Applying and Evaluating Halstead's Complexity Metrics and Maintainability Index for RPG

Zoltán Tóth

Department of Software Engineering, University of Szeged, Hungary
`zizo@inf.u-szeged.hu`

Abstract. Although RPG is an older programming language for developing general-purpose software systems, it is still widely used by many companies due to the many legacy modules written in RPG that are still in use. IBM's RPG programming language has continuously evolved with the new demands. RPG has become a high-level programming language, however its original purpose was only to replicate punched card processing. Whilst RPG went through a bunch of improvements, the methodologies related to code quality assurance for RPG hardly come along. RPG is strongly applied for business applications, yet there is a lack of appropriate research studies and tools in this field. In this study, we first propose an application of Halstead's complexity metrics for RPG/400 and RPG IV. Furthermore, we investigate the usefulness and the impact of Halstead's complexity metrics in RPG programs. We examine the Halstead's complexity metrics and four Maintainability Index metrics in details to get more insight about how they correlate with other software product metrics and how could we use them to improve the quality of RPG software systems. To do so, we used Principal Component Analysis (PCA) to show the dimensionality and behavior of these metrics. We found that Halstead's complexity metrics form a strong metric group that can be used to give more details about RPG software systems.

Keywords: Software Quality, Halstead Complexity Metrics, Maintainability Index, IBM RPG

1 Introduction

ISO 25010 standard describes maintainability as one of the eight quality characteristics. Maintainability has become the most important trait related to quality as the cost of maintaining a software system gives the 40-60% of the total costs of a software[4], [21]. This is why researchers focus on maintainability and try to discover relationships between maintainability and different characteristics of the system.

IBM RPG is a programming language with a long history. It was first released in 1959 and become quite popular on IBM mainframes. There are numerous business applications having the core written in RPG. Maintainability of legacy business application are likely to be more important. More effort should be put into research studies that deal with the legacy systems' maintainability to prevent software erosion.

Halstead metrics are the first complexity measures that were defined by Maurice Halstead[8]. He taught that many characteristics of a software system can be expressed by only using the number of operands and operators occurred in a software. Halstead complexity metrics were first occurred/calculated for IBM RPG systems in 1982 presented by Hartman[9]. At that time the calculation was performed on RPG II and RPG III systems that are rare nowadays.

In this study we propose a definition of Halstead complexity metrics for newer versions of RPG, namely IBM RPG/400 and RPG IV. RPG IV brought new core language features that makes the calculation of Halstead complexity metrics absolutely different than before. Free-form block (column independent) constructions has the most impact on the methodology. We extended our static analysis tool which is called SourceMeter[1] to calculate Halstead complexity measures for RPG. We used this tool to calculate the metrics for 348 RPG programs containing 7475 subroutines. We also applied four Maintainability Index (MI) metrics that are widely used to express the overall maintainability of a software. For instance, Microsoft's Visual Studio is currently using a Maintainability Index definition to provide an overall maintainability/quality measurement for a system. Maintainability Index depends upon Halstead's Volume which motivates us to investigate the Halstead's Complexity metrics to get a deeper insight on how they are related with other software metrics. Similarly, maintainability models are constructed to gain an overall maintainability score by aggregating low-level metric values. In our previous study, we defined a quality model for RPG[12].

To determine which metrics form groups that have strong inner connections, researchers often use the concept of Principal Component Analysis (PCA). PCA is also used to reveal hidden connections from the dataset and to reduce the dimensionality of the data. Based on the Principal Component Analysis, we determined how our previous model could be extended to involve more metrics, thus ensuring a stronger descriptive behavior of our maintainability model.

The rest of this paper is structured as follows. In Section 2, we present the most important studies that are related with our research domain. Then in Section 3 we describe the background for this study that includes the RPG programming language itself and the software metrics and the maintainability model. In Section 4, we present the definitions of the used metrics. Section 5 shows the results of the Principal Component Analysis and we make suggestions for extending the quality model. Finally, we end the paper with summarizing and concluding the results, enumerating the future work possibilities and the threats to validity.

2 Related Work

In this section we present the most important studies that relate to static source code analysis in RPG systems and software metrics defined for RPG language.

[1] `https://www.sourcemeter.com`

The literature is lack of studies that focus on RPG legacy software systems hence we can only enumerate a very limited number of papers that have RPG related research topics. We can hardly identify groups of research areas when RPG is in the spotlight. The first studies that are related to IBM RPG are from 1982. Naib investigates internal (not varying with time - McCabe, Halstead, Lines of Code) and external (varying with time - number of users) metrics on two large RPG packages to see whether the metrics have correlation with error rates.[18]. Different internal measures are calculated at module level for which Naib used Hartman's counting tool to support the identification of fault-prone RPG II and RPG III modules[9]. Hartman used the original definitions to calculate McCabe's Cyclomatic Complexity[14] and the Halstead's complexity metrics[8].

The usefulness of metrics are mostly accepted, however sometimes the metrics are criticized rather to pinpoint the weaknesses and add a gentle indication to change or modify the directions of the research areas[23]. These kind of studies often reflect the misuse of metrics in different models. Halstead's complexity metric family as being one of the first complexity metric set is sometimes handled as a golden hammer[5] that is obviously a bad practice. Consequently, more metrics were defined and used for empirical analysis to show different characteristics of the subject systems [16], [2]. For evaluating new complexity metrics, sometimes different frameworks are used [15]. Maintainability Index (MI) was first introduced by Oman et al. in 1994 [19], [7]. MI was designed to express the maintainability of a system (as its name reflects) with a single value. Its power has become its weakness since it does not provide any information on how the metric value was made up (maybe only one lower level metric is critical) or what changes should be made to improve the system's maintainability[10]. As ISO 25010 describes, maintainability is a derived quality indicator which is comprising modularity, reusability, analyzability, modifiability, testability. However, Maintainability Index is an ideal measurement when one would like to compare the overall maintainability of different software systems. Maintainability models were proposed to overcome the above mentioned problems [17], [11] and soon more complex quality models were given birth,[20], [22], [1].

