DOI:

QUALITY EVALUATION OF THE FORMULAS PROGRAMMING LANGUAGE

AVALIAÇÃO DA QUALIDADE DA LINGUAGEM DE PROGRAMAÇÃO DAS FÓRMULAS

Aline Vieira Malanovicz
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL - ORCID: https://orcid.org/0000-0002-6668-7365

Abstract
The objective of this research is to analyze the quality of the peculiar Programming Language of Formulas.
The peculiarity of the Formulas Programming Language to be analyzed is reflected in the fact that there are few research papers that report the realization of internally developed languages like this, although they are important to deliver high-quality IT solutions due to the partnership between IT and business.
In-Depth Case Study in a company in the financial sector, as it allows to understand in detail the dynamics of using the language in practice, in addition to identifying aspects of its quality based on testimonials from developers who use it in the company.
Details of the dynamics of the Formulas Programming Language and its functioning Evaluation of its quality regarding the criteria Legibility, Wording, Simplicity, Expressiveness, Modularity, Reliability and Efficiency, which, in their entirety, are violated Presentation and evaluation of improvement proposals for the product development process.
It is understood that such suggestions for improvements have the potential of a practical contribution to qualify the entire product development process used in the company. in addition to being exemplary.
The main contribution of this work lies in the exploration of a case of using a programming language, applied to the process of developing financial products, which is extremely peculiar and innovative – therefore potentially unknown to managers, engineers and developers in the scientific community.

Key words: product development, software engineering, programming language, quality criteria of programming languages, financial calculus

Resumo
O objetivo desta pesquisa é analisar a qualidade da peculiar Linguagem de Programação das Fórmulas.
A peculiaridade da Linguagem de Programação das Fórmulas a ser analisada se reflete no fato de que há poucos trabalhos de pesquisa que relatem a efetivação de linguagens internamente desenvolvidas como essa, embora sejam importantes para entregar soluções de TI de alta qualidade devidas à parceria entre TI e negócio.
Estudo de Caso em Profundidade em uma empresa do setor financeiro, pois permite entender em detalhes a dinâmica de utilização da linguagem na prática, além de permitir identificar aspectos da sua qualidade com base em depoimentos dos desenvolvedores que a utilizam na empresa.
Detalhamento da dinâmica da Linguagem de Programação das Fórmulas e de seu funcionamento Avaliação da sua qualidade quanto aos critérios Legibilidade, Redigibilidade, Simplicidade, Expressividade, Modularidade, Confiabilidade e Eficiência, os quais, em sua totalidade, são violados Apresentação e avaliação de propostas de melhorias para o processo de desenvolvimento de produtos.
Entende-se que tais sugestões de melhorias têm potencial de contribuição prática para qualificar todo o processo de desenvolvimento de produtos utilizado na empresa Os achados deste trabalho de pesquisa mostram a contribuição de apresentar à comunidade uma forma de organização da produção inusitada para os padrões atuais, além de exemplar.
A principal contribuição deste trabalho reside na exploração de um caso de utilização de uma linguagem de programação, aplicada ao processo de desenvolvimento de produtos financeiros, extremamente peculiar e inovadora – por isso potencialmente desconhecida dos gerentes, engenheiros e desenvolvedores na comunidade científica.

Palavras-chave: Desenvolvimento de Produto, Engenharia de Software, Linguagens de Programação, Critérios de Qualidade de Linguagens de Programaçã, Cálculos Financeiros

Avaliação da Qualidade da Linguagem de Programação das Fórmulas

Quality Evaluation of the Formulas Programming Language

Resumo: Este Estudo de Caso analisa a qualidade da peculiar Linguagem de Programação das Fórmulas, desenvolvida internamente na empresa estudada e utilizada para o desenvolvimento dos cálculos financeiros para os produtos de financiamento da empresa. A coleta de dados incluiu entrevistas, consulta documental, análise de artefatos como sistemas, telas e tutoriais, e observação participante. A análise dos resultados permitiu entender em detalhes a dinâmica da Linguagem das Fórmulas e de seu funcionamento, permitindo avaliar sua qualidade quanto aos critérios Legibilidade, Redigibilidade, Simplicidade, Expressividade, Modularidade, Confiabilidade e Eficiência, os quais são todos violados. Para superar tais problemas, são apresentadas e avaliadas algumas propostas de melhorias implantadas ou projetadas, incluindo a reforma do ambiente de testes e a criação de macro comandos para cálculos. A avaliação dessas propostas contribui para (expectativa de) melhora dos critérios de qualidade de linguagens de programação, e também do quesito Custo.
Palavras-chave: Desenvolvimento de Produto; Engenharia de Software; Linguagens de Programação; Critérios de Qualidade de Linguagens de Programação.

Abstract*: This Case Study analyzes the quality of the peculiar Formula Programming Language, developed internally in the studied company and used for the development of financial calculations for the company's financing products. Data collection included interviews, documentary consultation, analysis of artifacts such as systems, screens and tutorials, and participant observation. The analysis of the results allowed us to understand in detail the dynamics of the Formula Language and its operation, allowing to evaluate its quality in terms of the criteria Readability, Writability, Simplicity, Expressivity, Modularity, Reliability and Efficiency, which, in their entirety, are violated. . To overcome these problems, some proposed or implemented improvement proposals are presented and evaluated, including the testing environment reform and the creation of macro commands for calculations. The evaluation of these proposals contributes to (expectation of) improvement of the programming language quality criteria and also in the Cost aspect.*
Keywords*: product development; software engineering; programming language; quality criteria of programming languages.*

1 Introdução

Para organizações com uso intensivo de tecnologia de informação, como as instituições financeiras, o desenvolvimento de produtos constitui uma tarefa complexa e estratégica, e estes produtos são implantados em sistemas de cálculo e controle financeiro (Tavares; Thiry-Cherques, 2009). Tais sistemas são essenciais para as empresas, mesmo aqueles desenvolvidos há vários anos com tecnologias hoje consideradas obsoletas, que passam por contínuas alterações e não foram submetidos a ações sistemáticas de melhoria, os chamados sistemas legados (Valle et al., 2005).

Com as mudanças na economia, nas leis e na gestão das empresas, os sistemas de software efetivamente úteis para as empresas precisam ser evoluídos (Sommerville, 2016). Esses sistemas legados devem acompanhar a constante evolução das tecnologias e técnicas de criação de sistemas, ou podem tornar-se obsoletos ou até extintos no mercado (Barbosa;

Cândido, 2016). No entanto, essa adaptação pode ser difícil, por sua tecnologia, acoplamento de código ou arquitetura inapropriados, pois a tecnologia à época de seu desenvolvimento ficou defasada com o passar do tempo, demandando uma modernização (Freitas, 2017). Podem apresentar baixa qualidade de código, tornando a sua manutenção difícil, e necessitando de vários ciclos de manutenções para continuarem ativos e operacionais, por isso existe uma grande demanda para a migração de sistemas legados de arquiteturas monolíticas para se beneficiarem de novas tecnologias (Cintra; Vendramel, 2019).

