CENTRO UNIVERSITÁRIO LUTERANO DE PALMAS
ULBRA
Recredenciado pela Portaria Ministerial nº 3.607, de 17/10/05, D.O.U. nº 202, de 20/10/2005
ASSOCIAÇÃO EDUCACIONAL LUTERANA DO BRASIL

Victor Eduardo de Sousa Silva

CRIZON 2.0: UMA FERRAMENTA PARA ESTIMATIVA DE PONTOS DE FUNÇÃO USANDO DIAGRAMA DE SEQUÊNCIA DA UML E A INTERPRETAÇÃO DO SCRIPT DDL DA SQL

Palmas - TO
2015

Victor Eduardo de Sousa Silva

CRIZON 2.0: UMA FERRAMENTA PARA ESTIMATIVA DE PONTOS DE FUNÇÃO USANDO DIAGRAMA DE SEQUÊNCIA DA UML E A INTERPRETAÇÃO DO SCRIPT DDL SQL

Projeto apresentado como requisito parcial para aprovação na disciplina de Trabalho de Conclusão de Curso II (TCC II) do curso de Sistemas de Informação pelo Centro Universitário Luterano de Palmas (CEULP/ULRA).

Orientador: Prof. Mestre Fernando Luiz de Oliveira.

Palmas - TO
2015

Victor Eduardo de Sousa Silva

CRIZON 2.0 – UMA FERRAMENTA PARA ESTIMATIVA DE PONTOS DE FUNÇÃO USANDO DIAGRAMA DE SEQUÊNCIA DA UML E A INTERPRETAÇÃO DO SCRIPT DDL SQL

Trabalho de Conclusão de Curso (TCC) elaborado e apresentado como requisito parcial para obtenção do título de bacharel em Sistemas de Informação pelo Centro Universitário Luterano de Palmas (CEULP/ULBRA).

Orientador: Prof. M.Sc. Fernando Luiz de Oliveira.

Palmas - TO
2015

Victor Eduardo de Sousa Silva

CRIZON 2.0 – UMA FERRAMENTA PARA ESTIMATIVA DE PONTOS DE FUNÇÃO USANDO DIAGRAMA DE SEQUÊNCIA DA UML E A INTERPRETAÇÃO DO SCRIPT DDL SQL

Trabalho de Conclusão de Curso (TCC) elaborado e apresentado como requisito parcial para obtenção do título de bacharel em Sistemas de Informação pelo Centro Universitário Luterano de Palmas (CEULP/ULBRA).

Orientador: Prof. M.Sc. Fernando Luiz de Oliveira.

Aprovado em xxxxxxx de 2015.

BANCA EXAMINADORA

Prof. M.Sc. Fernando Luiz de Oliveira

Centro Universitário Luterano de Palmas

Prof. M.Sc. Cristina D'Ornellas Filipakis Souza

Centro Universitário Luterano de Palmas

Prof. M.Sc. Parcilene Fernandes de Brito

Centro Universitário Luterano de Palmas

Palmas - TO
2015

Em algum lugar, pra relaxar
Eu vou pedir pros anjos cantarem por mim
Pra quem tem fé
A vida nunca tem fim
Não tem fim
É

Se você não aceita o conselho, te respeito
Resolveu seguir, ir atrás, cara e coragem
Só que você sai em desvantagem se você não tem fé
Se você não tem fé

Te mostro um trecho, uma passagem de um livro antigo
Pra te provar e mostrar que a vida é linda
Dura, sofrida, carente em qualquer continente
Mas boa de se viver em qualquer lugar
É

Volte a brilhar, volte a brilhar
Um vinho, um pão e uma reza
Uma lua e um sol, sua vida, portas abertas

Anjos (Pra Quem Tem Fé) – O Rappa

AGRADECIMENTOS

Agradeço primeiramente a Deus, o todo poderoso, por ter me iluminado, guiado e dado força em todos os momentos da minha vida, desde no mergulho dos meus sonhos até os momentos mais difíceis da minha vida, que se inicia a cada dia.

À minha amada e idolatrada mãe, guerreira e minha fortaleza, que está comigo por onde eu ando, agradeço por ter acreditado e confiado em mim e sempre me incentivava e me dava uma luz em meio a tanta escuridão.

Ao meu pai, por mesmo do jeito dele ter me apoiado e acreditado na minha capacidade por todo esse tempo.

À minha namorada Ana Cecília Pott Cavalcante, por me aturar e incentivar quando eu sempre precisei, estando comigo por toda essa caminhada.

Aos meus colegas, por ter aturado o meu jeito "CAVALO" de ser e ter conseguido conviver e "aceitar" o meu jeito.

Ao meu orientador Fernando Luiz de Oliveira, por ter me guiado nesses 12 meses de muito aprendizado, erros e vitórias, bem como ter sido muito importante para minha formação, assim como os demais professores.

À minha mãe de Faculdade Cristina D'Ornellas Filipakis, por também ter me guiado e me ensinado a aprender com meus erros, além de ter sido também muito importante para minha formação.

Ao meu pai de Faculdade Fabiano Fagundes, por ter sempre me incentivado e aconselhado quando precisei, além das risadas e bullings.

Ao meu tio de Faculdade o "Branquinho", pelos bullings e aprendizados recebidos com suas belas palavras de um ilustre Doutor Ed.

À todos os professores do curso de Sistemas de Informação, pelo convívio, brincadeiras, abraços, risadas, metáforas, viagens em meio do emaranhado de códigos, discursões e conversas e risadas indesejadas pela professora Madianita, que fizeram parte de toda essa caminhada. Jamais esquecerei todos os momentos vividos nesta segunda Família que ganhei por esses 54 meses e que levarei por toda vida.

RESUMO

Para facilitar a determinação de um valor real para o software, Allan Albrecht elaborou a técnica Análise de Pontos de Função, que visa mensurar o valor funcional de um software independentemente das tecnologias utilizadas na implementação. A partir dessa técnica, a equipe ou gerente do projeto possui a possibilidade de determinar a quantidade de Pontos de Função de um projeto, assim, determinando um valor real do custo de desenvolvimento do software. Neste contexto, o presente trabalho discorre sobre o desenvolvimento de uma ferramenta para estimativa de Pontos de Função usando diagrama de sequência da UML e a interpretação do script DDL da SQL; tendo como objetivo auxiliar e facilitar o processo de análise e cálculo de Pontos de Função, de forma que seja mitigado o tempo da análise e eliminando a necessidade de possuir um conhecimento abrangente sobre a técnica de Pontos de Função.

PALAVRAS-CHAVES: análise de Pontos de Função, métrica de software, ferramenta de métrica de software

LISTA DE FIGURAS

LISTA DE TABELA

LISTA DE ABREVIATURAS

AIE	Arquivo de Interface Externa
ALI	Arquivo Lógico Interno
API	*Application Programming Interface*
ANSI	*American National Standards Institute*
APF	Análise de Pontos de Função
AR	Arquivo Referenciado
CE	Consulta Externa
CFPS	*Certified Function Point Specialist*
DDL	*Data Definition Language*
EE	Entrada Externa
IFPUG	*International Function Point Users Group*
ISO	*International Organization for Standardization*
MDA	*Model Driven Architecture*
PF	Pontos de Função
SE	Saída Externa
SGBD	Sistema de Gerenciamento de Banco de Dados
SQL	*Structured Query Language*
TCL	*Transact Control Language*
TD	Tipo de Dado
TR	Tipo de Registro
UML	*Unified Modeling Language*

SUMÁRIO

1. Introdução

A gerência de projetos tornou-se de suma importância no desenvolvimento de software, pois proporcionou maior organização de prazo, custo, tempo e manutenção da qualidade do projeto. Segundo uma das maiores associações de gestores de software, a *Project Management Institute* – PMI, "o gerenciamento de projetos é a aplicação de conhecimentos, habilidades e técnicas para a execução do projeto de forma efetiva e eficaz" (PMI *online*, 2014). Para que essa união entre conhecimento e habilidades possibilitem um bom gerenciamento, é necessário que o gestor se mantenha sempre atualizado e busque aprimoramento, amadurecimento e aplicação de atividades relacionadas à gerência, pois estas atividades influenciam diretamente na qualidade, e certamente, no sucesso de projetos de software.

As atividades listadas acima são direcionadas ao gerente de projetos, que tem como papel coordenar, monitorar e gerenciar o desenvolvimento do projeto. Além disso, o gerente de projeto tem o papel de mensurar o tamanho do software a ser desenvolvido, com o intuito de determinar um valor quantitativo a ser cobrado para desenvolver o software. Esse valor é obtido a partir de quatro fatores: o tamanho do produto a ser desenvolvido, o esforço a ser empregado para sua implementação, a duração do projeto e o custo gerado pela organização para o desenvolvimento.

No que se refere à determinação de um valor real para o desenvolvimento de software, há certas preocupações, pois não existe um padrão ou regra a ser utilizado para determinar ou alcançar esse valor. Entretanto, algumas empresas utilizam técnicas para auxiliar nessa medição de custo, que tentam buscar somar os esforços, máquinas utilizadas, tempo gasto, honorários, licenças e lucro para determinar um valor final a ser cobrado ao cliente. Existem várias técnicas para mensurar o tamanho do software, dentre elas: a métrica por Linhas de Códigos (LC), que

utiliza a contagem de linha de códigos para mensurar o valor; Pontos por Casos de Uso (PCU), que utiliza a análise do artefato caso de Uso como métrica; e Análise de Pontos de Função (APF), que utiliza a medida funcional do software, analisando sempre o que é visível para o usuário. Cada técnica possui seus conceitos, características e peculiaridades que serão necessários para realizar o processo de mensuração.

Para facilitar a determinação de um valor real para o software, Allan Albrecht elaborou a técnica Análise de Pontos de Função (SOBRE ANÁLISE..., [s.d.]), que visa mensurar o valor funcional de um software independentemente das tecnologias utilizadas na implementação. A partir dessa técnica, a equipe ou gerente do projeto tem a possibilidade de determinar a quantidade de Pontos de Função de um projeto, assim, pode-se obter um valor real do custo de desenvolvimento do software.

A técnica divide o software em dois tipos de análise, que são: Grupo de Dados, que representa a estrutura da base de dados; e Transações, que consistem no mapeamento das funcionalidades ou ações realizadas pelo software. Cada etapa, ao final, representará um valor em Pontos de Função, que será somado para definir o tamanho funcional do software como um todo.

O problema relacionado a esta técnica é que sua aplicação envolve tarefas manuais, necessitando sempre um grande gasto de tempo para realizar a análise funcional do software a partir da visão do usuário. Esse grande gasto de tempo reflete na análise de visões diferentes do software na qual, primeiramente, é analisada a estrutura dos dados presentes e, posteriormente, as funcionalidades ou transações existentes, levando sempre em consideração a visão do usuário.