Bakota et al. presented a probabilistic software quality model where the overall maintainability is derived from analyzability, changeability, stability, testability[1]. They used the ISO 9126 standard which is the ancestor of ISO 25010, thus this model has become quite out-dated and needs to be updated, however it is still usable. In case of RPG we have proposed a similar quality model in our paper[12] which is based on the results of the probabilistic software quality model. In this study we would like to give recommendations for extending the RPG quality model to involve more measurements that reflect the overall quality of a system in a more precise way.

3 Background

3.1 RPG Programming Language

RPG is a high level programming language developed by IBM (first released in 1959) and used in IBM mainframe environment. RPG is still a popular programming language on the IBM i OS. RPG has been continuously developed to fulfill the new demands and capabilities presented in other domain-free languages. These improvements result in multiple versions of the programming language.

Listing 1.1. A simple RPG IV program

```
..... *. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+...
     D Add              pr            15s 2
     D num1                           15s 2
     D num2                           15s 2
     p Add              b                     export
     d Add              PI            15s 2
     d num1                           15s 2
     d num2                           15s 2
     d result           s              6s 0
      /free
             result = num1 + num2;
             *inlr = *on;
             return result;
      /end-free
     p Add              e
```

Two commonly used versions are RPG/400 (also known as RPG III) and ILE RPG (also known as RPG IV). A simple RPG IV program is shown in Listing 1.1. RPG/400 uses a strict, column-based format that is inherited in RPG IV, but the latter has free-form blocks that makes possible a column-independent programming style which opens a different world for RPG programmers. The sample program consists of a simple procedure declaration (Add) and its definition which returns the sum of two numbers.

RPG has different specifications that are noted with a specified letter in the sixth column. For our investigations we will mainly focus on the Calculation Specifications that indicate the operations to be done on the data.

3.2 Software Metrics and Quality Model

”You can’t manage what you can’t measure.” is an old adage by Peter Drucker that is still accurate. This is why different software metrics became so important in the last decades[6].

We used a tool named SourceMeter which is our own development for static source code analysis. We used this tool to calculate the appropriate software product metrics, thus we can investigate the correlation between the original metrics like LLOC, McCC and the newly added ones (Halstead and Maintainability Index metrics).

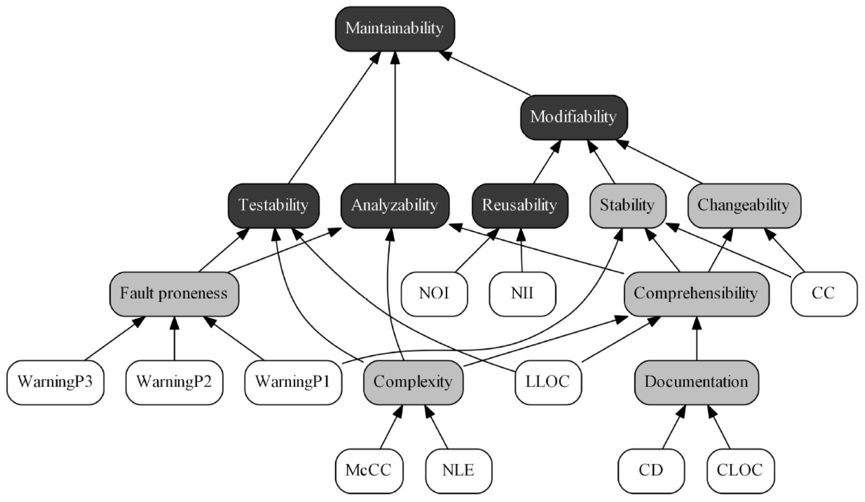

Fig. 1. Quality Model for RPG

The quality model makes use of the software product metrics as shown in Figure 1. Maintainability as the root node is calculated from testability, analyzability, and modifiability. These metrics are also aggregated from other metrics as depicted. Leaf node metrics are the sensor metrics (which we can directly measure). Every aggregated node is (constructed from sensor metrics - inner nodes) calculated from multiple lower level metrics with different weights (in other words the presented graph is weighted). The quality model is based on the 9126 ISO standard that has become quite obsolete and should be updated.

Table 1 presents the definitions of sensor metrics used in the quality model for RPG. We will later investigate the relationship of these metrics and the Halstead and MI metrics. One can find a more detailed description about the listed metrics on the User's Guide page[2].

4 Computing Halstead Metrics and Maintainability Index for RPG

Halstead Complexity metrics[8] are likely to be forgotten that is undeserving in many cases. For instance, Maintainablity Index[7] shows the strength of Halstead's metrics. Coleman et al. used Halstead's Effort metric amongst others to

[2] `https://www.sourcemeter.com/resources/rpg/`

Table 1. Definition of source code metrics used in the quality model

Metric name	**Abbreviation**	**Description**
Logical Lines Of Code	LLOC	Number of non-empty and non-comment code lines of the subroutine/procedure/program.
Number of Incoming Invocations	NII	Measures the number a subroutine/procedure/program has been called by other subroutines/procedures/programs.
Number of Outgoing Invocations	NOI	Measures the number of subroutines/procedures/programs the subroutine/procedure/program has called.
Clone Coverage	CC	Ratio of code covered by code duplications in the source code element to the size of the source code element.
Comment Density	CD	Ratio of the comment lines of the subroutine/procedure/program (CLOC) to the sum of its comment (CLOC) and logical lines of code (LLOC).
Comment Lines Of Code	CLOC	Number of comment and documentation code lines of the subroutine/procedure/program.
McCabe's Cyclomatic Compexity	McCC	Complexity of the subroutine/procedure/program expressed as the number of independent control flow paths in it.
Nesting Level Else-If	NLE	Complexity of the subroutine/procedure/program expressed as the depth of the maximum embeddedness of its conditional, iteration and exception handling block scopes.
Warning Occurrences	Warnings (P1,P2,P3)	Number of detected coding rule violations with a given severity (P1 - lowest severity, P3 - highest severity)

derive the original Maintainability Index (MI) metric. At that time Halstead's volume and effort metrics were considered as the best indicators for predicting the maintainability of a software system.

To produce the necessary metric values, we first present the list of definitions for Halstead metrics. Let us consider the following notations:

- η_1 = number of distinct operators
- η_2 = number of distinct operands
- N_1 = total number of operators
- N_2 = total number of operands

Now we have the definition of the four basic metrics we will use in our further formulas, however there is no intention or concept what should be considered as an operand and an operation. This problem can cause inconsistencies between research papers since they use different interpretations. Furthermore, the calculation of operands and operators can intensely differ by programming languages (mainly comes from the dissimilarities of the languages). Fortunately, in case of RPG we do not have to dig deep to figure out how different source code elements should be treated. We calculated the Halstead's complexity metrics similarly as it was presented by Hartman for RPG III, thus we concentrate on the peculiarities of RPG IV. Now we will present the different source code elements that should be included in the calculations.