Para enfrentar esse desafio, convém avaliar de que modo os sistemas legados estão implementados, em que condições de manutenção estão, suas opções de modernização, necessidades de melhoria, estruturas de dados, e especialmente as linguagens de programação utilizadas, pois são itens determinantes da estratégia de evolução para todo o sistema (Valle et al., 2005). Os elementos que denotam sistemas legados incluem o desenvolvimento ou uso de tecnologias obsoletas, principalmente em relação a linguagens de programação (Barbosa; Cândido, 2016). As linguagens de programação também evoluem ao longo do tempo, tanto por novos requisitos de negócio ou necessidades técnicas, tornando recursos e construções antigas obsoletas, o que traz problemas como custo de manutenção e demora no aprendizado.

Uma empresa da área financeira desenvolveu internamente uma linguagem de programação específica utilizada para o desenvolvimento dos cálculos financeiros para os produtos de financiamento implementados no sistema legado da empresa, a Linguagem de Programação das Fórmulas. Considerando esse contexto, o **objetivo** desta pesquisa é analisar a qualidade da peculiar Linguagem de Programação das Fórmulas. O Estudo de Caso em Profundidade (Yin, 2015), realizado em uma empresa do setor financeiro, foi o método escolhido para alcançar esse objetivo, pois permite entender em detalhes a dinâmica de utilização da linguagem na prática, além de permitir identificar aspectos da sua qualidade com base em depoimentos dos desenvolvedores que a utilizam na empresa.

Vale ressaltar que a peculiaridade da linguagem de programação a ser analisada também se reflete no fato de que há poucos trabalhos de pesquisa que relatem a efetivação de linguagens internamente desenvolvidas como essa, embora possivelmente sejam importantes para entregar soluções de tecnologia com alta qualidade devido à efetivação de uma parceria entre as áreas de tecnologia e negócio no processo de desenvolvimento de produtos (Bassellier; Benbasat, 2004). Pretende-se aqui auxiliar na redução dessa lacuna, pois, a exemplo de outras pesquisas (Freitas, 2017) (Cintra; Vendramel, 2019) (Malanovicz, 2021), esta também apresenta propostas de melhoria para algum aspecto dos sistemas legados, aqui em especial a linguagem de programação. Como em outros trabalhos, resgatam-se critérios da literatura para avaliar a complexidade e a qualidade de uma linguagem de programação usada em sistemas legados.

As bases conceituais que nortearam a pesquisa são apresentadas na Seção 2, e a descrição detalhada do percurso metodológico seguido nesta pesquisa está na Seção 3. A Seção 4 demonstra os resultados do trabalho, a Seção 5 apresenta uma breve discussão dos achados, e a seção 6 resume as conclusões desta pesquisa, apontando possíveis investigações adicionais suscitadas pelo trabalho.

2 Referencial Teórico

Esta seção apresenta brevemente os conceitos essenciais para compreender a pesquisa. O **processo de desenvolvimento de software** é definido na Engenharia de Software como um método de trabalho estruturado em etapas (análise, projeto, codificação, teste, implantação, manutenção) que objetiva produzir sistemas para determinada aplicação (Pressman, 2014). A etapa de **codificação/programação** é o processo de escrita, teste e manutenção de um programa de computador. Os programas são escritos em alguma linguagem de programação, e depois são compilados ou interpretados para linguagem e máquina, e então são processados ou executados (Knuth, 2011). A **manutenção** de sistemas também pode ser considerada como parte do processo de desenvolvimento, pois envolve realizar adaptações ou alterações ou ajustes de qualquer abrangência em sistemas existentes (Sommerville, 2016).

Sistemas legados são sistemas de informações já existentes, essenciais ao funcionamento das organizações e desenvolvidos em algum momento do passado, tendo sido submetidos a mudanças contínuas (mas não a melhorias sistemáticas) para adequação a novos produtos, requisitos e regras de negócio (Sommerville, 2016). Frequentemente, constituem a parte mais importante do fluxo de informações dentro das organizações, são os principais veículos para a consolidação das informações sobre o negócio, e o conhecimento neles agregado constitui um patrimônio corporativo significativo (Valle et al., 2005).

Linguagens de programação são os métodos padronizados para comunicar instruções para os computadores, na forma de conjuntos de regras sintáticas e semânticas usadas para definir os algoritmos nos programas (Sebesta, 2011). As linguagens de programação podem ser classificadas cronologicamente em cinco Gerações, sendo as de 1ª e 2ª gerações consideradas linguagens de baixo nível, e as demais, como linguagens de alto nível (Sebesta, 2011). Vale destacar que esta taxonomia é usual na literatura da área (Sebesta, 2011) (Price; Toscani, 2008).
(1ª) **linguagem de máquina** – notação numérica binária para códigos de operação e endereços;
(2ª) **linguagens simbólicas** (*Assembly*) – mnemônicos no lugar dos códigos binários, e execução só depois da montagem, a tradução para linguagem de máquina;
(3ª) **linguagens orientadas ao usuário** (*COBOL*, *BASIC*, *C*, *Pascal*) – instruções imperativas (entrada/saída, cálculo aritmético/lógico, desvios condicionais, incondicionais e iterativos);
(4ª) **linguagens orientadas à aplicação** (*ABAP*, *Delphi*, *PostScript*, *SQL*) – construções que refletem a terminologia e os elementos usados na descrição do problema; e
(5ª) **linguagens de conhecimento** (*LISP*, *ML*, *PROLOG*) – representação do conhecimento essencial para as simulações de Inteligência Artificial.

Assim, os principais **critérios para avaliação da qualidade de uma linguagem de programação** incluem os seguintes (Sebesta, 2011):

- **Legibilidade** – facilidade com que os programas podem ser lidos e compreendidos, o que influi na produtividade, depuração e manutenção do software;
- **Redigibilidade** – naturalidade da forma de expressar a solução para um problema, sem desviar a atenção do programador para "truques" da linguagem;
- **Simplicidade** – facilidade de se conhecer a linguagem toda;

- **Expressividade** – simplicidade das construções para operações comuns, suporte à abstração;
- **Modularidade** – possibilidade de escrever um programa por partes, para melhor o estruturar e compreender, o que influi na facilidade em escrever programas grandes;
- **Confiabilidade** – detecção de incompatibilidades de tipo em tempo de compilação, tratamento de exceções, impedimento de sinonímia;
- **Eficiência** – velocidade de execução dos programas, economia do uso de memória, esforço necessário para produzir os programas e para mantê-los.

Linguagem *Assembly* é uma notação razoavelmente legível por humanos para o código de máquina utilizado por uma arquitetura de computador específica (Weber, 2012). Utiliza mnemônicos para comandos (p.ex.: ADD para adição) e pode usar **macro comandos** (mnemônicos pré-traduzidos para uma sequência de comandos) (Price; Toscani, 2008). A conversão da linguagem de montagem para o código de máquina é feita pelo montador ou *assembler*, que é um programa tradutor de comandos, mais simples que um compilador (Knuth, 2011).