Assim, para facilitar a utilização da técnica e a tornar mais automática, o presente trabalho discorre sobre o desenvolvimento de três módulos de um sistema que possibilita a mensuração do tamanho funcional dos softwares a partir da técnica de análise de Pontos de

Função, utilizando como entrada os Diagramas de Sequências e o *script* DDL do banco de dados. Estes dois artefatos foram escolhidos por possibilitar a identificação dos Grupos de Dados e das Transações envolvidas no software. No caso, para identificar os Grupos de Dados foi utilizado o *script* DDL, que é composto por códigos da linguagem SQL que representam a estrutura da base de dados do software. Esse *script* é utilizado para construção de bancos de dados que utilizam a sintaxe SQL, representando sua estrutura em forma escrita. Já para identificar as Transações existentes no sistema, foi realizada a análise dos Diagramas de Sequência criados pelo usuário. Esse diagrama pertence a linguagem de modelagem da *Unified Modeling Language* (UML), que descreve a interação de um ator com a interface de usuário e interface com todas as camadas do sistema.

Por fim, essas funcionalidades foram unidas à ferramenta CRIZON, desenvolvida por Rodrigues (2012), que possibilita a criação de artefatos de modelagem para software e, com as novas funcionalidades, possibilita também a mensuração do tamanho funcional a partir da análise do *script* DDL do banco de dados e diagramas de sequência construídos pelo usuário.

2. Referencial teórico

Nessa seção serão apresentados os principais conceitos relacionados a este trabalho, que se fazem necessários para o desenvolvimento das funcionalidades e módulos da ferramenta CRIZON, para torná-la capaz de determinar o tamanho funcional de softwares, a partir da análise de produtos da modelagem de software.

Os conceitos relacionados são: Análise de Pontos de Função segundo abordagem do IFPUG, que foi utilizada para realizar a estimativa de software a partir dos artefatos; *script Data Definition Language* SQL, que é o *script* que determina a estrutura do banco de dados interpretará; Diagrama de Sequência da UML, artefato que ilustra uma interação composta por instâncias de classes, atores, componentes e/ou subsistemas, que foi utilizado para identificar as variáveis utilizadas na interação a partir da sua modelagem realizada pelo usuário.

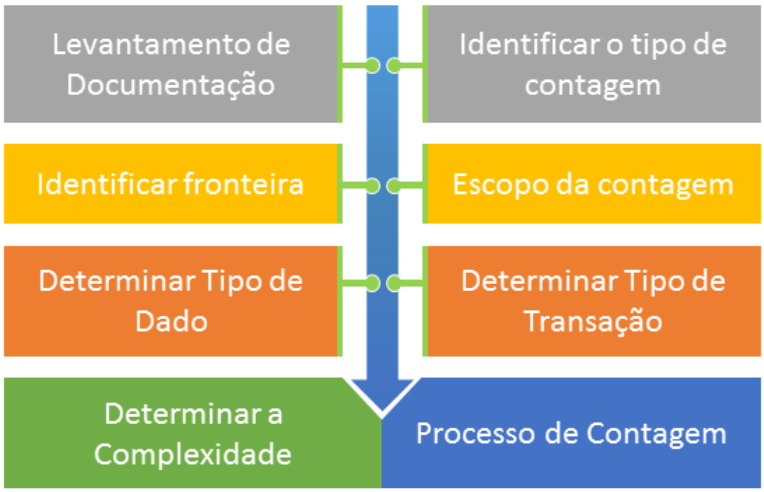
2.1. Análise de Pontos de Função

É uma técnica que foi proposta ao público nos anos 79 por Allan J. Albrecht (SOBRE ANÁLISE..., [s.d.]), que trabalhava como analista de sistemas na IBM e precisava apontar fatores críticos que pudessem ser utilizados para determinar o tamanho de um sistema. “A técnica de Análise de Pontos de Função mede uma aplicação através das funções desempenhadas para/e por solicitação do usuário final” (BRAGA, 1996, p.3), ou seja, mensura as funcionalidades de uma aplicação a partir do ponto de vista do usuário. Por isto, a APF é independente de qualquer tecnologia que possa ser utilizada, podendo ser analisada por pessoas distintas em que se pretende ter uma mensuração de Pontos de Função equivalente. Essa técnica é utilizada para analisar software orientado a objetos.

A Análise de Pontos de Função cria uma unidade de medida de tamanho funcional de software, do mesmo modo que o m² mede a área de uma casa e, a partir de um valor por m²,

determina o seu valor, por exemplo, a APF define a unidade Pontos de Função (PF) que reflete o tamanho funcional do software. Além disso, a técnica é mantida pelo *International Function Point Users Group* (IFPUG), em português Grupo Internacional de Usuários de Pontos de Função, que se encontra na versão 4.2.1 publicada em 2005 no IFPUG, que também é responsável pelo exame de certificação de especialistas em contagem de PF, denominada *Certified Function Point Specialist* (CFPS). A Figura 1 ilustra o processo da Análise de Pontos de Função.

Figura 1 - Processo de Contagem de Pontos de Função



Fonte: Próprio autor

O processo de contagem, ilustrado pela Figura 1, contém 8 etapas que precisam ser realizadas criteriosamente para que, ao final, seja possível determinar a quantidade de Pontos de Função do sistema sob análise. As etapas da análise são: levantamento da documentação, na qual se faz a busca de artefatos que auxiliem o processo de contagem; identificar o tipo de contagem, onde é determinado qual tipo de contagem será realizada para mensurar o tamanho funcional; identificar fronteira, onde é definido quais são as fronteiras da aplicação, identificando quais dados pertencem ou não à aplicação; escopo da contagem, determinando quais funcionalidades/módulos serão incluídos ou não no processo de contagem; determinar tipo de dado, é a etapa de identificar e definir quais são os tipos de dados do software sob análise; determinar tipo de transação, que consiste em identificar e definir quais são as

transações realizadas pelo software; determinar complexidade, na qual é feita a análise dos grupos de dados e inferência das suas complexidades de acordo com a técnica; e, por fim, processo de contagem, etapa na qual é feito o processo que analisa e determina a quantidade de Pontos de Função a partir das inferências realizadas nas etapas anteriores.

As próximas seções abordarão as etapas necessárias para se realizar a contagem de forma mais detalhada.

2.1.1. Definir propósito

Ao realizar a contagem de Pontos de Função é preciso definir um propósito. Isto porque, segundo Vazquez, Simões e Albert (2010, p.52), "uma contagem de Pontos de Função não é um fim em si mesmo; sempre há uma motivação maior, o seu propósito". É este propósito que leva o usuário a realizar a análise, como, por exemplo: contagem de PF para medir o tamanho do software que será desenvolvido em prol de determinar um valor para o cliente. Ou então, mensurar para determinar qual será o preço estimado do desenvolvimento de novos módulos para a empresa. O objetivo de definir o propósito é prover um maior esclarecimento sobre a contagem, que segundo Vazquez, Simões e Albert (2010, p.53) pode ajudar a:

- Definir o tipo de contagem;
- Determinar o nível de detalhe da contagem;
- Definir algumas premissas para o processo;
- Definir o escopo da contagem, e
- Definir a fronteira da aplicação;

2.1.2. Levantamento de artefatos de software

Para realizar a contagem de Pontos de Função é preciso possuir um conhecimento sobre o software a ser analisado, podendo também ter auxílio a partir de alguns artefatos da

modelagem de software, caso exista. A documentação pode ajudar a definir e identificar itens mais rapidamente ou então ajudar o usuário que não conhece muito o software realizar o processo de contagem de Pontos de Função.

2.1.3. Identificar tipo de contagem

Essa etapa consiste em identificar o tipo de contagem que será realizado, de acordo com o software a ser analisado, sendo dividido em três tipos: projeto em desenvolvimento, ou seja, quando o software a ser analisado está em fase de desenvolvimento; projeto de melhoria, quando a contagem é realizada em um software já desenvolvido, mas que será melhorado; e, aplicação pronta, que é a contagem realizada a partir de um software instalado e pronto para uso.

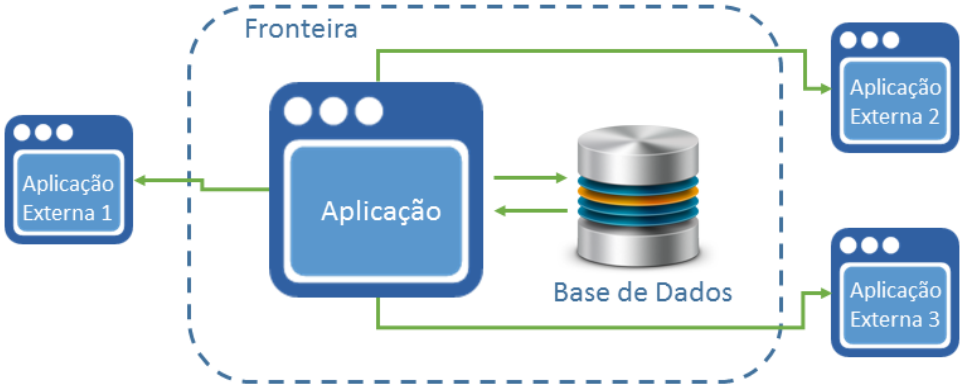
2.1.4. Identificar fronteira

A fronteira da aplicação é identificada a partir dos relacionamentos do software em análise com seu exterior, identificando os dados e processos que são mantidos pelas aplicações exteriores e utilizados pelo software em questão. A identificação desta fronteira possibilita determinar algumas características que auxiliaram na identificação das demais etapas da técnica. HAZAN (2000) apresenta algumas características providas desta identificação, são elas:

- Define as aplicações externas a aplicação;
- Auxilia na identificação dos Arquivos de Interface Externas, que não são mantidos pela aplicação mas fazem parte da aplicação;
- Define a interface conceitual entre a aplicação interna e externa;

Além disso, o processo de definição de grupo de dados utiliza a identificação da fronteira para determinar o tipo do grupo, que influencia diretamente na complexidade e Pontos de Função do mesmo. A Figura 2 ilustra a Fronteira da Aplicação.

Figura 2 - Fronteira da Aplicação



Fonte: Próprio autor

A Fronteira da Aplicação, ilustrada pela Figura 2, "separa o projeto ou aplicação que está sendo contado de aplicações externas" (BRAGA, 1996, p.22), ou seja, delimita e identifica outros sistemas em que os dados não são mantidos pela aplicação, e sim pelas aplicações externas. Um exemplo é quando uma aplicação armazena a latitude e longitude do mapa, que é disponibilizada pela API do Google Maps, onde os dados (latitude e longitude) são armazenados na aplicação; entretanto, as demais informações como: endereço, quadra ou cidade; são mantidas e atualizadas pela aplicação exterior. Neste exemplo, a aplicação do Google Maps é externa e somente fornece dois dados que serão armazenados na aplicação, que identificará no mapa a localidade.