Table 2. List of source code elements to be counted as operators

Specification name	Construct name	RPG version
Calculation	Operator	RPG/400 RPG IV
Free-form (C)	Infix expressions	RPG IV
Free-form (C)	Member Selection	RPG IV
Free-form (C)	Array Subscript	RPG IV
Free-form (C)	Parentheses	RPG IV
Free-form (C)	Prefix Expressions	RPG IV

Table 2 summarizes the source code elements in different RPG versions to be counted as operators. Calculation specification is the place where we can specify the operations to be done on the given operands. In RPG IV we use free-form to avoid column-sensitive programming. In free-form section we can use different operators such as infix operators (+,-,*,/, $<$,$>$, ...), member selection (data structure field select), array subscription (to get elements from an array), parentheses (to modify the operation precedence), and also prefix operations. Most of the free-form statements can be written in calculation specifications, some cannot.

Table 3. List of source code elements to be counted as operands

Specification name	Construct name	RPG version
Calculation	Factor 1	RPG/400 RPG IV
Calculation	Factor 2	RPG/400 RPG IV
Calculation	Result Field	RPG/400 RPG IV
Definition	(Variable) Name	RPG IV
Input	Program Field	RPG/400, RPG IV
Input	Data Structure	RPG/400, RPG IV
Input	Data Structure Subfield	RPG/400, RPG IV
Input	External Record	RPG/400, RPG IV
Input	External Field	RPG/400, RPG IV
Input	Data Structure	RPG/400, RPG IV
Input	Data Structure	RPG/400, RPG IV
Input	Named Constant	RPG/400
Free-form	Literal	RPG IV
Free-form	Identifier	RPG IV
Output	Output External Record	RPG/400, RPG IV
Output	Output External Field	RPG/400, RPG IV
Output	Output Program Field	RPG/400, RPG IV

Table 3 shows the RPG constructions to be counted as operands. When we use an operator in Calculation Specification we have to specify operand(s) (if needed) to perform the operation on. These operands should be specified in factor 1 and factor 2. The result of the operation is stored in the given result field. In RPG IV we can use Definition Specification to define variables and constants. We use Input and Output Specification to declare the appropriate input and output data structures and their fields (also constants in RPG/400). In RPG IV we can also use literals and identifiers in free-form section which are also counted as operands.

Table 4 introduces the Halstead metrics that are aggregated from the basic ones (η_1, η_2, N_1, N_2). Table 5 presents the different variants of Maintainability Index (MI) metrics.

Table 4. List of the used Halstead metrics

Metric Name	**Formula**
Program Vocabulary (HPV)	$\eta = \eta_1 + \eta_2$
Program Length (HPL)	$N = N_1 + N_2$
Calculated Program Length (HCPL)	$\hat{N} = \eta_1 \cdot log_2\, \eta_1 + \eta_2 \cdot log_2\, \eta_2$
Volume (HVOL)	$V = N \times log_2\, \eta$
Difficulty (HDIF)	$D = \frac{\eta_1}{2} \times \frac{N_2}{\eta_2}$
Effort (HEFF)	$E = D \times V$
Time required to program (HTRP)	$T = \frac{E}{18}$
Number of delivered bugs (HNDB)	$B = \frac{E^{\frac{2}{3}}}{3000}$

Table 5. List of the used Maintainablity Index metrics

MI variant	**Formula**
Original (MI)	$171 - 5.2 \times ln(HVOL) - 0.23 \times McCC - 16.2 \times ln(LLOC)$
SEI (MISEI)	$171 - 5.2 \times log_2(HVOL) - 0.23 \times McCC - 16.2 \times log_2(LLOC) + 50 \times sin(\sqrt{2.4 * CD})$
Visual Studio (MIMS)	$max(0, 100 \times \frac{171-5.2\times ln(HVOL)-0.23\times McCC-16.2\times ln(LLOC)}{171})$
SourceMeter (MISM)	$max(0, 100 \times \frac{171-5.2\times log_2(HVOL)-0.23\times McCC-16.2\times log_2(LLOC)+50\times sin(\sqrt{2.4*CD})}{171})$

In RPG, we have 3 levels of abstraction, namely subroutine, procedure, and program. We can define a subroutine by writing code between BEGSR and ENDSR operation codes. To call a subroutine we have to use the EXSR operation and specify the name of the subroutine to be called. Unlike subroutines, procedures can be prototyped and have parameters, thus supporting a more flexible way to reuse code portions. Programs are larger building blocks that encapsulate subroutines and procedures as well. In this study, we will only examine subroutines and programs because we accessed a limited set of source code files that mainly contains subroutines instead of procedures.

5 Evaluating the usefulness of Halstead's and MI metrics

Principal Component Analysis (PCA)[24] is widely used in many domains to accomplish dimensionality reduction and uncover patterns from the data.[3], [13]. PCA determines which dimensions are the most important ones and which ones represent the most variation in the data. PCA takes a dataset (a set of metrics in our case) as input and outputs principal components (uncorrelated dimensions) that span the direction of the $1^{st}, 2^{nd}, 3^{rd}, \ldots$ largest variations.

We have performed PCA both at program (RPG file) and subroutine level to see the difference between these levels if any exists. We investigated 348 RPG programs (185 RPG IV and 163 RPG/400 programs) and 7475 RPG subroutines with PCA.

We first present the correlation matrices that can be seen in Table 6 and Table 7. We included the Halstead, Maintainability Index metrics and the sensor metrics that are used by the quality model in the correlation matrix to investigate the relationship between them. Values in the table are mapped with color codes to help better understand the correlations between metrics. The color interpolation has three base points: -1, 0, 1. The greater the correlation between two metrics (negative or positive correlation) the greener the cell is (1 and -1 values imply pure green color). Red means that two variables are not correlated. One can see clear groups of metrics that correlation coefficients are very high inside the group. In case of programs Halstead metrics form such a group that is not surprising since many of them are calculated with the help of another (See Table 4). Maintainability Index metrics has the same characteristics. Their lowest correlation is 0.946 (program level) and 0.997 (subroutine level) that is a very high value. High correlation is caused by the fact that each variant has almost the same core in their formula. A relatively high correlation can be seen in case of the different warnings (avg. correlation: 0.774) at program level but the same cannot be told for subroutines. Warnings are different bad smells that should be reviewed because they can reveal the weak spots of the system.