Cada arquitetura de computador é definida pelo conjunto de instruções e operandos que formam sua linguagem *Assembly*. Por exemplo (Weber, 2012):

- a arquitetura do computador IAS de Von-Neumann, de 1946, é de **1-endereço** (instruções no formato Operação-Operando);
- a arquitetura do computador EDVAC, de 1951, é de **4-endereços** (instruções no formato Operação-Operando1-Operando2-Resultado-PróximaInstrução);
- e as de **3-endereços**, Operação-Operando1-Operando2-Resultado (Figura 1 – Exemplos de programas em linguagem *Assembly*).

(a) Programa "Hello World"

```
SECTION .data
msg:
 db "Hello, World!\n"
 len equ $ - msg
SECTION .text
 global start
start:
 mov edx,len
 mov ecx,msg
 mov ebx,1
 mov eax,4
 int 0x80
 mov ebx,0
 mov eax,1
 int 0x80
```

(b) Programa "A=((B+C)*D–E)/F em arquitetura de 3 endereços"

endereço	instrução	comentário
e_1	ADD B C A	Soma B com C, resultado em A; incrementa PC
e_1+1	MUL A D A	Multiplica A por D, resultado em A; incrementa PC
e_1+2	ADD A E A	Soma A com E, resultado em A; incrementa PC
e_1+3	SUB A F A	Subtrai F de A, resultado em A; incrementa PC
e_1+4	DIV A G A	Divide A por G, resultado em A; incrementa PC
e_1+5	DIV A H A	Divide A por H, resultado final em A; incrementa P
e_1+6	HALT	Fim do programa

Figura 1. Exemplos de programas em linguagem *Assembly*.
Fonte: (a) (Scriptol, 2018); (b) (Weber, 2012)

COBOL (*COmmon Business Oriented Language*, de 1959) é uma linguagem de programação orientada para o processamento de dados comerciais, sendo a linguagem mais utilizada em manutenção de sistemas legados (ABPC, 2000), geralmente escolhida para fazer cálculos financeiros por suportar aritmética inteira aplicada a números muito grandes (milhões, bilhões), e poder lidar com números muito pequenos como frações de centavos. Outras características são a formatação, classificação e geração de relatórios, e a elevada legibilidade e documentação (Figura 2 – Exemplos de programas em linguagem COBOL), mais valiosas quanto mais antigos são os sistemas (CSIS, 2010).

(a) Programa "Hello World"

```
IDENTIFICATION DIVISION.
PROGRAM-ID. HELLO-WORLD.

ENVIRONMENT DIVISION.

DATA DIVISION.

PROCEDURE DIVISION.
DISPLAY "Hello, World!".
STOP RUN.
```

(b) Programa "A=((B+C)*D–E)/F"

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CALCULO.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