2.1.5. Escopo da contagem

O escopo consiste em "definir quais funções serão incluídas na contagem, se ela abrangerá um ou mais sistemas ou apenas parte de um sistema" (VAZQUEZ, SIMÕES e ALBERT, 2010, p.57). Nesta etapa são listados quais módulos ou partes do sistema serão

incluídos na contagem e observar atentamente o processo de contagem, pois poderá ocorrer do usuário realizar a contagem de uma funcionalidade da aplicação que não deveria ser mensurada.

2.1.6. Determinar Grupo de Dados

Esta etapa consiste na identificação e determinação dos grupos de dados que estão presentes no software, que segundo BRAGA (1996, p.26) "representam as funcionalidades providas aos usuários através de dados internos ou externos à aplicação". Um grupo de dado é um conjunto de Tipo de Dados (TD) que representam um mesmo objeto, ou seja, são variáveis de uma classe que são vistas pelo usuário. Um dado é classificado como Tipo de Dado (TD) quando o seu conteúdo é "reconhecido pelo usuário como único, não repetido, mantido por um Arquivo Lógico Interno (ALI) ou recuperado de um Arquivo de Interface Externa (AIE) ou de um ALI" (ANDRADE *online*, 2004, p.14). Dessa forma, por não ser visível para o usuário e não ser classificada como um TD, a chave primária é classificada como inválida para a contagem. Já a chave estrangeira, apesar de não se encaixar na classificação de um Tipo de Dado, por também não ser visível pelo usuário, deve ser contado como uma unidade de TD. Isto porque este atributo não é visível mais referencia um relacionamento com outra tabela do conjunto de dados.

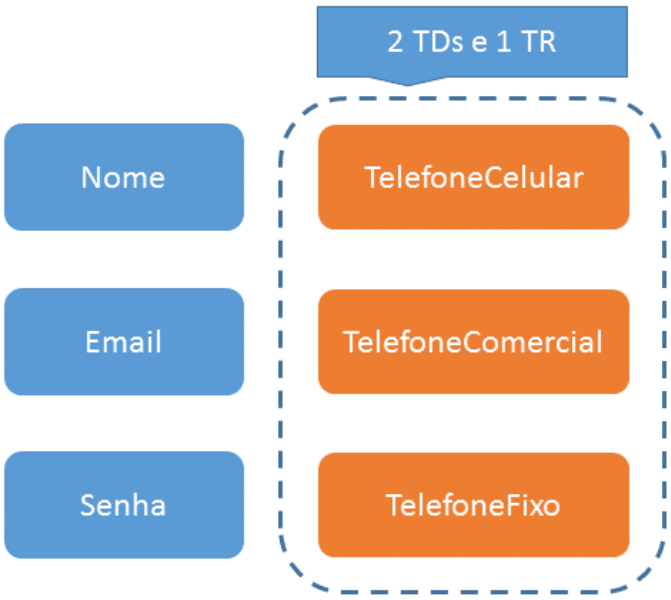
Os Grupos de Dados são classificados pela técnica de Análise de Pontos de Função em dois tipos. São eles:

- Arquivo Lógico Interno (ALI) - "um grupo identificável de dados relacionados logicamente que reside inteiramente dentro do limite aplicações e é mantida através de insumos externos" (LONGSTREET *online*, 2005, p.4). Um exemplo de ALI é uma determinada tabela do banco de dados que contêm dados (colunas) relevantes e visualizadas pelo usuário, podendo ser: nome, e-mail, telefone etc.

- Arquivo de Interface Externa (AIE) - é um grupo de dados mantidos e armazenados fora da aplicação, mas necessário para a aplicação. Segundo Farias et. al (*online*, 2004, p.10) "são dados que não sofrem manutenções pela aplicação que está sendo avaliada, dados identificados como necessidades de informação do usuário e dos dados que são contados como ALI para outra aplicação". Um exemplo de AIE é a localização de um estabelecimento mantido e armazenado pelo Google Maps. Dessa forma, as informações como cidade, endereço, rua e telefone, que serão importantes para os usuários, não serão mantidas na aplicação sob análise, e sim pela aplicação exterior.

Já o Tipo de Registro (TR) é um conjunto de Tipos de Dados que representa uma mesma informação, como, por exemplo, três tipos de dados (dia, mês e ano) que, juntos, representam uma data. A Figura 3 apresenta um Tipo de Registro.

Figura 3 - Tipo de Registro



Fonte: Próprio autor

A Figura 3 apresenta um exemplo de Tipo de Registro, onde são identificados três Tipos de Dados que referenciam uma informação, ou seja, o telefone (um Tipo de Registro) é utilizado para referenciar três informações (Tipos de Dados). Assim, define-se uma unidade de TD para determinar qual telefone se referencia (Celular, Comercial ou Fixo) e outro TD para identificar

qual o número do telefone. Outro exemplo é o endereço, que pode ser particionado em vários Tipos de Dados (endereço, rua, bairro etc.), mas representa um Tipo de Registro e dois Tipos de Dados.

2.1.7. Determinar Tipo de Transação

Segundo Reinaldo & Filipakis (*online*, 2009, p.3), "as funções do tipo transação representam as funcionalidades proporcionadas ao usuário para o processamento da aplicação", ou seja, denotam as operações disponíveis no sistema a partir da visão do usuário, exemplo: cadastro, gerar relatório etc. Do mesmo modo, as transações precisam ser classificadas, sendo elas:

- Entrada Externa (EE) – são transações lógicas que processam um ou mais grupos de dados de origem externa, cujo objetivo é realizar a manutenção de um ou mais ALI. Na visão de Alexander (apud IFPUG, 2004, p.5), "Entrada Externa (EE) é um processo elementar que processa dados ou controle de informação que vem de fora do aplicativo limite. A intenção principal de um EE é para manter um ou mais ALI e/ou para alterar o comportamento do sistema". Exemplos de transações do tipo EE são: inserção e alteração de um ou mais grupos de dados.
- Saída Externa (SE) – são transações lógicas que processam um ou mais grupos de dados de origem interna e exportados para fora da fronteira da aplicação, cujo objeto é apresentar os dados processados para o usuário. Porém, essa transação precisa conter uma lógica de processamento, onde, no mínimo, um cálculo matemático, fórmula ou criação de dados derivados precisam ser processados, de forma que, esses dados sejam providos de um ou mais ALI. Além disso, a transação SE deve manter um ou mais ALI e/ou alterar o comportamento do sistema (FARIAS et. al *online*, 2004).

- Consulta Externa (CE) – “visa apresentar informações para o usuário através da recuperação de dados ou informação de controle” (ANDRADE *online*, 2004, p.17), ou seja, são transações lógicas que processam um ou mais grupos de dados de origem interna e exportados para fora da fronteira da aplicação, do mesmo modo que as transações SE. Para *Eclipse Soluctions Gartner* (*online*, 2005, p.3) “a principal intenção é apresentar informações para um usuário através da recuperação de dados ou informações de controle de um ALI ou AIE”. Entretanto, as transações CE não possuem processamento lógico, ou seja, as informações exibidas por essa transação são recuperadas de um ou mais ALI ou AIE, sem nenhum cálculo, fórmula ou criação de dados derivados.

Após identificar e determinar os grupos de dados e tipos de transações existentes no software em análise, segue a análise determinando a complexidade dos mesmos, que será apresentada detalhadamente na seção seguinte.

2.1.8. Determinar Complexidade

Após identificar e definir os grupos de dados e tipos de transações que serão utilizados nesta etapa, é possível iniciar o processo de determinação da complexidade dos mesmos. Primeiramente, são analisados os grupos de dados que, por padrão, é determinado uma unidade TR para cada grupo e outro TR para cada tipo de registro existente. Em seguida, é somada a quantidade de dados classificados como Tipo de Dado, sempre levando em consideração as regras de dados que também são Tipo Registro, explanado anteriormente.

Assim, é possível determinar quantos TDs e TRs possui cada Grupo de Tipo de Dado, que podem ser classificados como ALI ou AIE. A partir dessa determinação, é possível especificar qual é a complexidade a partir da sua classificação. A Tabela 1 apresenta a complexidade funcional dos grupos de dados ALI e AIE.

Tabela 1 - Complexidade funcional dos grupos de dados ALI e AIE

Tipos de Dados (TD)			
Tipos de Registros (TR)	1 a 19	20 a 50	51 ou mais
1	Baixa	Baixa	Média
2 a 5	Baixa	Média	Alta
6 ou mais	Média	Alta	Alta

Fonte: ANDRADE *online*, 2004, p. 15

Essa complexidade, determinada a partir da quantidade de TDs e TRs em cada grupo de dados, será utilizada na próxima seção para realizar o processo de contagem de Pontos de Função.

Já os Tipos de Transação são determinados por outra regra, e utilizam outros fatores para determinar sua complexidade. Primeiramente, é preciso definir a quantidade de dados classificados como TD existentes na transação e a quantidade de Arquivos Referenciados (AR) que ela contém. Os AR são definidos a partir da quantidade de Grupos de Dados ALI ou AIE que são utilizados pela transação. Já os TDs são determinados pela quantidade de atributos que entram ou saem, a quantidade de mensagens de erros possíveis de serem exibidas e a quantidade de ação a ser tomada, por exemplo, um cálculo de variáveis.

Em uma visão orientada a objetos, os Tipos de Dados são as variáveis que entram ou saem da transação e os Arquivos Referenciados são as classes que são utilizadas na transação, que logicamente são as mesmas pertencentes às variáveis utilizadas. Por exemplo, a classe Pessoa possui três variáveis, são elas: Nome, Idade e E-mail; a transação “Consultar Pessoa” utiliza as três variáveis para apresentar a consulta, logo, a transação possui três Tipos de Dados, que são as variáveis “Nome”, “Idade” e “E-mail”, e um Arquivo Referenciado, que é a classe Pessoa.

As tabelas 2 e 3 apresentam a complexidade funcional das transações onde, a partir da quantidade de TD e AR, é possível identificar a complexidade existente de cada transação. Vale ressaltar que as transações SE e CE possuem a mesma tabela e a EE possui uma tabela separada.