It is promising that the correlations between Halstead metrics and warnings are high (avg. correlation is 0.812) since we can use the Halstead metrics to predict warnings in the system (at program level). Unfortunately, no valuable correlation found at subroutine level between these metrics. The Halstead complexity metrics are also highly correlated with the McCC metric (we use the Program Complexity (PC) terminology at program level) which means that each complexity measure can express the other. This is partly true at subroutine level since HCPL, HPL, HPV and HVOL have poor correlations with McCC. At program level, the McCabe's Complexity metric also can be used to express the

Table 6. Correlation between metrics (Program Level)

Variables	**CC**	**HCPL**	**HDIF**	**HEFF**	**HNDB**	**HPL**	**HPV**	**HTRP**	**HVOL**	**MI**	**MIMS**	**MISEI**	**MISM**	**NLE**	**PC**	**NOI**	**CD**	**CLOC**	**TCD**	**LLOC**	**Warning Info**	**Clone Metric Rules**	**Complexity Metric Rules**	**Coupling Metric Rules**	**Doc. Metric Rules**	**Size Metric Rules**
CC	1	0,073	0,162	0,033	0,067	0,066	0,087	0,033	0,055	-0,186	-0,186	-0,156	-0,149	0,099	0,061	0,074	0,083	0,207	0,009	0,145	0,435	0,485	0,058	0,020	0,007	0,079
HCPL	0,073	1	0,811	0,891	0,947	0,970	0,996	0,891	0,965	-0,698	-0,698	-0,662	-0,667	0,521	0,951	0,215	0,076	0,461	-0,417	0,513	0,785	0,712	0,889	0,834	0,789	0,933
HDIF	0,162	0,811	1	0,736	0,865	0,841	0,835	0,736	0,812	-0,673	-0,673	-0,610	-0,627	0,645	0,836	0,346	0,162	0,463	-0,400	0,253	0,771	0,720	0,774	0,669	0,619	0,829
HEFF	0,033	0,891	0,736	1	0,967	0,953	0,858	1,000	0,966	-0,464	-0,464	-0,438	-0,447	0,370	0,949	0,082	0,057	0,502	-0,255	0,334	0,745	0,675	0,871	0,825	0,815	0,847
HNDB	0,067	0,947	0,865	0,967	1	0,992	0,933	0,967	0,990	-0,578	-0,578	-0,539	-0,551	0,481	0,984	0,172	0,091	0,510	-0,330	0,346	0,813	0,744	0,910	0,850	0,813	0,911
HPL	0,066	0,970	0,841	0,953	0,992	1	0,956	0,953	0,998	-0,606	-0,606	-0,567	-0,578	0,483	0,987	0,189	0,090	0,515	-0,350	0,379	0,818	0,746	0,918	0,866	0,821	0,928
HPV	0,087	0,996	0,835	0,858	0,933	0,956	1	0,858	0,946	-0,740	-0,740	-0,700	-0,706	0,554	0,935	0,256	0,088	0,451	-0,442	0,521	0,786	0,715	0,871	0,813	0,764	0,937
HTRP	0,033	0,891	0,736	1,000	0,967	0,953	0,858	1	0,966	-0,464	-0,464	-0,438	-0,447	0,370	0,949	0,082	0,057	0,502	-0,255	0,334	0,745	0,675	0,871	0,825	0,815	0,847
HVOL	0,055	0,965	0,812	0,966	0,990	0,998	0,946	0,966	1	-0,579	-0,579	-0,543	-0,554	0,459	0,986	0,162	0,081	0,514	-0,332	0,380	0,806	0,733	0,919	0,869	0,827	0,918
MI	-0,186	-0,698	-0,673	-0,464	-0,578	-0,606	-0,740	-0,464	-0,579	1	1,000	0,946	0,950	-0,631	-0,572	-0,482	-0,121	-0,360	0,654	-0,680	-0,540	-0,492	-0,546	-0,432	-0,486	-0,716
MIMS	-0,186	-0,698	-0,673	-0,464	-0,578	-0,606	-0,740	-0,464	-0,579	1,000	1	0,946	0,950	-0,631	-0,572	-0,482	-0,121	-0,360	0,654	-0,680	-0,540	-0,492	-0,546	-0,432	-0,486	-0,716
MISEI	-0,156	-0,662	-0,610	-0,438	-0,539	-0,567	-0,700	-0,438	-0,543	0,946	0,946	1	0,995	-0,578	-0,535	-0,421	0,209	-0,254	0,743	-0,701	-0,496	-0,449	-0,511	-0,406	-0,512	-0,670
MISM	-0,149	-0,667	-0,627	-0,447	-0,551	-0,578	-0,706	-0,447	-0,554	0,950	0,950	0,995	1	-0,593	-0,546	-0,438	0,180	-0,269	0,743	-0,665	-0,507	-0,459	-0,520	-0,415	-0,510	-0,682
NLE	0,099	0,521	0,645	0,370	0,481	0,483	0,554	0,370	0,459	-0,631	-0,631	-0,578	-0,593	1	0,472	0,380	0,133	0,276	-0,461	0,318	0,443	0,398	0,570	0,366	0,337	0,554
PC	0,061	0,951	0,836	0,949	0,984	0,987	0,935	0,949	0,986	-0,572	-0,572	-0,535	-0,546	0,472	1	0,164	0,088	0,479	-0,319	0,352	0,796	0,724	0,915	0,865	0,807	0,911
NOI	0,074	0,215	0,346	0,082	0,172	0,189	0,256	0,082	0,162	-0,482	-0,482	-0,421	-0,438	0,380	0,164	1	0,165	0,282	-0,265	0,135	0,163	0,142	0,165	0,068	0,112	0,314
CD	0,083	0,076	0,162	0,057	0,091	0,090	0,088	0,057	0,081	-0,121	-0,121	0,209	0,180	0,133	0,088	0,165	1	0,305	0,301	-0,094	0,110	0,111	0,084	0,059	-0,102	0,109
CLOC	0,207	0,461	0,463	0,502	0,510	0,515	0,451	0,502	0,514	-0,360	-0,360	-0,254	-0,269	0,276	0,479	0,282	0,305	1	-0,052	0,171	0,584	0,574	0,432	0,428	0,384	0,496
TCD	0,009	-0,417	-0,400	-0,255	-0,330	-0,350	-0,442	-0,255	-0,332	0,654	0,654	0,743	0,743	-0,461	-0,319	-0,265	0,301	-0,052	1	-0,511	-0,228	-0,178	-0,345	-0,222	-0,455	-0,460
LLOC	0,145	0,513	0,253	0,334	0,346	0,379	0,521	0,334	0,380	-0,680	-0,680	-0,701	-0,665	0,318	0,352	0,135	-0,094	0,171	-0,511	1	0,272	0,225	0,385	0,251	0,467	0,449
WarningInfo	0,435	0,785	0,771	0,745	0,813	0,818	0,786	0,745	0,806	-0,540	-0,540	-0,496	-0,507	0,443	0,796	0,163	0,110	0,584	-0,228	0,272	1	0,992	0,737	0,715	0,599	0,748
Clone Metric Rules	0,485	0,712	0,720	0,675	0,744	0,746	0,715	0,675	0,733	-0,492	-0,492	-0,449	-0,459	0,398	0,724	0,142	0,111	0,574	-0,178	0,225	0,992	1	0,654	0,657	0,516	0,668
Complexity Metric Rules	0,058	0,889	0,774	0,871	0,910	0,918	0,871	0,871	0,919	-0,546	-0,546	-0,511	-0,520	0,570	0,915	0,165	0,084	0,432	-0,345	0,385	0,737	0,654	1	0,788	0,801	0,857
Coupling Metric Rules	0,020	0,834	0,669	0,825	0,850	0,866	0,813	0,825	0,869	-0,432	-0,432	-0,406	-0,415	0,366	0,865	0,068	0,059	0,428	-0,222	0,251	0,715	0,657	0,788	1	0,657	0,755
Documentation Metric Rules	0,007	0,789	0,619	0,815	0,813	0,821	0,764	0,815	0,827	-0,486	-0,486	-0,512	-0,510	0,337	0,807	0,112	-0,102	0,384	-0,455	0,467	0,599	0,516	0,801	0,657	1	0,780
Size Metric Rules	0,079	0,933	0,829	0,847	0,911	0,928	0,937	0,847	0,918	-0,716	-0,716	-0,670	-0,682	0,554	0,911	0,314	0,109	0,496	-0,460	0,449	0,748	0,668	0,857	0,755	0,780	1