 77 A       PIC 99 VALUE 2.
 77 B       PIC 99.
 77 C       PIC 99.
 77 D       PIC 99.
 77 E       PIC 99.
 77 F       PIC 99.
PROCEDURE DIVISION.
COMPUTE A=((B+C)*D–E)/F.
DISPLAY "A = ", A.
STOP RUN.
```

Figura 2. Exemplos de programas em linguagem COBOL.
Fonte: (a) (Scriptol, 2018); (b) autores.

3 Metodologia

Esta pesquisa tem caráter **exploratório** e natureza **qualitativa**, visando uma análise intensiva de uma situação particular, em que se coloca questão tipo **como**, e o foco está em um processo inserido em um contexto da vida real. Por isso optou-se pelo **Estudo de Caso** (Yin, 2015). A seguir, é descrito o **Objeto de Estudo**.

O **critério para seleção do caso** é a peculiaridade da existência, na empresa, de uma (meta)linguagem de programação desenvolvida internamente para implementar os cálculos financeiros dos produtos de financiamento comercializados pela empresa. O **caso selecionado** é a Linguagem de Programação das Fórmulas, utilizada no processo de desenvolvimento de produtos financeiros de uma empresa. Esses produtos exigem o acompanhamento pós-venda dos saldos e prestações das operações, os quais são atualizados por procedimentos de cálculo programados nessa linguagem, e processados no sistema legado de controle financeiro (desenvolvido em COBOL e implantado em *mainframe* na década de 1970). Talvez importe destacar que a linguagem aqui analisada NÃO é a linguagem comercial COBOL, mas sim a Linguagem de Programação das Fórmulas, descrita nas seções de Resultados deste trabalho. Nos últimos cinco anos, a empresa conduz um projeto de migração dos sistemas para uma plataforma mais moderna, para obter ganhos de eficiência e produtividade.

Os **Procedimentos de Coleta de Dados** incluíram coleta de dados realizada por meio de: Entrevistas com todos os desenvolvedores da equipe de Fórmulas; Consulta Documental, para caracterização do cenário da pesquisa; Análise de Artefatos como sistemas, telas e tutoriais; e Observação Participante do dia-a-dia do processo de trabalho em diversos projetos (Bauer; Gaskell, 2015). As entrevistas foram realizadas no horário e local de trabalho dos desenvolvedores, tendo duração média de 30 minutos e questões gerais simples e abertas (próprias para estudos exploratórios): "*Como é a Linguagem de Programação das Fórmulas? Como é a qualidade dessa linguagem? Algo poderia ser melhorado?*".

Os **Procedimentos de Análise de Dados** contemplaram as etapas de análise a seguir descritas. Os dados provenientes da coleta de dados desta pesquisa foram organizados por temas (seção 4). Sua análise seguiu três etapas: Caracterização da Linguagem,

Classificação conforme Critérios de Avaliação de Qualidade, e Melhorias Propostas. Para a etapa inicial, de organização dos dados, feita por meio da Caracterização da Linguagem, são mostrados seus diferentes aspectos. Entre eles, incluem-se, logo após um histórico da criação da linguagem: um exemplo de fórmula, os comandos da linguagem, suas memórias, os eventos que disparam as fórmulas, os conjuntos de fórmulas específicos, o projeto original, o desenvolvimento e manutenção das fórmulas, e como são feitos os testes.

A etapa de análise dos dados propriamente dita foi realizada por meio da Classificação da linguagem, já previamente caracterizada, conforme os Critérios de Avaliação de Qualidade de linguagens de programação. Tais critérios estão apresentados na seção 2 deste artigo, servindo de fundamento teórico para a análise dos resultados. A avaliação dos resultados fundamenta-se na apresentação de Propostas de Melhorias para a Linguagem de Programação das Fórmulas. Cada melhoria tem como objetivo mitigar os problemas identificados em um ou mais dos critérios de qualidade avaliados na etapa de Classificação.

4 Resultados

4.1 Histórico da Linguagem

A Linguagem de Fórmulas foi criada internamente na empresa na década de 1980, como uma solução para atender a uma demanda específica de elaboração de cálculos financeiros por parte do analista de negócios do setor financeiro. Pode ser considerada uma metalinguagem, pois precisa ser interpretada para *COBOL* (não tendo um compilador próprio, nem uma definição de gramática própria) para então ser executada. Sua integração aos demais controles do sistema de controle financeiro corporativo se dá quando as estruturas de dados descritoras das operações de financiamento são atualizadas com os valores calculados pelas Fórmulas.

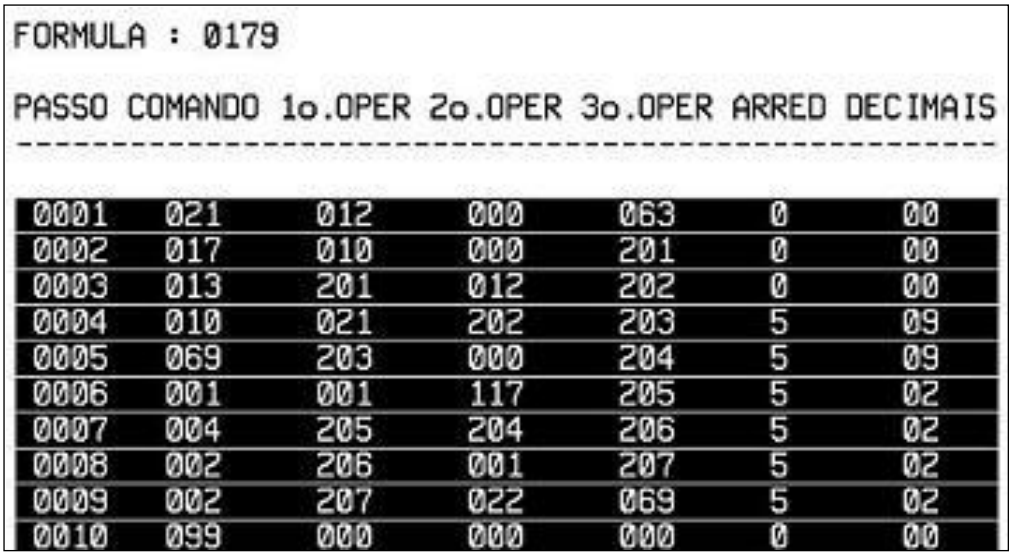
A criação desta metalinguagem foi sugerida por uma consultoria de tecnologia e sistemas contratada pela empresa. A motivação para isso foi a possibilidade de os analistas de negócios definirem os cálculos financeiros, sem demandar os pouquíssimos programadores existentes no pessoal da empresa na época (década de 1980). Também não existia a necessidade de os analistas de negócio aprender *COBOL*.

Além disso, os usuários teriam agilidade para a produção de cálculos, por não precisarem realizar especificações técnicas para o desenvolvimento de tais cálculos pela equipe de desenvolvedores COBOL para mainframe atuante na empresa. Essa agilidade também se verificava na maior rapidez da interpretação da (até então simples) linguagem das fórmulas, comparada à compilação de programas no mainframe na época, que poderia demorar horas (uma estimativa dessa diferença é de 100 para 1).

4.2 Caracterização da Linguagem

Um exemplo típico de Fórmula é o cálculo dos valores exigíveis na próxima parcela de um financiamento. Cada Fórmula (Figura 3 – Exemplo de Fórmula) tem um identificador numérico e é estruturada como uma sequência numerada de **passos**, que são executados sequencialmente. Cada **passo** tem uma **instrução** de estrutura semelhante às das