Tabela 2 - Complexidade funcional das transações do tipo EE

Tipos de Dados (TD)			
Arquivos Referenciados (AR)	1 a 4	5 a 15	16 ou mais
0 a 1	Baixa	Baixa	Média
2	Baixa	Média	Alta
3 ou mais	Média	Alta	Alta

Fonte: ANDRADE *online*, p. 16, 2004

Tabela 3 - Complexidade funcional das transações do tipo SE e CE

Tipos de Dados (TD)			
Arquivos Referenciados (AR)	1 a 5	6 a 19	20 ou mais
0 a 1	Baixa	Baixa	Média
2 a 3	Baixa	Média	Alta
4 ou mais	Média	Alta	Alta

Fonte: ANDRADE *online*, p. 17-18, 2004

Com as complexidades das transações determinadas a partir das Tabela 2 e 3, finaliza o processo de determinação da complexidade e é iniciado o processo de contagem, explanado detalhadamente na próxima seção, que consiste no processo de determinação da quantidade de Pontos de Função que o sistema sob análise possuirá, a partir da complexidade dos grupos de dados e das transações determinados nesta seção.

2.1.9. Processo de Contagem

O Processo de Contagem ocorre somente quando a complexidade dos grupos de dados e transações for definidas, a partir das tabelas da seção anterior. Com isso, é preciso determinar

a quantidade de PF que cada grupo de dados e transações possui. A Tabela 4 apresenta a quantidade de Pontos de Função que um grupo de dados ou transação possui a partir da sua complexidade.

Tabela 4 - Pontos de Função a partir da complexidade

Tipo de Função	Baixa	Média	Alta
ALI	7 PF	10 PF	15 PF
AIE	5 PF	7 PF	10 PF
EE	3 PF	4 PF	6 PF
SE	4 PF	5 PF	7 PF
CE	3 PF	4 PF	6 PF

Fonte: REINALDO & FILIPAKIS online, 2009, p. 5

Após determinar as complexidades dos grupos de dados e transações, é possível determinar qual a quantidade de PF os mesmos possuem. Para isso, é preciso somar a quantidade de Pontos de Função de cada grupo. A Tabela 5 apresenta um exemplo da soma e contagem de Pontos de Função de um sistema fictício. É importante ressaltar que este exemplo refere-se a uma agenda telefônica, no qual o usuário poderá visualizar a lista de usuários, cadastrar um novo usuário ou removê-lo.

Tabela 5 - Exemplo de soma e contagem de Pontos de Função.

Nome	Tipo	TD	TR/AR	Complexidade	PF
Contato	ALI	4	1	Baixa	7 PF
Endereco	ALI	2	2	Baixa	7 PF
Cadastrar Contato	EE	4	1	Baixa	3 PF
Editar Contato	EE	3	1	Baixa	3 PF
Excluir Contato	EE	3	2	Baixa	3 PF
Listar Contatos	CE	3	3	Baixa	3 PF
Total					**26 PF**

Fonte: Próprio autor

De acordo com a Tabela 5, foram determinados os grupos de dados e suas características, são eles: Contato, que representa a tabela com os dados do usuário (Id, NomeContato, Email, IdEndereco, Telefone); Endereco, que contém os dados relacionais do relacionamento entre Contato e Endereco (Id, TipoLogradouro, Logradouro, Numero, Complemento e Casa). Como a entidade "Contato" é mantida no sistema, possui a classificação de ALI e contém quatro TDs (NomeContato, Email, IdEndereco e Telefone), pois a chave estrangeira "IdEndereco" não é visível pelo contato, entretanto, referencia uma dependência entre as tabelas e precisa ser contada. Além disso, a entidade "Endereco" possui somente dois TD, pois mesmo tendo cinco campos em sua estrutura é enquadrado como um tipo de registro, tendo um TD que referência o tipo de endereço e o outro TD referenciando o campo do endereço; tendo dois tipos de registro, um TR para a entidade como um todo e outro para o TR identificado. Posteriormente, foram determinadas as transações existentes e suas características, são elas: cadastrar contato, ação que cria um novo contato; editar contato, ação de alterar as informações de um contato existente; excluir contato, ação de eliminar o registro do contato existente; listar contatos, funcionalidade responsável por exibir a lista de contatos existentes. Ao final, é determinada a quantidade de Pontos de Função que o sistema possui, que só é possível após classificar e determinar os grupos de dados e transações. Essa classificação é realizada a partir das Tabelas 1, 2 e 3 que determinam a partir da quantidade de TD e TR/AR a complexidade dos grupos de dados/transações. Dessa forma, é possível determinar a partir da Tabela 4 a quantidade de Pontos de Função de cada grupo, que é determinado pelo tipo do grupo e sua complexidade.

As seções seguintes (2.2 e 2.3) apresentam conceitos relacionados à sintaxe do *script Date Definition Laguage* (DDL) SQL e o Diagrama de Sequência da *Unified Modeling Language* (UML), que serão utilizados para determinar a quantidade de Pontos de Função de um projeto de software. A utilização da interpretação do *script* DDL da SQL resultará na

definição dos grupos de dados existentes no projeto de software, na qual, ao relacionar com os diagramas de sequência cadastrados pelo usuário, será possível inferir quais variáveis serão visíveis ao usuário a partir dos parâmetros enviados e recebidos nas trocas de mensagens de cada diagrama. Assim, é possível definir os grupos e tipos de dados existentes no projeto de software. Por outro lado, os diagramas de sequência possibilitaram determinar os tipos de transações, que referem-se a uma das etapas para mensuração do tamanho funcional do projeto de software da técnica. Além disso, o diagrama de sequência também irá possibilitar a verificação da existência dos tipos de registros, realizada a partir de questionamentos ao usuário.

2.2. *Data Definition Language* (DDL) da SQL

Este trabalho utilizará a interpretação das instruções *Data Definition Language* (DDL) do banco de dados para auxiliar na mensuração dos Pontos de Função. Por isso, serão apresentados os principais conceitos relacionados a *Structured Query Language* (SQL) e a categoria de instrução *Data Definition Laguage* (DDL) que se fazem necessários para o entendimento deste trabalho. A seção 2.2.1 apresentará detalhadamente os conceitos relacionados a SQL.

2.2.1 *Structured Query Language* (SQL)

Criado no início de 1970 pela IBM, primeiramente chamada SEQUEL (*Structured English Query Language*), foi desenvolvida como forma de interface para banco de dados relacional. Posteriormente revisada em 1977, passou a ser chamada de Linguagem de Consulta Estruturada, em inglês, *Structured Query Language* (SQL), que logo mais tarde foi reconhecida e padronizada pelo Instituo Nacional Americano de Padrões (ANSI) juntamente com a

Organização Internacional de Padrões (ISO). Com isso, o SQL tornou-se uma linguagem padrão adotada para banco de dados relacional, tais como SQL Server, MySQL, Oracle, entre outros.

A partir dessa padronização, a linguagem passou a ter algumas características em destaque, tal como Puga, França e Goya (2013, p.170) citam:

- Portabilidade entre as plataformas que utilizam a SQL;
- Ao ocorrer a migração entre plataformas que utilizam a SQL, a adaptação dos profissionais é facilitada, reduzindo tempo e custo para treinamentos.
- Migração entre plataformas que utilizam a SQL não exige muitas mudanças.
- Padronização de instruções segue a mesma sintaxe e formato entre SGBD distintos.

A *Structured Query Language* é dividida em três categorias de instruções, que possibilitam manipular, controlar e definir todos os dados e estruturas do banco de dados relacional, são elas:

- *Data Definition Language* (DDL) – utilizada para definir a estrutura das tabelas.
- *Data Manipulation Language* (DML) – utilizada para manipular os dados contidos nas tabelas.
- *Transact Control Language* (TCL) – utilizada para controle das transações.

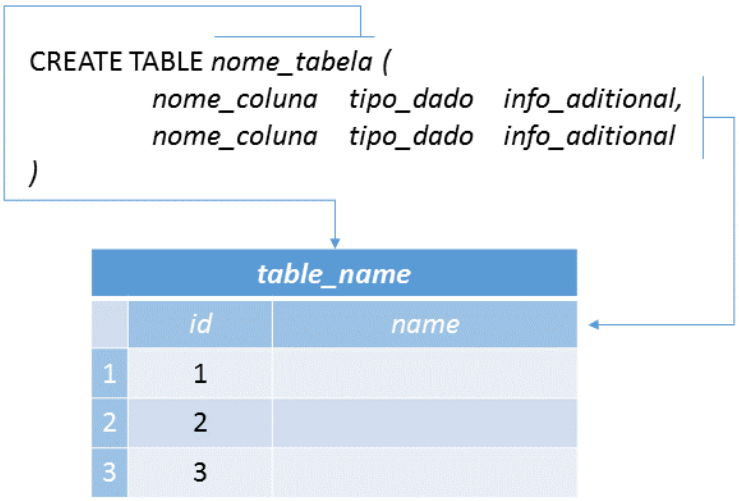
Este trabalho utilizará a categoria de instrução DDL para desenvolver um *script* que interprete essa instrução e auxilie na mensuração dos Pontos de Função. Por isso, serão abordadas detalhadamente as definições dessa categoria na próxima seção.

2.2.2 *Data Definition Language* (DDL)

A Linguagem de Definição de Dados, em inglês, *Data Description Language*, é composta pelos “comandos para definição das estruturas das tabelas e dos tipos de atributos respectivos” (MEDEIROS, 2013, p.96), mais conhecidos como os comandos *CREATE*, *DROP* e *ALTER*. Cada comando tem uma definição e função distinta, são elas:

- Instrução *Create Table* – responsável por definir a estrutura dos objetos do banco de dados, contendo respectivamente o nome da tabela, colunas, tipos de dado de cada coluna e informações complementares. A Figura 4 apresenta a estrutura dessa instrução.

Figura 4 - Estrutura da Instrução *Create Table*



Fonte: Próprio autor

A sintaxe utilizada para definir a instrução *Create Table*, ilustrada na Figura 4, é iniciada pela definição "*CREATE TABLE*", que representa a criação de uma nova tabela. Em seguida, é preciso descrever uma palavra que identifique os dados que serão armazenados na tabela, para que a mesma seja identificada de maneira fácil. Após definir o nome da tabela, é necessário identificar e descrever as colunas existentes nesta tabela; essa definição precisa estar entre parênteses, como ilustrado na figura acima. A sintaxe das colunas segue um padrão definido pela linguagem, que é: nome da coluna, que também precisa ser uma palavra; tipo de dado da coluna, que segue a definição dos tipos de acordo com a linguagem; e informações adicionais, que são expressas como campos nulos ou coluna chave.

- Instrução *Alter Table* – "modifica uma definição de tabela alterando, adicionando ou removendo colunas e restrições, reatribuindo e recriando partições, ou desabilitando ou habilitando restrições e gatilhos" (MSDN *online*, [s.d]). Cada um desses tipos de

operação possui uma sintaxe diferente e é definida pela linguagem, pois cada tipo de operação tem uma manipulação de operações e variáveis distintas. A Figura 5 ilustra a sintaxe básica dessa instrução.