Table 7. Correlation between metrics (Subroutine Level)

Variables	**CC**	**HCPL**	**HDIF**	**HEFF**	**HNDB**	**HPL**	**HPV**	**HTRP**	**HVOL**	**MI**	**MIMS**	**MISEI**	**MISM**	**McCC**	**NLE**	**NII**	**NOI**	**CD**	**CLOC**	**LLOC**	**Warning Info**	**Clone Metric Rules**	**Complexity Metric Rules**	**Coupling Metric Rules**	**Doc. Metric Rules**	**Size Metric Rules**
CC	**1**	-0,089	-0,047	-0,016	-0,045	-0,064	-0,102	-0,016	-0,058	0,100	0,100	0,105	0,105	-0,026	-0,135	0,015	0,027	0,142	-0,026	-0,063	0,825	0,902	-0,023	0,026	-0,021	-0,050
HCPL	-0,089	**1**	0,358	0,607	0,728	0,954	0,982	0,607	0,952	-0,684	-0,684	-0,680	-0,666	0,306	0,188	-0,092	0,132	-0,509	0,859	0,952	0,265	0,101	0,268	0,076	0,301	0,671
HDIF	-0,047	0,358	**1**	0,636	0,798	0,438	0,455	0,636	0,353	-0,689	-0,688	-0,691	-0,690	0,743	0,650	0,031	0,167	-0,633	0,316	0,451	0,280	0,123	0,622	0,050	0,384	0,358
HEFF	-0,016	0,607	0,636	**1**	0,922	0,740	0,601	1,000	0,710	-0,492	-0,487	-0,481	-0,462	0,701	0,228	-0,018	0,104	-0,332	0,600	0,734	0,281	0,116	0,388	0,080	0,347	0,580
HNDB	-0,045	0,728	0,798	0,922	**1**	0,827	0,763	0,922	0,768	-0,722	-0,720	-0,716	-0,704	0,750	0,387	-0,031	0,158	-0,553	0,680	0,828	0,352	0,145	0,537	0,087	0,444	0,690
HPL	-0,064	0,954	0,438	0,740	0,827	**1**	0,925	0,740	0,989	-0,669	-0,668	-0,662	-0,645	0,408	0,162	-0,099	0,098	-0,475	0,888	0,995	0,298	0,121	0,301	0,067	0,331	0,722
HPV	-0,102	0,982	0,455	0,601	0,763	0,925	**1**	0,601	0,901	-0,798	-0,798	-0,795	-0,784	0,387	0,289	-0,088	0,177	-0,630	0,831	0,928	0,290	0,115	0,336	0,094	0,331	0,679
HTRP	-0,016	0,607	0,636	1,000	0,922	0,740	0,601	**1**	0,710	-0,492	-0,487	-0,481	-0,462	0,701	0,228	-0,018	0,104	-0,332	0,600	0,734	0,281	0,116	0,388	0,080	0,347	0,580
HVOL	-0,058	0,952	0,353	0,710	0,768	0,989	0,901	0,710	**1**	-0,580	-0,579	-0,573	-0,554	0,330	0,111	-0,094	0,074	-0,393	0,883	0,983	0,263	0,105	0,240	0,054	0,288	0,670
MI	0,100	-0,684	-0,689	-0,492	-0,722	-0,669	-0,798	-0,492	-0,580	**1**	1,000	0,998	0,997	-0,584	-0,543	0,061	-0,292	0,893	-0,591	-0,687	-0,318	-0,151	-0,472	-0,144	-0,347	-0,528
MIMS	0,100	-0,684	-0,688	-0,487	-0,720	-0,668	-0,798	-0,487	-0,579	1,000	**1**	0,998	0,998	-0,580	-0,543	0,061	-0,292	0,894	-0,591	-0,686	-0,317	-0,151	-0,471	-0,143	-0,347	-0,528
MISEI	0,105	-0,680	-0,691	-0,481	-0,716	-0,662	-0,795	-0,481	-0,573	0,998	0,998	**1**	1,000	-0,574	-0,549	0,058	-0,271	0,914	-0,566	-0,680	-0,309	-0,143	-0,469	-0,130	-0,368	-0,519
MISM	0,105	-0,666	-0,690	-0,462	-0,704	-0,645	-0,784	-0,462	-0,554	0,997	0,998	1,000	**1**	-0,568	-0,552	0,058	-0,273	0,916	-0,551	-0,663	-0,308	-0,143	-0,469	-0,129	-0,367	-0,515
McCC	-0,026	0,306	0,743	0,701	0,750	0,408	0,387	0,701	0,330	-0,584	-0,580	-0,574	-0,568	**1**	0,531	0,066	0,333	-0,474	0,307	0,428	0,296	0,120	0,663	0,232	0,383	0,393