linguagens de montagem de 3–endereços (com duas entradas e uma saída), informando também o método de arredondamento do cálculo e o número de casas decimais do resultado. Uma **instrução** da Linguagem das Fórmulas é como: Passo-Comando-Operando1-Operando2-Resultado-Arredondamento-Decimais, e todos os termos são números, não mnemônicos (ADD, SUB, etc.) comuns em linguagens de montagem.



FORMULA : 0179

PASSO	COMANDO	1o.OPER	2o.OPER	3o.OPER	ARRED	DECIMAIS
0001	021	012	000	063	0	00
0002	017	010	000	201	0	00
0003	013	201	012	202	0	00
0004	010	021	202	203	5	09
0005	069	203	000	204	5	09
0006	001	001	117	205	5	02
0007	004	205	204	206	5	02
0008	002	206	001	207	5	02
0009	002	207	022	069	5	02
0010	099	000	000	000	0	00

Figura 3. Exemplo de Fórmula. Fonte: coleta de dados.

Os **comandos** da linguagem (Figura 4 – Comandos para Codificação de Fórmulas**Erro! Fonte de referência não encontrada.**) permitem realizar tarefas como: *processamento aritmético* (soma, subtração, multiplicação, divisão, potência); *cálculos financeiros* (juros lineares, juros exponenciais pelo calendário civil, juros exponenciais pelo calendário comercial, valor de prestação pela tabela PRICE, busca de cotação de moedas e indexadores econômicos por data, cálculo de número de dias entre duas datas pelo ano civil ou pelo ano comercial); *instruções de desvio* (condicional simples (=,<,>,<=,>=,<>), incondicional).

Nº	Descrição	1º operando ou M1	2º operando ou M2	3º operando ou M3
01	Soma de 2 valores	Valor	Valor	$M3 = M1 + M2$
02	Subtração de 2 valores	Valor	Valor	$M3 = M1 - M2$
03	Multiplicação de 2 valores	Valor	Valor	$M3 = M1 \times M2$
04	Divisão de 2 valores	Valor	Valor	$M3 = \frac{M1}{M2}$
05	Potência	Valor	Valor	$M3 = M1^{M2}$
06	Raiz	Valor	Valor	$M3 = \sqrt[M2]{M1}$
07	Nova data vencimento variável (anterior)	Data		Nova data, 1º dia útil anterior, se M1 for não útil
10	Coeficiente de juros exponenciais do período, ano de 365 dias	Taxa decimal de juros	Nº de dias do período	$M3 = \left(\sqrt[365]{(1+M1)^{M2}}\right) - 1$
12	Coeficiente de juros exponenciais do período, ano de 360 dias	Taxa decimal de juros	Nº de dias do período	$M3 = \left(\sqrt[360]{(1+M1)^{M2}}\right) - 1$
13	Nº de dias entre 2 datas (ano civil)	Data final	Data inicial	$M3 = M1 - M2$
14	Nº de dias entre 2 datas (ano comercial)	Data final	Data inicial	$M3 = M1 - M2$
15	Montagem de nova data	Data (dia)	Data (mês/ano)	Dia da data (M1) e mês e ano da data (M2)

Figura 4. Comandos para Codificação de Fórmulas. Fonte: coleta de dados.

Os **operandos** das instruções das fórmulas são chamados de Memórias e são codificados por números (Figura 5 – Operandos ou "Memórias" das Fórmulas). Representam campos das estruturas de dados descritoras das operações de financiamento. As estruturas de dados estão organizadas em três partes: informações de cada operação de financiamento ("Planos"); informações de cada parcela de cada operação ("Recibos"); e informações comuns às operações de financiamento do mesmo tipo ("Fundos"). Exemplos de Memórias seriam: Taxa de Juros de uma operação, Valor de Prestação de uma parcela, Código da Moeda de um tipo de operações, Saldo de uma operação (Figura 5 – Operandos ou "Memórias" das Fórmulas).

NUM	CORRESP	ARQUIVO	TIPO	DESCRIÇÃO
1	56	PLANOS	E	SALDO ESTÁTICO EM UM
2	57	PLANOS	E	SALDO ESTÁTICO EM R$
3		PLANOS	E	PARCELA DO PRINCIPAL CRONOGRAMA
4	59	RECIBOS	E	VALOR DA PRESTAÇÃO EM UMC
5	60	RECIBOS	E	VALOR DA PARCELA VENCIDA EM UMI
6		MEIN	E	INDICADOR DE SALDO VENCIDO DE VALOR MA
7	62	RECIBOS	E	ENCARGOS IV
8		RECIBOS	E	VALOR PAGO (R$)
9		RECIBOS	E	DATA INÍCIO PERÍODO
10		RECIBOS	E	DATA DO VENCIMENTO
11		MEIN	E	DATA FINAL DO MÊS DA DATA PROCESSAMENT
12		MEIN	E	DATA DA POSIÇÃO DO PROCESSAMENTO
13		PLANOS	E	DATA FINAL DE RETORNOS (VENC. FINAL)
14		RECIBOS	E	DATA VALORIZAÇÃO RETORNOS
15		RECIBOS	E	NÚMERO PARCELAS VINCENDAS NA AMORTIZ
16		RECIBOS	E	DATA DE VENCIMENTO DO PERÍODO SEGUINTE
17		PLANOS	E	TAXA/COEF OUTROS IV
18	139	RECIBOS	E	DATA VALORIZAÇÃO VENCIMENTO
19		RECIBOS	E	INDICADOR DE SITUAÇÃO (0=CARENCIA; 1=AM
20	68	RECIBOS	E	JUROS TOTAIS
21		FUNDOS	E	COEFICIENTE DE JUROS

Figura 5. Operandos ou “Memórias” das Fórmulas. Fonte: coleta de dados.

Os **resultados** das instruções das fórmulas também são Memórias, e estas podem ser o valor de saída dos campos das estruturas de dados, para atualização dos valores no fim dos cálculos, ou podem ser Memórias Auxiliares (espaços de armazenamento temporário de valores para cálculos feitos em dois ou mais passos). Essas memórias podem ser utilizadas e reutilizadas, e são livres de tipagem, podendo receber tanto valores monetários como datas ou códigos.

Cada Memória tem código diferente para o seu valor de entrada e para o seu valor de saída, sendo que o valor de entrada deve ser utilizado somente para leitura (como Operando1 ou Operando2), e o valor de saída deve ser utilizado somente para escrita (como Resultado). A atualização dos campos do sistema legado é feita no fim da execução de cada fórmula, pela soma desse Resultado ao valor de entrada da mesma Memória.

O **arredondamento** codifica o método de arredondamento do resultado:

- “0”: para baixo; “9”: para cima;
- “5”: 0-1-2-3-4 para baixo e 5-6-7-8-9 para cima.

As **decimais** indicam o número de casas decimais da Memória que contém o Resultado.

Eventos são os momentos da existência de um financiamento que requerem a realização de cálculos e indicam ao sistema legado a necessidade de disparo das fórmulas. Por exemplo: Início do Financiamento (início dos cálculos das parcelas da dívida); Fim de Mês

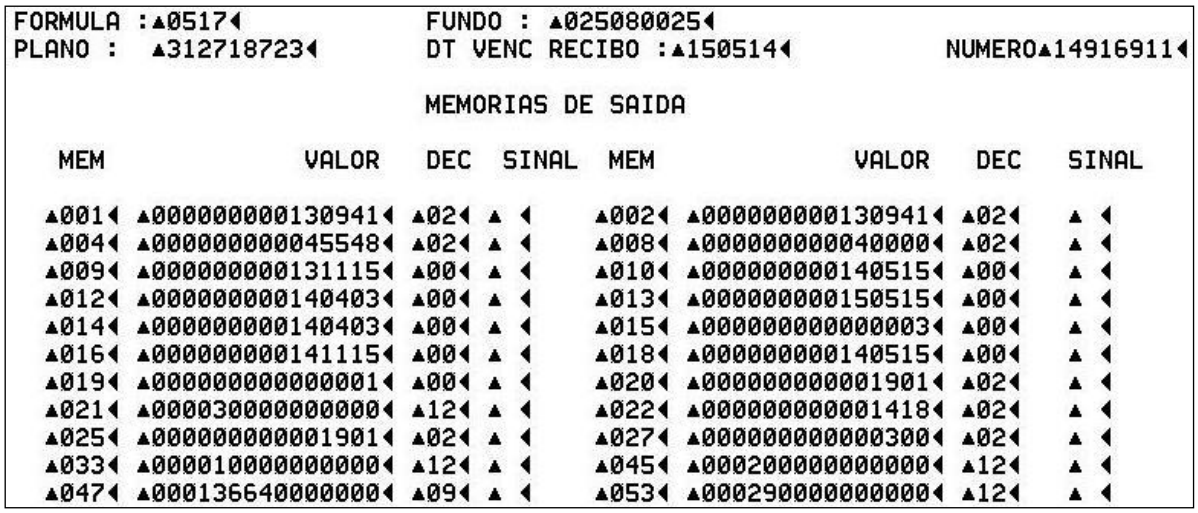
(momento de atualizar os saldos com encargos, para fins fiscais); Data de Vencimento (momento de atualizar o saldo e calcular o valor da próxima prestação); Pagamento (momento de atualizar o saldo e distribuir o total pago entre as rubricas de amortização, juros e outros encargos, calcular encargos de inadimplência no caso de pagamento em atraso, e descontos pertinentes no caso de pagamento antecipado).