Figura 5 - Estrutura da Instrução *Alter Table*

```
ALTER TABLE  nome_tabela  tipo_operação  nome_coluna  operação
```

Fonte: Próprio autor

A utilização dessa instrução é definida pela expressão "*ALTER TABLE*", seguida do nome da tabela que receberá a alteração, tipo de operação a ser realizada, nome das colunas que serão alteradas e operação a ser realizada. A Figura 6 ilustra um exemplo dessa instrução.

Figura 6 - Exemplo de uma Instrução *Alter Table*

```
ALTER TABLE  [dbo.][Usuario]  WITH CHECK ADD CONSTRAINT [FK_USUARIO_STATUS] FOREIGN KEY
([IdStatus])  REFERENCES [dbo].[Status]  ([Id])
```

Fonte: Próprio autor

A Figura 6 apresenta um exemplo de uma instrução *Alter Table* que define um relacionamento entre duas tabelas, criando uma chave estrangeira. Após a expressão "*ALTER TABLE*" é definido o nome da tabela que receberá a alteração. Em seguida, é definido o tipo de operação que será realizada, que nada mais é do que uma *Forent Key* ou chave estrangeira chamada de "FK_USUARIO_STATUS" que é referenciada pela coluna "IdStatus" da tabela "Usuario", referenciando a coluna "IdStatus" da tabela "Status" da base de dados.

Após apresentar os conceitos relacionados a Linguagem de Definição de Dados (DDL) da SQL, será apresentado na seção 2.3 os principais conceitos do Diagrama de Sequência da UML, que também será utilizado como entrada da contagem de Pontos de Função.

2.3. Diagrama de Sequência da UML

Este trabalho utilizará o diagrama de sequência para auxiliar na mensuração dos Pontos de Função. Por isso, serão apresentados os principais conceitos relacionados a estrutura e componentes existentes nesse diagrama, que pertence a linguagem de modelagem da *Unified Modeling Language* (UML).

Segundo Guedes (2011, p.193), este é um diagrama comportamental que procura determinar a sequência de eventos que ocorrem em um determinado processo, identificando quais mensagens devem ser disparadas entre os elementos envolvidos e em que ordem. Assim, esse diagrama busca relatar quais passos são necessários para realizar uma determinada ação. Esses passos vão desde identificar quais classes serão utilizadas para realizar a ação, até as mensagens e dados de retorno ao final de cada interação.

O diagrama de sequência faz parte da UML (*Unified Modeling Language*), que "é uma linguagem para especificação, documentação, visualização e desenvolvimento de sistemas orientados a objetos" (VARGAS, [s.d.], p.1). Essa linguagem de modelagem unificada foi lançada em 1996 por Booch, Rumbaugh e Jacobson, com o objetivo de unificar métodos de modelagens de software desenvolvidas pelos mesmos, criando um padrão de modelagem, eliminando alguns pontos fracos e evidenciando pontos fortes de cada método.

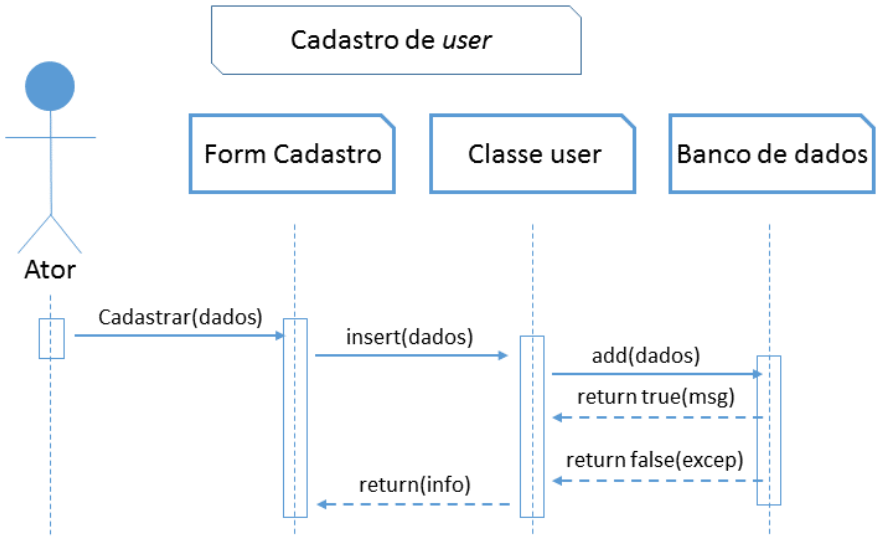
Stadzisz (*online*, 2002, p.29) ressalta que "um diagrama de sequência é um diagrama de objetos, ou seja, ele contém como primitiva principal um conjunto de objetos de diferentes classes", que, necessariamente, serão identificados e utilizados no decorrer das interações de um diagrama. Já Vargas ([s.d.], p.7) expressa que "o diagrama de sequência mostra a troca de mensagens entre diversos objetos, em uma situação específica e delimitada no tempo [...] colocando ênfase especial na ordem e nos momentos nos quais mensagens para os objetos são enviadas".

O Diagrama de Sequência da UML possibilita documentar um caso de uso especifico, já que um diagrama de sequência documenta as etapas necessárias para que determinada funcionalidade seja realizada por um ator, resultando na relação ator-funcionalidade, que é o objetivo do caso de uso.

> Um Diagrama de Sequência costuma identificar o evento gerador do processo modelado, bem como o ator responsável por este evento, e determina como o processo deve se desenrolar e ser concluído por meio do envio de mensagens, que em geral disparam métodos entre os objetos (GUEDES, 2014, p.20).

A Figura 7 ilustra um exemplo do diagrama de sequência.

Figura 7 - Exemplo de Diagrama de Sequência da UML



Fonte: Próprio autor

O diagrama "é organizado colocando-se os objetos correspondentes na parte superior, ao longo do eixo horizontal; e suas respectivas mensagens são colocadas ao longo do eixo vertical, em uma ordem cronológica, de cima para baixo" (SEABRA, 2001, p.17). Assim, o diagrama de sequência descreve em uma ordem cronológica os passos que serão realizados para que determinada funcionalidade seja completada, ilustrando a comunicação entre o ator e objetos. As seções seguintes detalham alguns dos conceitos relacionados ao diagrama de sequência, tais como: atores, objetos, linha de vida e mensagens.

2.3.1 Atores

"Os atores modelados neste diagrama são instâncias dos atores declarados no diagrama de caso de uso, representando entidades externas que interagem com o sistema e que solicitam serviços, gerando, assim, eventos que iniciam processos" (GUEDES, 2011, p.193). Ou seja, são representações de um ou mais usuários que realizam a ação, para que a mesma seja executada pelo sistema. Assim, o ator é representado por um boneco magro, representado pela Figura 8, juntamente com a linha de vida e o nome do(s) autor(es).

Figura 8 - Ator do Diagrama de Sequência da UML

Fonte: Próprio autor

O Ator, ilustrado pela Figura 8, possui em sua parte inferior uma linha pontilhada, denominada linha de vida, a qual é utilizada para interagir com um objeto ou outro ator. Essa interação é estabelecida por mensagens trocadas entre objetos e atores, que é ilustrada por uma seta. Na seção 2.3.2 serão apresentados os conceitos que envolvem a criação de Objetos no diagrama de sequência.

2.3.2 Objetos

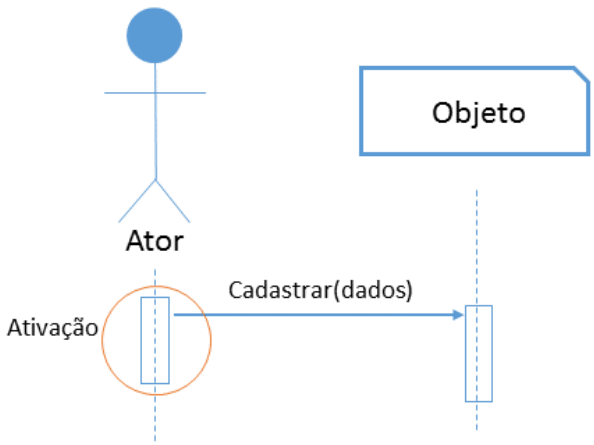
"Um diagrama de sequência é um diagrama que mostra objetos reais e interações entre objetos no sentido horizontal, e sequência no sentido vertical" (TEPFENHART, 2001, p.521). Esses objetos representam uma instância de uma classe ou método que será utilizado no

diagrama para executar alguma ação. Essas instâncias podem ser acessadas via mensagens e são enviadas por atores ou outras instâncias. Para que a comunicação entre essas instâncias ou atores seja realizada, é preciso existir a sua linha de vida, que representa o tempo de existência da instância. Na seção 2.3.3 serão expostos conceitos sobre a linha de vida de atores e objetos.

2.3.3 Linha de Vida

A Linha de Vida "representa o tempo em que um objeto existe durante um processo" (GUEDES, 2011, p.45), ou seja, é o tempo de vida de um ator ou objeto presente em um diagrama. Essa Linha de Vida é representada por uma linha tracejada verticalmente abaixo do ator/objeto, na qual pode ser interrompida com um "X" ao final da linha. Um ator/objeto só é instanciado no diagrama quando o mesmo será utilizado, isso implica na existência de uma ordem cronológica ao instanciar um objeto/ator. A Figura 9 apresenta uma instância de ator e objeto com suas respectivas linhas de vida.

Figura 9 - Linha de Vida do Ator e do Objeto



Fonte: Próprio autor

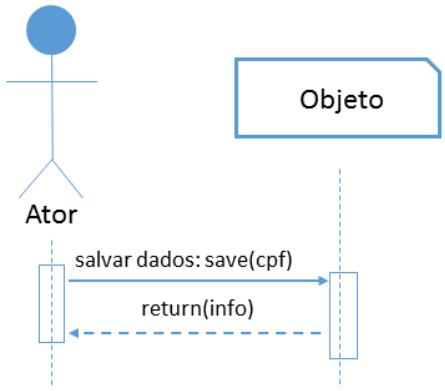
A Figura 9 apresenta um exemplo de diagrama de sequência que ilustra a comunicação entre o ator e um objeto. Essa comunicação é representada pela seta iniciada no ator até o objeto. As setas que representam esta comunicação precisam iniciar a partir de um retângulo, que representa a ativação do objeto/ator na linha de vida, como ilustrado na figura acima. Além

disso, “durante um período de ativação, o objeto respectivo está em execução realizando algum processamento. Nos períodos em que o objeto não está ativo, ele está alocado (ele existe), mas não está executando nenhuma instrução” (STADZISZ *online*, 2002, p.30). Na seção 2.3.4 serão apresentados os conceitos sobre as mensagens presentes na linha de vida do ator/objeto.