NLE	-0,135	0,188	0,650	0,228	0,387	0,162	0,289	0,228	0,111	-0,543	-0,543	-0,549	-0,552	0,531	**1**	0,062	0,261	-0,587	0,101	0,198	0,092	-0,007	0,591	0,082	0,185	0,050
NII	0,015	-0,092	0,031	-0,018	-0,031	-0,099	-0,088	-0,018	-0,094	0,061	0,061	0,058	0,058	0,066	0,062	**1**	0,091	0,019	-0,119	-0,092	-0,014	-0,008	0,068	0,053	0,016	-0,102
NOI	0,027	0,132	0,167	0,104	0,158	0,098	0,177	0,104	0,074	-0,292	-0,292	-0,271	-0,273	0,333	0,261	0,091	**1**	-0,124	0,248	0,127	0,200	0,137	0,266	0,655	0,035	0,073
CD	0,142	-0,509	-0,633	-0,332	-0,553	-0,475	-0,630	-0,332	-0,393	0,893	0,894	0,914	0,916	-0,474	-0,587	0,019	-0,124	**1**	-0,268	-0,493	-0,187	-0,062	-0,385	-0,036	-0,371	-0,327
CLOC	-0,026	0,859	0,316	0,600	0,680	0,888	0,831	0,600	0,883	-0,591	-0,591	-0,566	-0,551	0,307	0,101	-0,119	0,248	-0,268	**1**	0,890	0,303	0,158	0,237	0,169	0,109	0,652
LLOC	-0,063	0,952	0,451	0,734	0,828	0,995	0,928	0,734	0,983	-0,687	-0,686	-0,680	-0,663	0,428	0,198	-0,092	0,127	-0,493	0,890	**1**	0,305	0,125	0,324	0,083	0,340	0,722
WarningInfo	0,825	0,265	0,280	0,281	0,352	0,298	0,290	0,281	0,263	-0,318	-0,317	-0,309	-0,308	0,296	0,092	-0,014	0,200	-0,187	0,303	0,305	**1**	0,961	0,288	0,162	0,201	0,311
Clone Metric Rules	0,902	0,101	0,123	0,116	0,145	0,121	0,115	0,116	0,105	-0,151	-0,151	-0,143	-0,143	0,120	-0,007	-0,008	0,137	-0,062	0,158	0,125	0,961	**1**	0,093	0,093	0,045	0,087
Complexity Metric Rules	-0,023	0,268	0,622	0,388	0,537	0,301	0,336	0,388	0,240	-0,472	-0,471	-0,469	-0,469	0,663	0,591	0,068	0,266	-0,385	0,237	0,324	0,288	0,093	**1**	0,208	0,325	0,275
Coupling Metric Rules	0,026	0,076	0,050	0,080	0,087	0,067	0,094	0,080	0,054	-0,144	-0,143	-0,130	-0,129	0,232	0,082	0,053	0,655	-0,036	0,169	0,083	0,162	0,093	0,208	**1**	0,000	0,066
Documentation Metric Rules	-0,021	0,301	0,384	0,347	0,444	0,331	0,331	0,347	0,288	-0,347	-0,347	-0,368	-0,367	0,383	0,185	0,016	0,035	-0,371	0,109	0,340	0,201	0,045	0,325	0,000	**1**	0,340
Size Metric Rules	-0,050	0,671	0,358	0,580	0,690	0,722	0,679	0,580	0,670	-0,528	-0,528	-0,519	-0,515	0,393	0,050	-0,102	0,073	-0,327	0,652	0,722	0,311	0,087	0,275	0,066	0,340	**1**

warnings in the system since it has 0.836 avg. correlation coefficient with the warning metrics.

PCA constructs 25 dimensions (factors) from 26 dimensions that is not the best case scenario. However, using the first ten factors will give back 96.865 (program level) and 96.366 (subroutine level) percent of the total variability. Figure 2 and Figure 3 depict the eigenvalues for all the 25 factors and the cumulative variability at program level and subroutine level respectively. The cumulative variability is slightly steeper in case of programs meaning that we can reconstruct the original data by using less dimensions (factors).

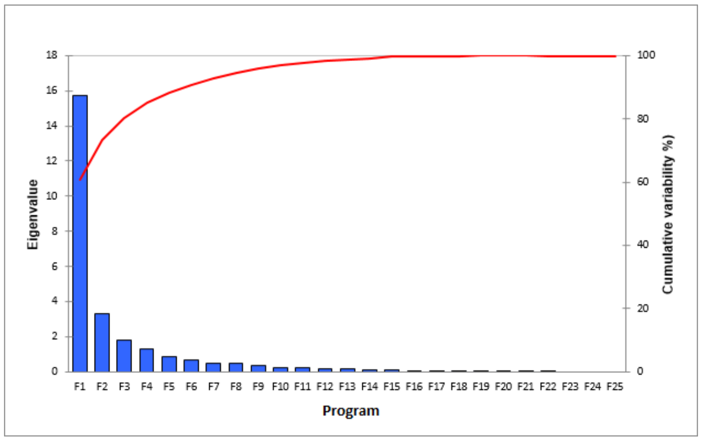

Fig. 2. Eigenvalues and variability of principal components (Program level)