Existem **conjuntos de fórmulas específicos** para cada tipo de financiamento operado pela empresa, sendo uma fórmula diferente para cada evento (em média oito fórmulas por conjunto). Atualmente existem cerca de 250 diferentes conjuntos de fórmulas em manutenção para as operações de financiamento da empresa, incluindo tanto produtos à venda, quanto produtos comercializados no passado com operações ainda em acompanhamento.

O **projeto original da linguagem** contemplava somente uma aplicação limitada aos cálculos de juros e amortização de prestações, atualização de saldos e apropriações contábeis de encargos. As fórmulas tinham cerca de 20 passos, e ofereciam a possibilidade de parametrizar os procedimentos de cálculos com relativa simplicidade. Com as demandas de alteração dos sistemas legados (por exemplo, a crescente complexidade dos produtos de financiamentos), as fórmulas passaram a tratar também conversão de moedas, controle de dias úteis ou não úteis, concessão de descontos para pagamentos em dia, e controles extra cálculo como situação de inadimplência. Atualmente existem fórmulas com mais de 200 linhas, elaboradas para realizar controles complexos, que exigiram a construção de fórmulas repletas de desvios condicionais e incondicionais, pois a linguagem não permite a estruturação de módulos de código (não existe controle de iteração, nem se-então-senão, nem chamada de sub-rotinas).

O **desenvolvimento e a manutenção das fórmulas** são realizados sem o uso de um ambiente de desenvolvimento próprio para a linguagem. A equipe de desenvolvimento lança mão dos recursos de planilhas eletrônicas de pacotes de software de escritório para realizar a construção e a documentação de cada fórmula. Esse desenvolvimento geralmente consiste em uma cópia e adaptação de outras fórmulas preexistentes que atendam a algum tipo de financiamento semelhante ao que está para ser desenvolvido, quando surge uma demanda de novos cálculos ou de adaptações nos já existentes. Uma fórmula é carregada no *mainframe* via digitação literalmente "passo a passo" do código numérico do número de fórmula e dos termos de cada instrução. Não existe a opção de inclusão de comentários no código, por isso a documentação requer um controle manual, ficando separada, nas planilhas de apoio. Com a necessidade de manutenção das fórmulas, o controle de versões também é feito manualmente, mantendo-se nas mesmas planilhas um registro de datas, responsáveis e motivos das alterações.

Os **testes de fórmulas** são realizados diretamente no *mainframe*, com entrada de dados por digitação de dados operações por operação. Não existe opção de automação dos testes. É necessário informar manualmente: o número da fórmula a ser testada; o número da operação; o número do tipo de financiamento; a data do vencimento daquela operação; e o número desta parcela. Além disso, tanto a entrada dos valores de teste como também a saída dos seus resultados são feitas em números (por exemplo, o resultado "001 000000000130941 02 _" significa que o Saldo a Vencer (Memória 001) da operação tem o valor positivo de R$ 1.309,41) (Figura 6 – Teste de Fórmulas).



FORMULA :▲0517◀ FUNDO : ▲025080025◀
PLANO : ▲312718723◀ DT VENC RECIBO :▲150514◀ NUMERO▲14916911◀

MEMORIAS DE SAIDA

MEM	VALOR	DEC	SINAL	MEM	VALOR	DEC	SINAL
▲001◀	▲000000000130941◀	▲02◀	▲ ◀	▲002◀	▲000000000130941◀	▲02◀	▲ ◀
▲004◀	▲000000000045548◀	▲02◀	▲ ◀	▲008◀	▲000000000040000◀	▲02◀	▲ ◀
▲009◀	▲000000000131115◀	▲00◀	▲ ◀	▲010◀	▲000000000140515◀	▲00◀	▲ ◀
▲012◀	▲000000000140403◀	▲00◀	▲ ◀	▲013◀	▲000000000150515◀	▲00◀	▲ ◀
▲014◀	▲000000000140403◀	▲00◀	▲ ◀	▲015◀	▲000000000000003◀	▲00◀	▲ ◀
▲016◀	▲000000000141115◀	▲00◀	▲ ◀	▲018◀	▲000000000140515◀	▲00◀	▲ ◀
▲019◀	▲000000000000001◀	▲00◀	▲ ◀	▲020◀	▲000000000001901◀	▲02◀	▲ ◀
▲021◀	▲000030000000000◀	▲12◀	▲ ◀	▲022◀	▲000000000001418◀	▲02◀	▲ ◀
▲025◀	▲000000000001901◀	▲02◀	▲ ◀	▲027◀	▲000000000000300◀	▲02◀	▲ ◀
▲033◀	▲000010000000000◀	▲12◀	▲ ◀	▲045◀	▲000200000000000◀	▲12◀	▲ ◀
▲047◀	▲000136640000000◀	▲09◀	▲ ◀	▲053◀	▲000290000000000◀	▲12◀	▲ ◀

Figura 6. Teste de Fórmulas. Fonte: coleta de dados.

Esta ferramenta de testes executa cada fórmula, porém "artificialmente", independentemente do evento que a dispara, e produz um resultado indireto, que precisa ser criticado para receber validação. A verificação dos cálculos das fórmulas é feita pela comparação com os mesmos cálculos realizados em planilha eletrônica, e a "igualdade no centavo" indica a validade do cálculo desenvolvido. Depois dessa validação, é feita a liberação do conjunto de fórmulas desenvolvido, informando-se o seu código identificador, os descritores de cálculos específicos, e a documentação pertinente.

4.3 Classificação da Linguagem segundo Critérios de Qualidade

Nesta subseção, são apresentados os resultados da etapa de análise de dados propriamente dita. Nesta fase, as características da linguagem, apresentadas na subseção 4.2, são organizadas e tratadas de acordo com os Critérios de Avaliação da Qualidade de Linguagens de Programação apresentados na seção 2. Os resultados dessa análise são exibidos nas Figuras 7, 8 e 9.

O critério Legibilidade foi analisado, e sua avaliação é exibida na Figura 8 – Melhorias para Linguagem das Fórmulas. Percebe-se, pela análise de dados, a dificuldade de leitura dos programas escritos na linguagem.

	Aspectos Peculiares da Linguagem das Fórmulas
Legibilidade	— baixíssima legibilidade dos programas (ausência de mnemônicos para operadores e para operandos, existindo somente representações numéricas para todos os operadores e operandos - mais dificilmente legível do que linguagem Assembly) — impossibilidade de comentários no código, acarretando dificuldade de leitura e entendimento e de produção de documentação — ausência de padronização na linguagem, acarretando miscelânea de estilos de programação e dificuldade de leitura do código — possibilidade de reuso de memórias (*overwrite*), prejudicando o entendimento do código — dificuldade de compreensão das regras de negócio implementadas nas fórmulas — existência de sinonímia em algumas fórmulas (projeção, pagamento antecipado e liberação), com uso de nomes diferentes para os mesmos campos, e uso de nomes iguais para conteúdos diferentes