2.3.4 Mensagens

“As mensagens são utilizadas para demonstrar a ocorrência de eventos, que normalmente forçam a chamada de um método em algum dos objetos envolvidos no processo” (GUEDES, 2011, p.196). Essas mensagens podem ser classificadas em dois tipos, mensagens de envio, que são setas não tracejadas que partem de um objeto ou ator até o destinatário no sentido da esquerda para direita, e mensagens de retorno, que são setas tracejadas no sentido oposto da mensagem de envio. As mensagens podem executar um método, enviar parâmetros e/ou expressar a comunicação entre dois atores. As setas apontam sempre para onde a mensagem será executada, é dessa forma que as classes colaboram (MEDEIROS, 2004, p.148). A Figura 10 ilustra uma mensagem de envio e retorno.

Figura 10 - Mensagem de envio e retorno



Fonte: Próprio autor

É possível perceber, a partir da Figura 10, a comunicação entre o ator e o objeto, na qual é executado um método *save* que recebe como parâmetro um CPF. Esse método que pertence ao objeto é executado e retorna uma informação para o ator. Para descrever o título das

mensagens de envio, é utilizado um padrão para que seja possível entender o que a mensagem faz, qual método será executado e os parâmetros que serão enviados. Esse padrão corresponde em uma pequena descrição da ação prosseguida de dois pontos, o nome do método e parâmetros existentes (MEDEIROS, 2004; TEPFENHART, 2001; GUEDES, 2011), como ilustrado na Figura acima. Além disso, "mensagens podem incluir números de sequência, nomes de operação e parâmetros reais" (TEPFENHART, 2001, p.521).

Após expor os principais conceitos relacionados a este trabalho, serão apresentados na seção 2.4 trabalhos relacionados, os quais possibilitarão obter uma visão geral dos trabalhos científicos produzidos que envolvam análise de Pontos de Função e utilizam um ou mais artefatos ou metodologias como entrada para realização da análise.

2.4 Trabalhos Correlatos

Nessa seção serão apresentados trabalhos relacionados à análise de Pontos de Função que utilizam algum modelo, técnica ou artefato como auxilio na mensuração dos Pontos de Função de projetos de software.

2.4.1 Contagens a partir de Artefatos da UML

Batista *et al.* (*online,* 2011) propõem uma ferramenta de apoio para especialistas em Pontos de Função, que utiliza um modelo chamado "Modelo de Problema" para determinar o tamanho funcional. Esse modelo é semelhante ao Modelo de Análise do RUP e segue os padrões da norma IEEE-830 (BATISTA *et al. online,* 2011, p.2, tradução nossa), que representa uma norma de recomendações para especificações de exigências de software.

O Modelo de Problema é dividido em dois pontos de vista: visão de requisito, que "descreve o produto desejado a partir do ponto de vista do usuário, representando funções desejadas como Casos de Uso" (BATISTA *et al. online,* 2011, p.2, tradução nossa), e visão de

análise, que “descreve o produto desejado a partir do ponto de vista do desenvolvedor [...] modelando conceitos do problema de domínio, procedimentos e interfaces como classes, e suas interações por estereotipados Colaborações UML” (BATISTA *et al. online,* 2011, p.3, tradução nossa). Além disso, a ferramenta requer alguns processos manuais como a entrada dos tipos de registros e o agrupamento de classes que possuem relacionamentos um-pra-um, que, na concepção dos autores, representam uma mesma estrutura e precisam ser mapeados manualmente pelo especialista.

Similar ao trabalho citado, Uemura *et al.* (2001) propõem uma ferramenta que “pode contar PFs a partir das especificações de projetos desenvolvidos pela *Rational Rose*” (UEMURA *et al.,* 2001, p. 233, tradução nossa), que é uma ferramenta da IBM utilizada para desenvolver modelos com padrões da UML. A ferramenta utiliza dois artefatos UML como entrada, são eles: o diagrama de sequência e o diagrama de classe; produzidos a partir da ferramenta, e tem como saída os Pontos de Função não ajustados, funções transacionais, funções de dados e objetos que podem estar relacionados com o cálculo PFs (UEMURA *et al.,* 2001, tradução nossa). Além disso, a ferramenta define algumas etapas, regras e padrões para que seja possível analisar os artefatos desenvolvidos na ferramenta *Rational Rose* e, assim, inferir o tamanho funcional do projeto de software.

2.4.2 Contagens a partir de Metodologias

Os dois trabalhos citados na seção anterior utilizam uma forma semiautomática para mensurar o tamanho funcional de um projeto de software. Eles utilizam modelos UML como diagrama de sequência e diagrama de caso de uso para mitigar o trabalho manual e tentar automatizar a técnica de análise de Pontos de Função utilizando a documentação de modelagem.

Pinel (2012) também propõe a utilização de modelos UML para automatização da contagem de Pontos de Função. A diferença é que ele utiliza a metodologia *Model Driven*

Architecture (MDA) nesta automatização, a qual "automatiza o ciclo de vida de projetos, [...] reduzindo o tempo de desenvolvimento e permitindo a padronização do código fonte" (PINEL, 2012, p. 2). Essa metodologia é baseada em modelos UML e utiliza, dentre outros padrões, a UML como linguagem de modelagem. A MDA exige a construção de modelos com alto nível de abstração, independentemente de qualquer tecnologia. O autor utiliza esse alto nível em prol da técnica de Pontos de Função para analisar os diagramas de classe, diagramas de caso de uso e diagramas de atividades, e mensurar o tamanho funcional do software. A ferramenta proposta pelo autor analisa sistemas de informação já desenvolvidos ou em finalização. O autor expressa que "apesar da contagem ser aplicada a projetos já desenvolvidos, os resultados produzidos podem ser utilizados para ajustar e melhorar a precisão das regras da APF" (PINEL, 2012, p. 2).

Similar a proposta de Pinel (2012), Fraternali *et al.* (2006) propõem a contagem de Pontos de Função também utilizando a MDA. Entretanto, utiliza como componente essencial uma modelagem diferenciada, na qual adequa tanto a geração de código como a estimativa de tamanho. A metodologia utilizada em sua proposta é a WebML, que "explora de uso geral diagramas de classe UML para representar os objetos de negócios subjacentes à aplicação, e uma notação de domínio específico, chamados de diagramas de hipertexto, para expressar a estrutura do *front-end* de aplicações" (FRATERNALI *et al. online* 2006, p. 2). Essa metodologia, orientada a desenvolvimento Web, é dividida em três níveis: os objetos de conteúdo, que representa a estrutura das classes, equivalente ao diagrama de classe UML; a organização *front-end*, que é uma divisão hierárquica das áreas da aplicação, similar ao diagrama de navegação; e aparência da aplicação, que representa as interfaces do sistema. Assim, o autor tenta utilizar os diagramas da metodologia WebMAL para contagem de Pontos de Função.

3. Materiais e Métodos

Nesta seção são apresentados a metodologia utilizada para o desenvolvimento deste trabalho, bem como os procedimentos realizados durante a construção da aplicação para que seja possível realizar a mensuração do tamanho funcional do software a partir da visão do usuário. Este trabalho tem como característica ser uma pesquisa aplicada, tendo como propósito resolver um problema real e, assim, facilitar o trabalho dos gestores de projetos na mensuração do tamanho do software.

Para que seja possível alcançar o objetivo com êxito, o desenvolvimento deste trabalho foi dividido em 5 etapas. A Figura 11 ilustra as 5 etapas que representam a metodologia do trabalho.

Figura 11 - Metodologia

Fonte: Próprio autor

A primeira etapa, representada na Figura 11, teve grande importância, pois nela foram realizados os estudos aprofundados sobre APF, Diagrama de Sequência da UML e script DDL da SQL do banco de dados. A partir desses estudos e entrevistas informais realizadas com a

Professora Cristina D'Ornellas Filipakis Souza, foi possível consolidar o conhecimento necessário para relacionar os mesmos e, assim, entender como o Diagrama de Sequência e o script DDL poderiam ser utilizados como entrada para a técnica de Análise de Pontos de Função.

A partir da conclusão da primeira etapa, foi possível realizar a elaboração do projeto (2º Etapa), consolidando os objetivos, hipóteses e justificativas do trabalho. Além disso, foi realizado o levantamento das tecnologias e ferramentas que foram utilizadas na etapa de desenvolvimento, na qual foi utilizada a linguagem C# juntamente com o *framework* ASP.NET e *Model-View-Controler* (MVC) na versão 4 como arquitetura de desenvolvimento, bem como Visual Studio 2013 como IDE de desenvolvimento e SQL Server 2008 R2 como base de dados.

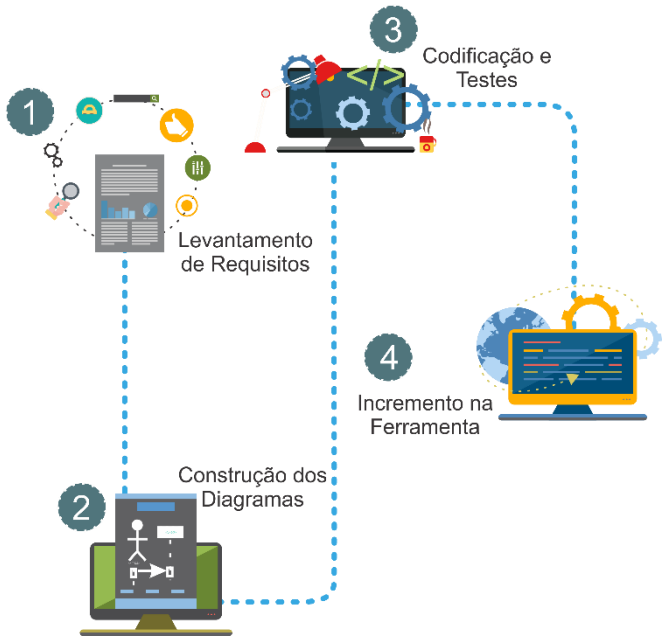
Para apresentação dos conceitos envolvidos, adotou-se como procedimento metodológico a pesquisa bibliográfica, com embasamento em livros, artigos, dissertações de mestrado, teses de doutorado e trabalhos de conclusão de curso. Os conceitos abordados foram:

- Análise de Pontos de Função (APF), na qual foram apresentados os conceitos da técnica, bem como o processo de análise, determinação e contagem dos Pontos de Função (PF) a partir da visão do usuário;
- *Data Definition Language* (DDL) SQL, que traz as definições, estruturas e sintaxe da linguagem que representa a estrutura do banco de dados; e
- Diagrama de Sequência da UML, na qual são abordados seus conceitos e definições, bem como a estrutura do mesmo.

Já a terceira etapa é constituída pelo desenvolvimento dos três módulos que foram, ao final desta etapa, acoplados ao software final. Esse software, que foi desenvolvido por Willian Almeida Rodrigues em seu trabalho de Estágio Supervisionado em Sistemas de Informação, possibilitou a criação de artefatos de modelagem. Acoplado com as novas funcionalidades, o

mesmo também possibilitou a mensuração do tamanho funcional. A Figura 12 apresenta a metodologia utilizada no desenvolvimento dos módulos da ferramenta.