Factors are constructed from the original metrics with linear combination. It is important to examine the so called factor loadings which gives us the linear combinations for each factor. We analyzed the factor loadings only for the first five factors since they retrieve 88.204 and 83.814 percent of the whole variability at program and subroutine level respectively, thus analyzing the most dominant factors is enough to detect the most dominant original metrics. Table 8 shows the factor loadings for the first five factors both at program and subroutine levels. Values higher than 0.7 are highlighted. It is clearly visible that the first factors are made up from many metrics to caption the maximum possible variability. Both in case of program and subroutine levels the Halstead metrics are the most prominent ones that contribute with the largest weights meaning that they are the most descriptive metrics. Maintainability Index variants are combined with negative weights but they are also significant ones. Further dominant metrics are different at program and subroutine level. The McCabe Cyclomatic Complexity is as strong as the warning occurrence metrics at program level. At subroutine

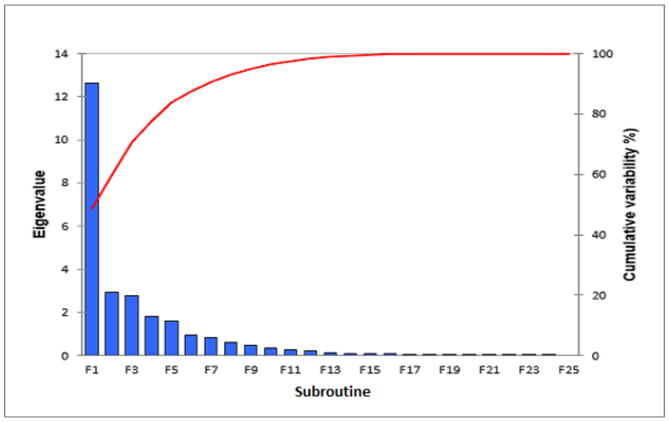

Fig. 3. Eigenvalues and variability of principal components (Subroutine level)

level, the CD, CLOC and LLOC metrics are stronger besides the Halstead and MI metrics which are absolutely dominating.

5.1 Extend Quality Model For RPG

Consider the maintainability model presented in Figure 1. We can enhance the expressiveness of the model by involving further metrics based on the results of the Principal Component Analysis. PCA showed that the Halstead complexity metrics form an independent group that captures the most information of the system (has the largest weights in factor loadings). Considering the correlation matrix we suggest to involve HNDB metric into the model to contribute to the calculation of fault proneness since it has the largest correlation coefficients with the warning occurrences. Furthermore, we suggest to include the HPV metric to contribute to the Complexity aggregated node since it has low correlation with the McCabe's cyclomatic complexity in case of subroutines but it has a large weight in the linear combination in factor loading (dominant metric) thus McCabe's complexity, NLE and HPV forms a unit together to describe the overall Complexity.

6 Conclusion

We have defined the Halstead Complexity metrics for RPG/400 and RPG IV programming languages that has never done before. Furthermore, we have a prototype implementation for these defined metrics. We also work out four different Maintainability Index variants in our static source code analyzer. We performed a Principal Component Analysis on 348 RPG programs and on 7475 subroutines and we investigated the relationships between the calculated metrics. We experienced that the Halstead's Complexity metrics form a disjoint group that can

Table 8. Factor loadings

	Program					**Subroutine**				
	F1	**F2**	**F3**	**F4**	**F5**	**F1**	**F2**	**F3**	**F4**	**F5**
CC	0,153	-0,078	0,616	0,663	0,017	-0,047	-0,081	**0,947**	-0,081	-0,167
HCPL	**0,969**	0,082	-0,090	-0,028	0,072	**0,853**	-0,397	-0,100	-0,193	0,069
HDIF	**0,874**	0,020	0,166	-0,109	-0,249	**0,725**	0,442	0,014	0,291	-0,199
HEFF	**0,891**	0,358	-0,151	-0,009	0,049	**0,774**	-0,198	0,035	0,530	-0,044
HNDB	**0,955**	0,260	-0,070	-0,039	-0,035	**0,928**	-0,052	0,010	0,328	-0,079
HPL	**0,966**	0,230	-0,074	-0,038	-0,007	**0,888**	-0,426	-0,064	-0,026	0,030
HPV	**0,971**	0,022	-0,055	-0,040	0,051	**0,906**	-0,245	-0,102	-0,237	0,046
HTRP	**0,891**	0,358	-0,151	-0,009	0,049	**0,774**	-0,198	0,035	0,530	-0,044
HVOL	**0,956**	0,259	-0,099	-0,030	0,016	**0,827**	-0,508	-0,071	-0,021	0,053
MI	**-0,771**	0,580	-0,136	0,068	-0,127	**-0,892**	-0,285	0,065	0,305	0,032
MIMS	**-0,771**	0,580	-0,136	0,068	-0,127	**-0,891**	-0,286	0,066	0,309	0,032
MISEI	**-0,736**	0,643	0,071	-0,102	0,010	**-0,887**	-0,297	0,076	0,312	0,058
MISM	**-0,745**	0,631	0,049	-0,072	0,034	**-0,877**	-0,313	0,076	0,323	0,061
NLE	0,602	-0,326	0,186	-0,240	-0,288	0,463	0,664	-0,097	0,035	-0,049
McCC	**0,947**	0,260	-0,090	-0,042	-0,020	0,678	0,393	0,092	0,478	0,044
NOI	0,297	-0,430	0,377	-0,422	-0,281	0,258	0,315	0,203	-0,036	**0,782**
CD	0,072	0,215	0,627	-0,516	0,412	**-0,724**	-0,439	0,155	0,348	0,215
CLOC	0,537	0,185	0,455	-0,065	0,150	**0,771**	-0,455	0,008	-0,108	0,247
NII	-	-	-	-	-	-0,059	0,182	0,047	0,208	0,123
LLOC	0,516	-0,502	-0,180	0,185	0,549	**0,899**	-0,393	-0,059	-0,033	0,046
Warning Info	**0,837**	0,206	0,314	0,306	-0,110	0,397	-0,012	**0,897**	-0,088	-0,095
Clone Metric Rules	**0,768**	0,209	0,379	0,362	-0,125	0,188	-0,029	**0,948**	-0,162	-0,129
Complexity Metric Rules	**0,899**	0,202	-0,101	-0,068	-0,033	0,538	0,453	0,089	0,311	0,055
Coupling Metric Rules	**0,813**	0,332	-0,106	-0,008	-0,028	0,156	0,191	0,201	0,028	**0,829**
Documentation Metric Rules	**0,808**	0,126	-0,312	0,053	0,081	0,439	0,125	0,002	0,199	-0,244
Size Metric Rules	**0,947**	0,021	-0,030	-0,105	0,001	**0,705**	-0,330	-0,011	0,056	0,001

be used to characterize the warning occurrences in the system at program level. Moreover, Halstead metrics can be involved in a maintainability model to improve its usefulness and compactness. We suggest to use the Halstead's Program Vocabulary (HPV) and the Halstead's Number of Delivered Bugs metrics in the model since these two metrics best expend the model based on our observations.