Redigibilidade	— metalinguagem interpretada para COBOL, sem um compilador próprio, sem uma definição de gramática própria — dificuldades com memórias de entrada e saída, cujo funcionamento das rotinas de atualização revela-se não-intuitivo — ausência de controle de ponteiros ou rótulos para desvio (desvios são *hardcode*), ausência de atualização automática dos desvios — ausência de estruturas de controle de fluxo do tipo se-então-senão, acarretando uso de desvios condicionais duplos redundantes — existência de sinonímia em algumas fórmulas (projeção, pagamento antecipado e liberação), com uso de códigos diferentes para os mesmos campos, e uso de códigos iguais para conteúdos diferentes — geração de informação de data obrigatoriamente no formato não-intuitivo AMD — geração de constante de números grandes em pedaços quebrados em dois operandos ou até mesmo usando mais de um comando
Simplicidade	— dificuldade de aprendizado de todas as peculiaridades da linguagem (memórias de entrada e saída, atualização aditiva de valores, códigos de campos diferenciados de alguns tipos de fórmulas) — metalinguagem interpretada para COBOL, sem um compilador próprio, sem uma definição de gramática própria — histórico de ampliações da linguagem para tratamento de funções extra-cálculo — desconhecimento, por parte da equipe atual de desenvolvimento, das razões para decisões tomadas a respeito da linguagem — possibilidade de modo de endereçamento imediato (*hardcode*) permitindo constantes "escondidas" dentro das fórmulas — existência de sinonímia em algumas fórmulas, com uso de nomes diferentes para os mesmos campos
Expressividade	— baixa parametrização e consequente dificuldade de manutenção dos numerosos diferentes conjuntos de fórmulas — limitação do escopo da linguagem para controles complexos — ausência de estruturas de controle de fluxo iterativo — ausência de estruturas de controle de fluxo do tipo se-então-senão — impossibilidade de estruturação de módulos de código — dificuldade de modificação de sistemas complexos — dificuldades com memórias de entrada e saída, cujo funcionamento revela-se não-intuitivo — dificuldades com a expansão da linguagem para tratamento de controles extra-cálculo (p.ex.: indicadores de inadimplência)
Confiabilidade	— fragilidade dos controles de fluxo de processamento da linguagem (desvios são feitos em *hardcode*) — ausência de controle de ponteiros ou rótulos para desvio, inexistência de atualização automática dos desvios — ausência de tipos e consequente possibilidade de miscelânea de operações (por exemplo, entre datas e valores) — ausência de compilador próprio e consequente ausência de detecção de incompatibilidades de tipo em tempo de compilação — dificuldades e limitações de utilização do ambiente de testes, dificuldade de depuração de programas — possibilidade de reuso de memórias (*overwrite*), permitindo uma inadequada sobreposição de valores — possibilidade de modo de endereçamento imediato (*hardcode*) permitindo constantes "escondidas" dentro das fórmulas — existência de sinonímia em algumas fórmulas (projeção, pagamento antecipado e liberação), com uso de códigos diferentes para os mesmos campos, e uso de códigos iguais para conteúdos diferentes
Eficiência	— dificuldade de aprendizado de todas as peculiaridades da linguagem — inexistência de um ambiente de desenvolvimento amigável — dificuldade de manutenção dos numerosos diferentes conjuntos de fórmulas pela baixa parametrização — dificuldade de produção de documentação (impossibilidade de comentários no código) — limitação da possibilidade de desenvolvimento de cálculos parametrizados — dificuldade de realização de testes e depuração de programas — dificuldade de modificação de sistemas complexos

Figura 7. Avaliação dos Aspectos Peculiares da Linguagem das Fórmulas segundo os critérios.
Fonte: coleta e análise de dados.

A análise realizada contemplou diferentes critérios (Figura 7 – Avaliação dos Aspectos Peculiares da Linguagem das Fórmulas segundo os critérios). É possível discernir, com base nessa análise de dados, que a Linguagem das Fórmulas viola todos os critérios de avaliação de qualidade de linguagens de programação citados. Tanto a Legibilidade quanto a Redigibilidade foram critérios em que a avaliação da linguagem deixou a desejar. Da mesma forma, os critérios Simplicidade (apesar de sua forma de dois operandos) e Expressividade (além de difícil de ler e redigir, também avalia-se como pouco expressiva). Os demais critérios, Confiabilidade e Eficiência têm avaliação negativa, talvez como consequência da baixa qualidade dos demais critérios.

4.4 Propostas de Melhorias

Tendo em vista tais aspectos peculiares da Linguagem das Fórmulas identificados pela equipe de desenvolvimento, algumas propostas de melhorias foram sugeridas, planejadas e algumas já implantadas. A apresentação dos resultados desta pesquisa finaliza então com a apresentação, o comentário e a avaliação dos benefícios efetivos (ou ao menos esperados) por conta das referidas propostas de melhorias (Figura 8 – Melhorias para Linguagem das Fórmulas) (Figura 9 – Propostas de Melhorias e seus Benefícios para a Linguagem das Fórmulas).

Vale ressaltar que o projeto de utilização de macro comandos representa um avanço dentro da lógica da própria linguagem, pois o uso de sub-rotinas nas fórmulas permite a modularização dos cálculos, facilitando seu reuso. A repercussão dessa mudança representa principalmente melhoras em clareza e manutenibilidade.

(a)

