study [22] conducted measurements in COSMIC standard by semantic methods but the ontology only presented requirement element. The calculation is conducted by java programs. To the best of the author's knowledge, no other research performs the entire function size calculation in knowledge graphs.

By incorporating metadata items, estimation parameters and IFPUG functions into the same ontology, the knowledge graph could provide not only a UFP value but also the detail information of each function and the whole mapping logic chain. All the IFPUG functions are presented as instances and are related to the metadata instances by which they are inferred from. Diversified queries, calculation or verification on a certain function could be conducted without the learning costs of different models.

In this way, the knowledge graph would be not only a measurement instrument, but also a comprehensive information platform supporting development and maintenance of software. It's especially suitable for dealing with frequent version updates. Updating the changed instances is enough, no need for completed rework.

On the other hand, the knowledge graph is machine readable so that it could act as an integrated knowledge source for programs or AIs.

REFERENCES

[1] International Organization for Standardization, "Information technology—Software measurement—Functional size measurement—Part 1: Definition of concepts," ISO/IEC 14143-1:2007, Geneva, Switzerland, 2007.

[2] International Organization for Standardization, "Software engineering—IFPUG 4.1 unadjusted functional size measurement method—Counting practices manual," ISO/IEC 20926:2009, Geneva, Switzerland, 2009.

[3] International Function Point Users Group, *Function Point Counting Practices Manual, Release 4.3.1*. East Windsor, NJ, USA: IFPUG, 2010.

[4] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowledge Acquisition*, vol. 5, no. 2, pp. 199–220, 1993, doi: 10.1006/knac.1993.1008.

[5] A. Hogan et al., "Knowledge graphs," *ACM Comput. Surv.*, vol. 54, no. 4, pp. 1–37, 2022, doi: 10.1145/3447772.

[6] F. Erdős, B. I. Kasangaki, and L. Koloszár, "Proprietary vs. open-source ERP: The case of SAP and Odoo," in *Proc. 1st SAP UA Community Conf.: Central and Eastern Europe*, T. Orosz, Ed., Budapest, Hungary: Eötvös Loránd Univ. Faculty of Informatics, 2024, pp. 10–13.

[7] Odoo S.A., "Odoo Community Edition 19.0," 2025. [Online]. Available: https://github.com/odoo/odoo

[8] S. Klusener, "Source code based function point analysis for enhancement projects," in *Proc. Int. Conf. Softw. Maintenance (ICSM)*, 2003, pp. 373–376, doi: 10.1109/ICSM.2003.1235445.

[9] S. Kusumoto, M. Imagawa, K. Inoue, S. Morimoto, K. Matsusita, and M. Tsuda, "Function point measurement from Java programs," in *Proc. 24th Int. Conf. Softw. Eng. (ICSE)*, 2002, p. 576, doi: 10.1145/581339.581412.

[10] T. Edagawa, T. Akaike, Y. Higo, S. Kusumoto, S. Hanabusa, and T. Shibamoto, "Function point measurement from web application source code based on screen transitions and database accesses," *J. Syst. Softw.*, vol. 84, no. 6, pp. 976–984, 2011, doi: 10.1016/j.jss.2011.01.029.

[11] International Organization for Standardization, "Information technology—Object Management Group Automated Function Points (AFP), 1.0," ISO/IEC 19515:2019, Geneva, Switzerland, 2019.

[12] T. Uemura, S. Kusumoto, and K. Inoue, "Function point measurement tool for UML design specification," in *Proc. 6th Int. Softw. Metrics Symp.*, 1999, pp. 62–69, doi: 10.1109/METRIC.1999.809727.

[13] A. Živkovič, I. Rozman, and M. Heričko, "Automated software size estimation based on function points using UML models," *Inf. Softw. Technol.*, vol. 47, no. 13, pp. 881–890, 2005, doi: 10.1016/j.infsof.2005.02.008.

[14] A. Malanowska and J. Zabuski, "Afpatoo: Tool to automate function point analysis based on UML class and sequence diagrams," in *Proc. 19th Int. Conf. Eval. Novel Approaches Softw. Eng. (ENASE)*, 2024, pp. 625–632, doi: 10.5220/0012704300003687.

[15] S. Abrahão and G. Poels, "Experimental evaluation of an object-oriented function point measurement procedure," *Inf. Softw. Technol.*, vol. 49, no. 4, pp. 366–380, 2007, doi: 10.1016/j.infsof.2006.06.001.

[16] O. Pastor, R. Noel, and J. I. Panach, "Three decades of the OO-Method: Fostering conceptual-model software engineering," *Softw. Syst. Model.*, vol. 25, pp. 1289–1304, 2026, doi: 10.1007/s10270-025-01299-w.

[17] N. A. Z. Adem and Z. M. Kasirun, "Automating function points analysis based on functional and non functional requirements text," in *Proc. 2nd Int. Conf. Comput. Autom. Eng. (ICCAE)*, 2010, pp. 664–669, doi: 10.1109/ICCAE.2010.5451322.

[18] M. de Freitas, M. Fantinato, V. Sun, L. H. Thom, and V. Garaj, "Function point tree-based function point analysis: Improving reproducibility whilst maintaining accuracy in function point counting," in *Enterprise Information Systems*, Cham, Switzerland: Springer, 2020, pp. 182–209, doi: 10.1007/978-3-030-40783-4_10.

[19] C. Quesada-López, A. Martínez, M. Jenkins, L. C. Salas, and J. C. Gómez, "Automated functional size measurement: A multiple case study in the industry," in *Quality of Information and Communications Technology*, Cham, Switzerland: Springer, 2019, pp. 263–279, doi: 10.1007/978-3-030-35333-9_19.

[20] L. Lavazza, A. Locoro, G. Liu, and R. Meli, "Estimating software functional size via machine learning," *ACM Trans. Softw. Eng. Methodol.*, vol. 32, pp. 1–27, 2023, doi: 10.1145/3582575.

[21] S.-F. Wen and B. Katt, "Ontology-based metrics computation for system security assurance evaluation," *J. Appl. Secur. Res.*, vol. 19, pp. 230–275, 2024, doi: 10.1080/19361610.2022.2157190.

[22] S. Bagriyanik and A. Karahoca, "Automated COSMIC function point measurement using a requirements engineering ontology," *Inf. Softw. Technol.*, vol. 72, pp. 189–203, 2016, doi: 10.1016/j.infsof.2015.12.011.

[23] Z. Zhao, H. Jiang, R. Zhao, and B. He, "Emergence of a novel domain expert: A generative AI-based framework for software function point analysis," in *Proc. 39th IEEE/ACM Int. Conf. Autom. Softw. Eng. (ASE)*, 2024, pp. 2245–2250, doi: 10.1145/3691620.3695293.

[24] I. Nonaka and H. Takeuchi, *The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation*. New York, NY, USA: Oxford Univ. Press, 1995.

[25] B. W. Boehm et al., *Software Cost Estimation with COCOMO II*. Upper Saddle River, NJ, USA: Prentice Hall, 2000.

[26] T. Drake, Z. Chalabi, and R. Coker, "Buy now, saved later? The critical impact of time-to-pandemic uncertainty on pandemic cost-effectiveness analyses," *Health Policy Plan.*, vol. 30, no. 1, pp. 100–110, 2015, doi: 10.1093/heapol/czt101.