6.1 Threats to Validity

It is a very challenging task to find any open source software system written in RPG (since RPG is used in business applications). Consequently, it is hard to gather RPG source code sets from different domains that would guarantee the generality. We only have source code from one company and they mostly use subroutines that obviously moderates the generality.

6.2 Future Work

We have calculated four different Maintainability Index metrics for RPG. We plan to compare the MI variants with the Maintainability value obtained from the maintainability model. MI variants only use the subset (McCC, LLOC, CD, HVOL) of metrics we applied in the model thus this kind of research could reveal the possible differences between these maintainability measures.

Gathering RPG source code is a harsh task but we would like to gather more source code from more companies to ensure the generality and also investigate the behavior of Halstead metrics at Procedure level.

Acknowledgment

This research was supported by the EU-funded Hungarian national grant GINOP-2.3.2-15-2016-00037 titled "Internet of Living Things".

References

1. Tibor Bakota, Péter Hegedűs, Péter Körtvélyesi, Rudolf Ferenc, and Tibor Gyimóthy. A probabilistic software quality model. In *Software Maintenance (ICSM), 2011 27th IEEE International Conference on*, pages 243–252. IEEE, 2011.
2. Dilek Baski and Sanjay Misra. Metrics suite for maintainability of extensible markup language web services. *IET software*, 5(3):320–341, 2011.
3. Brett A Becker and Catherine Mooney. Categorizing compiler error messages with principal component analysis. In *12th China-Europe International Symposium on Software Engineering Education (CEISEE 2016), Shenyang, China, 28-29 May 2016*, 2016.
4. Frederick P Brooks Jr. The mythical man-month (anniversary ed.). 1995.
5. William H Brown, Raphael C Malveau, Hays W McCormick, and Thomas J Mowbray. *AntiPatterns: refactoring software, architectures, and projects in crisis.* John Wiley & Sons, Inc., 1998.

6. Shyam R Chidamber and Chris F Kemerer. A metrics suite for object oriented design. *IEEE Transactions on software engineering*, 20(6):476–493, 1994.
7. Don Coleman, Dan Ash, Bruce Lowther, and Paul Oman. Using metrics to evaluate software system maintainability. *Computer*, 27(8):44–49, 1994.
8. Maurice Howard Halstead. *Elements of software science*, volume 7. Elsevier New York, 1977.
9. Sandra D Hartman. A counting tool for rpg. In *ACM SIGMETRICS Performance Evaluation Review*, volume 11, pages 86–100. ACM, 1982.
10. Ilja Heitlager, Tobias Kuipers, and Joost Visser. A practical model for measuring maintainability. In *Quality of Information and Communications Technology, 2007. QUATIC 2007. 6th International Conference on the*, pages 30–39. IEEE, 2007.
11. Matinee Kiewkanya, Nongyao Jindasawat, and Pornsiri Muenchaisri. A methodology for constructing maintainability model of object-oriented design. In *Quality Software, 2004. QSIC 2004. Proceedings. Fourth International Conference on*, pages 206–213. IEEE, 2004.
12. Gergely Ladányi, Zoltán Tóth, Rudolf Ferenc, and Tibor Keresztesi. A software quality model for rpg. In *Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on*, pages 91–100. IEEE, 2015.
13. Anuradha Lakshminarayana and Timothy S Newman. Principal component analysis of lack of cohesion in methods (lcom) metrics. *Technical Report TRUAH-CS-1999-01*, 1999.
14. Thomas J McCabe. A complexity measure. *IEEE Transactions on software Engineering*, (4):308–320, 1976.
15. Sanjay Misra, Ibrahim Akman, and Ricardo Colomo-Palacios. Framework for evaluation and validation of software complexity measures. *IET software*, 6(4):323–334, 2012.
16. Sanjay Misra, Murat Koyuncu, Marco Crasso, Cristian Mateos, and Alejandro Zunino. A suite of cognitive complexity metrics. *Computational Science and Its Applications–ICCSA 2012*, pages 234–247, 2012.
17. S Muthanna, Kostas Kontogiannis, Kumaraswamy Ponnambalam, and B Stacey. A maintainability model for industrial software systems using design level metrics. In *Reverse Engineering, 2000. Proceedings. Seventh Working Conference on*, pages 248–256. IEEE, 2000.
18. Farid A Naib. An application of software science to the quantitative measurement of code quality. In *ACM SIGMETRICS Performance Evaluation Review*, volume 11, pages 101–128. ACM, 1982.
19. Paul Oman and Jack Hagemeister. Construction and testing of polynomials predicting software maintainability. *Journal of Systems and Software*, 24(3):251–266, 1994.
20. Maryoly Ortega, María Pérez, and Teresita Rojas. Construction of a systemic quality model for evaluating a software product. *Software Quality Journal*, 11(3):219–242, 2003.
21. Girish Parikh and Nicholas Zvegintzov. The world of software maintenance. *Tutorial on Software Maintenance*, pages 1–3, 1983.
22. Adnan Rawashdeh and Bassem Matalkah. A new software quality model for evaluating cots components. *Journal of Computer Science*, 2(4):373–381, 2006.
23. Martin Shepperd and Darrel C Ince. A critique of three metrics. *Journal of Systems and Software*, 26(3):197–210, 1994.
24. Jonathon Shlens. A tutorial on principal component analysis. *arXiv preprint arXiv:1404.1100*, 2014.