```
ROT-COM-205.
***
***  JUROS DE ADIMPLENCIA
***
     MOVE   0,5          TO W05-COMP.
***
     IF     L-MEM (303)  = 01 OR 02 OR 03
            GO TO        ROT-COM-205-LINEAR
     ELSE
     IF     L-MEM (303)  = 04 OR 05 OR 06
            GO TO        ROT-COM-205-EXPO
     ELSE
     IF     L-MEM (303)  = 10 OR 11 OR 12
            GO TO        ROT-COM-205-TJLP
```

(b)

```
303   CalcDiasJAd
01    Linear360/360
02    Linear365/365
03    Linear366/366
04    Exponencial_360/360
05    Exponencial_365/360
06    Exponencial_365/365
07    JurosBNDES_operacoes635
08    Linear_360carência_365retorno
09    Linear_365/360
10    Taxa_Juros_mais_TJLP_dias360_coef360
11    Taxa_Juros_mais_TJLP_dias365_coef365
12    Taxa_Juros_mais_TJLP_dias365_coef360
```

Figura 8. Melhorias para Linguagem das Fórmulas: (a) Ambiente de Testes
(b) Macrocomandos e Parâmetros. Fonte: coleta de dados.

Melhorias	**Benefícios**	**Situação**
Integração da documentação das alterações à documentação do código	**Legibilidade**	Melhorias simples já implementadas
Automação da comparação entre fórmula implementada e versão documentada	**Confiabilidade**	
Carga em lote de uma fórmula inteira, eliminando a digitação passo a passo	**Eficiência**	
Eventos para disparo de fórmulas quando há mudança de cotações de moedas	**Expressividade**	
Flexibilização de algumas fórmulas segundo alguns poucos parâmetros	**Modularidade**	
Reformulação da sistemática de uso do ambiente de Teste de Fórmulas o integração do sistema financeiro em *mainframe* com interface gráfica o automação de etapas para efetivação de testes de fórmulas o áreas de teste independentes para cada desenvolvedor	**Confiabilidade** **Eficiência** **Custo**	Melhoria complexa já implementada (Figura 8 (a) *versus* Figura 6)
Reformulação da linguagem pela criação de macrocomandos em COBOL o macrocomandos parametrizados para cálculos específicos (p.ex.:juros,multa) o ampliação dos descritores de operações, incluindo todos os parâmetros o alterações dos descritores em lotes para cada tipo de grupo de fórmulas o documentação dos cálculos legível nos parâmetros nos descritores o uso de linguagem de alto nível, documentação facilitada, comentários no código, controle de versões, registro de auditoria de atualizações	**Legibilidade** **Redigibilidade** **Simplicidade** **Expressividade** **Modularidade** **Eficiência** **Custo**	Melhoria complexa apenas especificada (Figura 8 (b) *versus* Figura 4 e Figura 5)

Figura 9. Propostas de Melhorias e seus Benefícios para a Linguagem das Fórmulas.
Fonte: elaborado pela autora com base na coleta e análise de dados.

Algumas dessas melhorias já foram implantadas ou projetadas na empresa, incluindo a reforma do ambiente de testes e a criação de macro comandos para cálculos. A avaliação dessas propostas identificou a melhora (ou expectativa de melhora) de vários critérios de qualidade de linguagens de programação.

5 Conclusões

Com base na análise dos resultados desta pesquisa, é possível concluir que está em uso uma linguagem de programação peculiar. O próprio critério de seleção do caso a ser estudado neste trabalho identificou um tipo de processo de desenvolvimento de produto (no caso, de produtos financeiros, que envolvem cálculos em sistemas informatizados) que utiliza uma Linguagem de Programação potencialmente desconhecida de engenheiros de produto, desenvolvedores de *software* ou mesmo gerentes financeiros e pesquisadores acadêmicos.

Comparando-se a caracterização da linguagem com a classificação das gerações de linguagens de programação apresentada no Referencial Teórico, percebe-se que ela se

enquadra entre as linguagens de 1ª geração. Isso pode ser verificado pela constatação de que a linguagem é praticamente de baixo nível.

Quanto à classificação dos critérios de qualidade de linguagens de programação, a linguagem se enquadra no Referencial Teórico de maneira exemplar, pois configura um exemplo de linguagem que contraria, ou viola, **todos** os critérios de qualidade elencados na literatura. Configura, desta forma, um caso de ensino peculiar para fins de estudo e como provocação para a proposta de melhorias pelos estudantes do tema.

Considerando os resultados apresentados, é possível afirmar que foi alcançado o objetivo desta pesquisa, de analisar a qualidade da Linguagem de Programação das Fórmulas, que implementa os cálculos dos produtos financeiros do sistema legado de uma organização. A análise dos resultados do Estudo de Caso realizado permitiu entender em detalhes a dinâmica da Linguagem de Programação das Fórmulas e de seu funcionamento. Também permitiu avaliar sua qualidade em termos dos critérios Legibilidade, Redigibilidade, Simplicidade, Expressividade, Modularidade, Confiabilidade e Eficiência (Classificação segundo Critérios de Qualidade), os quais, em sua totalidade, são violados.

Para superar tais problemas, foram apresentadas e avaliadas novas propostas de melhorias, e a avaliação dessas propostas identificou a melhora (ou expectativa de melhora) de vários critérios de qualidade de linguagens de programação. Entende-se que tais medidas têm potencial de contribuição prática para qualificar todo o processo de desenvolvimento de produtos utilizado na empresa. Entretanto, somente a médio ou longo prazo, depois de um treinamento formal e de um tempo de experiência inicial, deverá ser possível – e fica aqui uma sugestão de pesquisa futura – avaliar a percepção da efetiva melhora de qualidade proveniente da adoção do novo processo, do novo sistema e da sua nova linguagem.

A principal contribuição deste trabalho reside na exploração de um caso de utilização de uma linguagem de programação, aplicada ao processo de desenvolvimento de produtos financeiros, extremamente peculiar e inovadora – por isso potencialmente desconhecida dos gerentes, engenheiros e desenvolvedores na comunidade científica. Os achados deste trabalho de pesquisa mostram a contribuição de apresentar à comunidade uma forma de organização da produção inusitada para os padrões atuais, além de ser exemplar, no sentido de exemplificação propriamente dita, para fins de estudos e proposta de melhorias.

6 Recomendações

A descrição do ambiente de desenvolvimento e das condições de teste das “fórmulas” elaboradas usando a linguagem demonstra um processo artesanal de desenvolvimento de produto, associado a uma série de problemas que lhe reduzem a qualidade. Para abordar tais problemas, foram propostas mudanças, que melhorem todos os critérios de qualidade.

Na etapa de avaliação dos resultados da pesquisa, os achados apontam que uma melhoria realmente efetiva parece ser a substituição da linguagem de programação utilizada por outra mais aprimorada. De acordo com o referencial teórico, essa linguagem deverá ser de outra geração, plenamente integrada à parametrização do sistema de informação utilizado. Supõe-se, ainda de acordo com o referencial teórico, que essa medida deve qualificar todo o processo de desenvolvimento de produtos utilizado na empresa.

Entretanto, percebe-se uma condição de contorno em relação à implantação de melhorias neste aspecto central do processo de desenvolvimento de produto da empresa pesquisada. A condição refere-se à gestão organizacional, pois é provável que a reformulação técnica demande uma mudança cultural substancial, tanto para os desenvolvedores, quanto para os clientes. É possível que somente após treinamentos formais e de uma experiência inicial, de médio (ou até mesmo longo) prazo, seja viável avaliar a percepção das pessoas, clientes e desenvolvedores. Complementando a teoria, acredita-se que esses são os critérios que permitirão verificar a efetiva melhora de qualidade resultante da adoção de uma linguagem de programação diferente.

Referências Bibliográficas

ABPC – Associação Brasileira de Profissionais COBOL. *Mas... COBOL?*. 2000. Disponível em: http://urlm.com.br/www.abpc.com.br Acesso em: 11 abr.2018.

BARBOSA, Pedro Luis Saraiva; CÂNDIDO, Adriano Lima. Diferenças entre Engenharia Reversa e Reengenharia nos Sistemas de Informação. *Interfaces*. v. 4, n. 13, 2016.

BASSELLIER, G.; BENBASAT, I. Business Competence of Information Technology Professionals: Conceptual Development and Influence on IT-Business Partnerships. *MIS Quarterly*, v. 28, n. 4, p. 673-694, 2004.

BAUER, M.W.; GASKELL, G. (org.). *Pesquisa qualitativa com texto, imagem e som.* 5.ed. Petrópolis: Vozes, 2015.

CINTRA, Rafael; VENDRAMEL, Wilson. Padrões de Migração de Sistemas Legados para Arquitetura Baseada em Microsserviços. *Revista de Sistemas e Computação*. v.9, n.1, 2019.

CSIS – Department of Computer Science and Information Systems. *COBOL Programming Course*. 2010. Disponível em: http://www.csis.ul.ie/cobol/course/ Acesso em: 11 abr. 2018.

FREITAS, Bruno Chaves de. *Modernização de sistemas legados para disponibilização em dispositivos móveis com arquitetura baseada em microservices*. Dissertação. Mestrado em Ciência da Computação. Universidade Federal de Pernambuco. 2017.

KNUTH, D. *The Art of Computer Programming*. 3.ed. Reading, MA: Addison-Wesley, 2011.

MALANOVICZ, Aline Vieira. A implantação do sistema (des)integrado. *Revista Brasileira de Gestão e Inovação*. v. 8, n.2, jan.-abr. 2021. DOI: 10.18226/23190639.v8n2.08

PRESSMAN, R. *Software Engineering*: a practitioner's approach. 8.ed. McGraw-Hill, 2014.

PRICE, A.M.; TOSCANI, S. *Implementação de Linguagens de Programação*. 3.ed. Porto Alegre: Bookman, 2008.

SCRIPTOL. *List of Hello World Programs in 200 Programming Languages*. 17 mar. 2018. Disponível em: http://www.scriptol.com/programming/hello-world.php. Acesso em: 11 abr. 2018.

SEBESTA, R.W. *Conceitos de Linguagens de Programação*. 9.ed. Porto Alegre: Bookman, 2011.

SOMMERVILLE, I. *Engenharia de software*. 8.ed. São Paulo: Pearson Addison-Wesley, 2016.

TAVARES, E.; THIRY-CHERQUES, H.R. A interação entre sistemas de informação e o trabalho no setor bancário no Brasil. São Paulo: *Anais*... ENANPAD, 2009.

VALLE, N.A.Y.; SOUSA, R.C.; UNSONST, V.A.F.; ANQUETIL, N.A. Critérios de avaliação para reengenharia de sistemas legados. In: WMSWM. 2. *Anais*. SBC. Manaus. 2005.
WEBER, Raul Fernando. *Fundamentos de Arquitetura de Computadores*. 4.ed. Porto Alegre: Bookman, 2012.
YIN, R.K. *Estudo de Caso: Planejamento e Métodos*. 5 ed. São Paulo: Bookman, 2015.