Figura 12 - Fases do Desenvolvimento da Ferramenta



Fonte: Próprio autor

Esta metodologia, ilustrada na Figura 12, representa quatro fases do desenvolvimento, são elas: Levantamento de Requisitos, indagação das funcionalidades que foram implementadas; Construção dos Diagramas, que representa a fase de desenvolvimento dos diagramas de caso de uso e de classe para melhor entendimento das funcionalidades desenvolvidas; Codificação e Testes, desenvolvimento dos módulos de acordo com o planejamento realizado nas etapas anterior e testes informais de verificação das funcionalidades implementadas; e Incremento na Ferramenta, que é o acoplamento dos três módulos ao software final. Os módulos que foram desenvolvidos são:

- Módulo de Interpretação de *script Data Definition Languague* (DDL) SQL, que é capaz de interpretar o *script* DDL para auxiliar na mensuração dos Pontos de Função;
- Módulo de diagrama de sequência da UML, que é capaz de permitir a construção dos diagramas de sequências e mapear as variáveis utilizadas; e

- Módulo de Mensuração de Pontos de Função, que é capaz de analisar as informações providas da interpretação do *script* DDL do banco de dados e dos diagramas de sequência da UML e determinar o tamanho funcional do projeto.

A quarta etapa consiste em testes de verificação dos módulos desenvolvidos, ou seja, testes de análise realizada pela ferramenta, comparando análises realizadas manualmente juntamente com análises da ferramenta. Assim, foi possível verificar e ajustar a ferramenta para que a análise seja realizada com uma maior precisão.

Por fim, foi realizada a documentação de tudo o que foi desenvolvido, envolvendo o processo de implementação da ferramenta, os testes de verificação de análise realizada pela ferramenta perante a análise realizada manualmente, resultando na elaboração da seção de resultados e discussão da monografia.

4. Resultados e Discussão

Esta seção apresenta os resultados e discussão acerca deste trabalho, abordando primeiramente o desenvolvimento dos módulos (seção 4.1), possibilitando o entendimento e o funcionamento das funcionalidades. Posteriormente, são apresentados os testes de verificação (seção 4.2) realizados, que possibilitam verificar se o processo de Análise de Pontos de Função realizado pelos módulos desenvolvidos é eficaz.

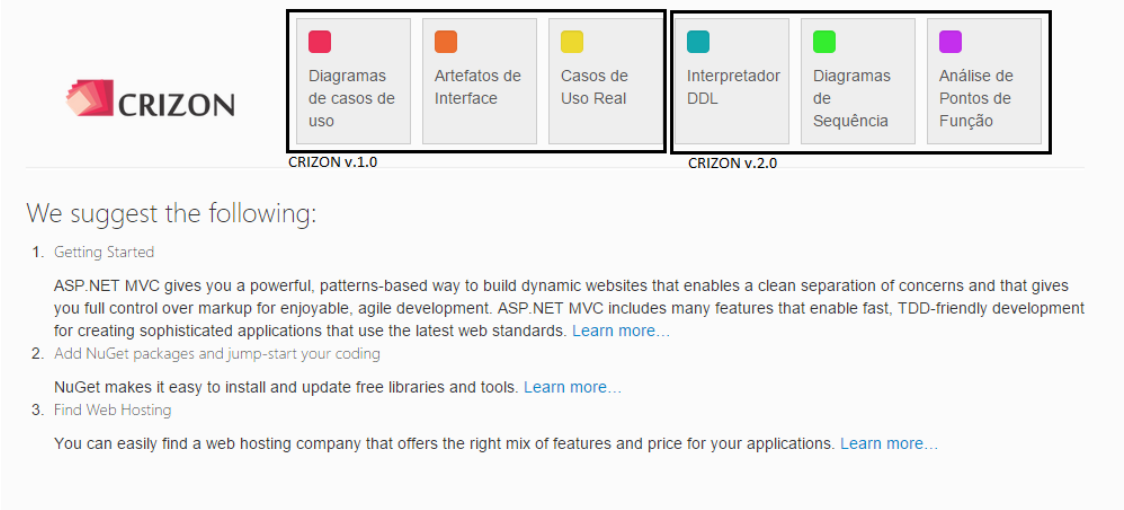
4.1. Desenvolvimento dos Módulos

Os módulos propostos neste trabalho têm como objetivo auxiliar e facilitar o processo de análise e cálculo de Pontos de Função, de forma que seja mitigado o tempo da análise e eliminada a necessidade de possuir um conhecimento abrangente sobre a técnica de Pontos de Função. Dessa forma, o usuário que necessita realizar a análise de um software precisará apenas do *script* DDL do banco de dados, que representa as tabelas e relacionamentos existentes entre elas, e o entendimento das funcionalidades existentes no software analisado para a construção dos diagramas de sequências.

4.1.1. Visão Geral dos Módulos

A análise realizada a partir dos módulos divide-se em três etapas, que representam desde a interpretação do *script* DDL do banco de dados até a análise dos dados existentes no sistema para inferência dos Pontos de Função. A Figura 13 ilustra os módulos presentes no CRIZON na sua versão 1.0 e os três módulos desenvolvidos neste trabalho, que representa a versão 2.0 do software.

Figura 13 - Módulos do CRIZON na versão 1.0 e 2.0



Os três módulos, representado pela figura acima, na versão 2.0, possibilita a mensuração do tamanho funcional do software a partir de três etapas, são elas:

- Interpretação do script DDL – consiste na interpretação do script DDL do banco de dados para definição das entidades existentes no software a ser analisado. Essa etapa é subdividida em: definição de entidades, atividade que define as entidades (tabelas) existente no script; definição de Tipos de Dados, onde são definidos quais são os dados (colunas) existentes na em cada entidade; definição de Validação, onde é verificado cada dado das entidades para determinação da sua validade a partir dos conceitos da técnica.
- Construção dos Diagramas de Sequências – consiste na criação e definição das transações ou funções realizadas pelo software a ser analisado, de forma que indique primeiramente os atores e objetivos pertencentes a ação, o tipo da ação a ser realizada entre os atores e objetos, quantas mensagens ela poderá exibir, e quais dados (colunas) das entidades providas da interpretação realizada na etapa anterior são utilizadas no diagrama a ser cadastrado.
- Análise de Pontos de Função – representa a análise dos grupos de dados (entidades) e transações (diagramas) cadastrados, de forma que indique primeiramente quais as

entidades e diagramas participaram desta análise, bem como os tipos de dados existentes para que, ao final desta análise, seja possível inferir a quantidade de Pontos de Função que o software possui e gerar um relatório com as informações providas desta análise.

As etapas listadas acima são explanadas nas próximas seções, onde serão descritos como os módulos foram implementados e a apresentação de algumas telas, bem como os artefatos gerados na fase de modelagem. Com todas as etapas anteriores concluídas, é possível realizar o processo de análise de Pontos de Função que, ao final, resulta na quantidade de PF que tem o software analisado. A cada análise realizada pelo sistema é gerado um relatório, na qual é possível visualizar todas as inferências realizadas na análise. Após o entendimento do sistema, a partir da visão geral, a próxima seção (4.1.1) apresenta os artefatos gerados para documentar os módulos desenvolvidos neste trabalho.

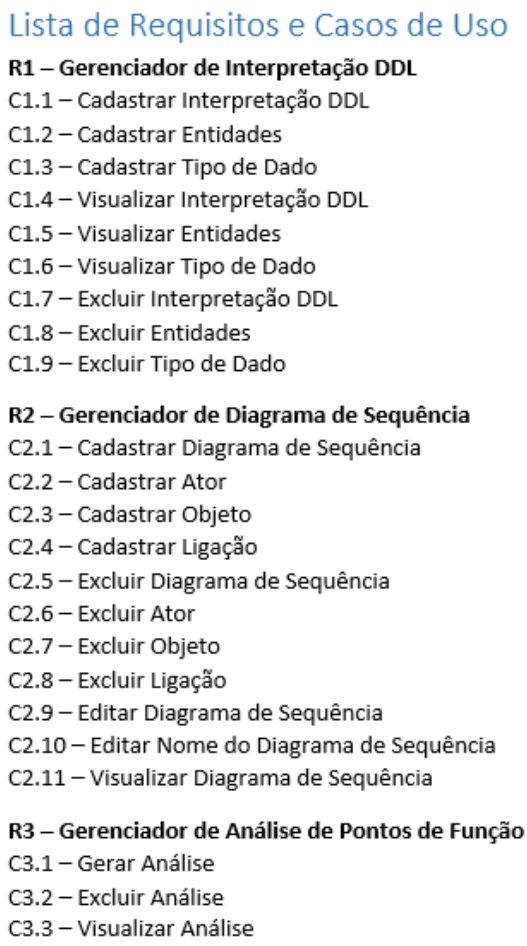
4.1.2. Artefatos

Os artefatos são de grande importância para os desenvolvedores, pois eles representam ou descrevem funcionalidades, funcionamento, arquitetura ou ações existentes no software. Além disso, artefatos de software são relevantes para manutenção do software, pois é a partir dela que a equipe de manutenção terá conhecimento do sistema como um todo, e dos módulos e funcionalidades existentes no mesmo. A seguir, serão apresentados os artefatos desenvolvidos para documentar os módulos, são eles:

- Lista de Requisitos e Casos de Uso - é um dos documentos propostos na engenharia de software que documenta as funcionalidades que existirão no sistema. “Um caso de uso deve especificar as expectativas de comportamento de um sistema, quando este apoia uma e somente uma transação do negócio” (OLIVEIRA *online*, 2009). Já os requisitos, representam as funcionalidades necessárias para atender o objetivo do software. A

Figura 14 ilustra a Lista de Requisitos e Casos de Uso dos módulos propostos neste trabalho.

Figura 14 - Lista de Requisitos e Casos de Uso



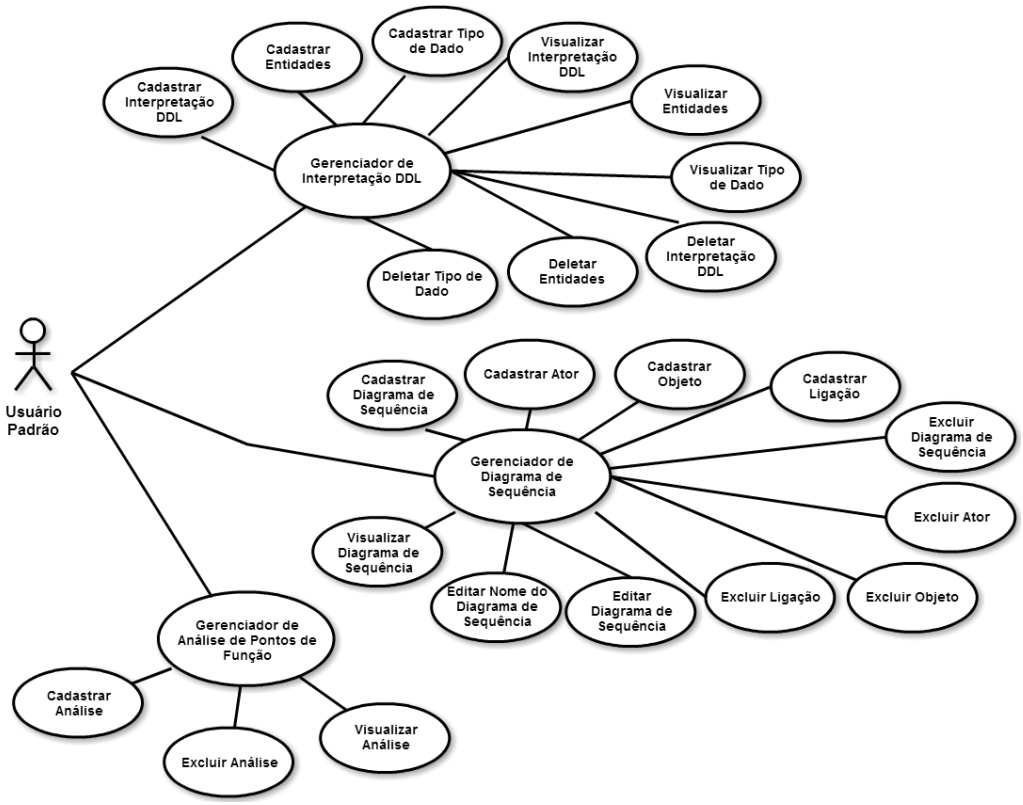
Fonte: Próprio autor

O Lista de Requisitos, ilustrada na Figura 14, apresenta as funcionalidades existentes nos módulos propostos neste trabalho, dividido em três módulos, são eles: Gerenciador de Interpretação DDL, que oferece as funcionalidades de cadastro, visualização e exclusão de uma interpretação DDL, além de possibilitar o cadastro, edição, visualização e exclusão das Entidades e Tipos de Dado; Gerenciador de Diagrama de Sequência, que agrupa as funcionalidades necessárias para possibilitar criar e manipular os Diagramas de Sequências existentes no sistema; e, Gerenciador de

Análise de Pontos de Função, que possibilita cadastrar, visualizar e excluir análises realizadas, bem como é responsável pela análise e cálculo para determinação dos Pontos de Função. Após a definição das funcionalidades existentes nos módulos, é preciso identificar quais os atores existentes nos módulos e as ações disponíveis para os mesmos. O próximo tópico apresenta o diagrama de caso de uso, que representa as ações possíveis de cada ator do sistema.

- Diagrama de Caso de Uso - é um dos nove diagramas pertencentes ao *Unified Modeling Language* – UML, onde "UML é uma linguagem gráfica padrão para a elaboração da estrutura de projetos complexos de software" (I-WEB *online*, 2013). Esse diagrama descreve as operações possíveis, de acordo com cada usuário do sistema, ou seja, "descreve as principais funcionalidades do sistema e a interação dessas funcionalidades com os usuários do mesmo sistema" (RIBEIRO *online*, [s.d.]). Nos módulos desenvolvidos, tem-se apenas um tipo de usuários, sendo ele:
    - Usuário Padrão – é qualquer usuário cadastrado do sistema, possuindo uma gama de funcionalidades que serão acessíveis a ele. Na Figura 15, é possível observar as funcionalidades interligadas ao ator usuário.

Figura 15 - Diagrama de Caso de Uso



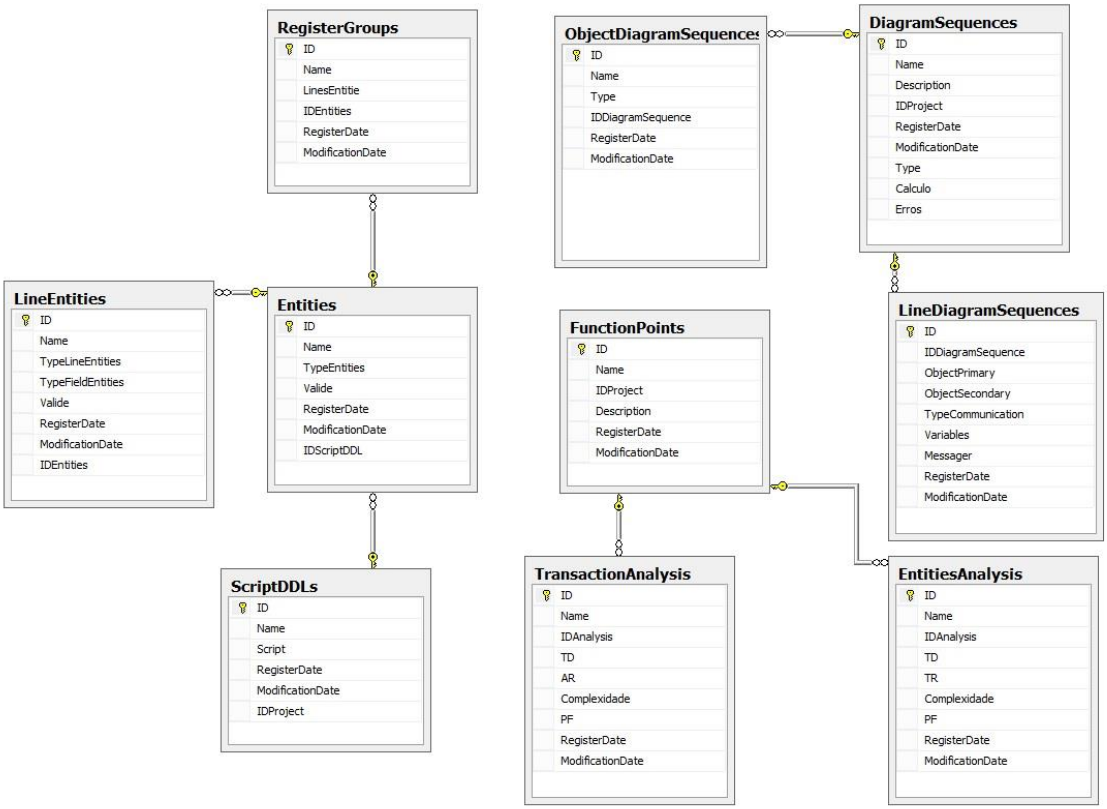
Fonte: Próprio autor

É possível observar, a partir da Figura 15, as funcionalidades disponíveis para o usuário. Dadas as funcionalidades pertencentes ao ator, bem como a dependência dessas funcionalidades, é necessário apresentar a estrutura de dados dos módulos, de forma que seja possível entender quais dados são gerados e mantidos na aplicação no decorrer das ações, bem como as operações implementadas no sistema.

- Modelo Relacional do Banco de Dados - consiste na descrição das tabelas do banco de dados, de forma que seja possível identificar as tabelas, colunas e relacionamentos existentes. Essa descrição é feita de forma textual e é derivada do Modelo Conceitual (MC) do banco de dados. O modelo relacional de banco de dados é a representação dos relacionamentos das tabelas existentes no banco de dados, de forma que sejam visíveis as dependências e os tipos de relacionamentos (um pra um, um pra n ou n pra n) existentes entre as tabelas. Segundo Costa (*online*, p. 33, 2011) “o Modelo Relacional

(MR) é um modelo de dados representativo (ou de implementação) que foi proposto por Ted Codd, em 1970. O modelo fundamenta-se em conceitos da matemática – teoria dos conjuntos e lógica de predicado". A Figura 16 apresenta o Modelo Relacional desenvolvido.

Figura 16 - Modelo Relacional do Banco de Dados

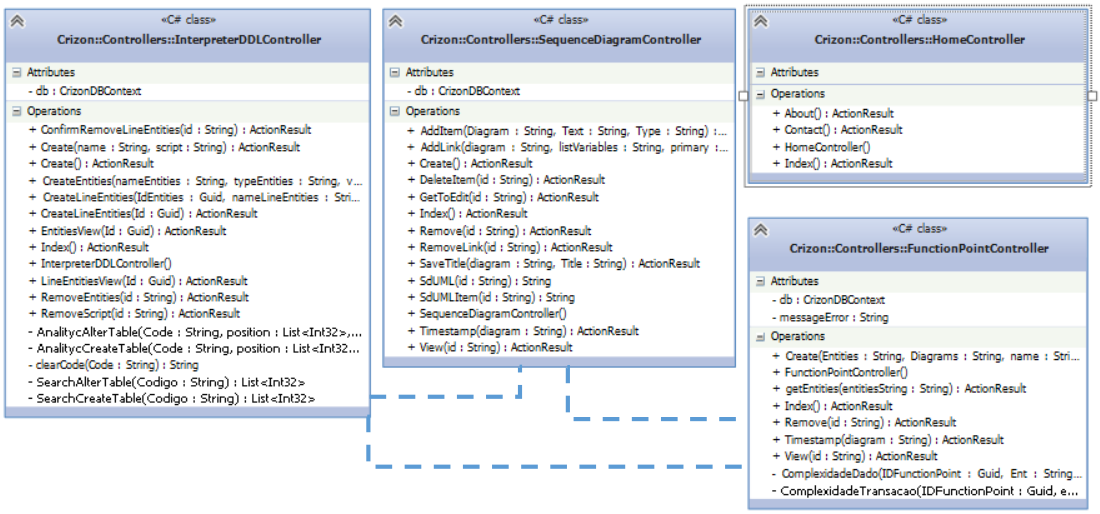


Fonte: Diagrama Gerado pelo Microsoft SQL Serve Management Studio

Na Figura 16 é apresentado o modelo relacional do banco de dados. A tabela "Entitie", responsável por armazenar o tipo de dado, possui uma coluna que define se a mesma é válida ou não. A validade dessa coluna irá refletir na sua utilização para determinar os Pontos de Função. Caso o projeto em análise possua vários TDs que representam um TR, somente um TD será válido, esta inferência é realizada no cadastro de TR.

- Diagrama de Classes – o diagrama de classes também faz parte do pacote de diagramas UML, sendo o responsável por “descrever o objeto e as estruturas usadas pelo seu aplicativo internamente e comunicação com seus usuários de informações” (MSDN *online*, [s.d.]), ou seja, denota as operações e atributos utilizados internamente pelo sistema, operações essas responsáveis por ações e funcionalidades do sistema. A Figura 17 apresenta este diagrama.

Figura 17 - Diagrama de Classe

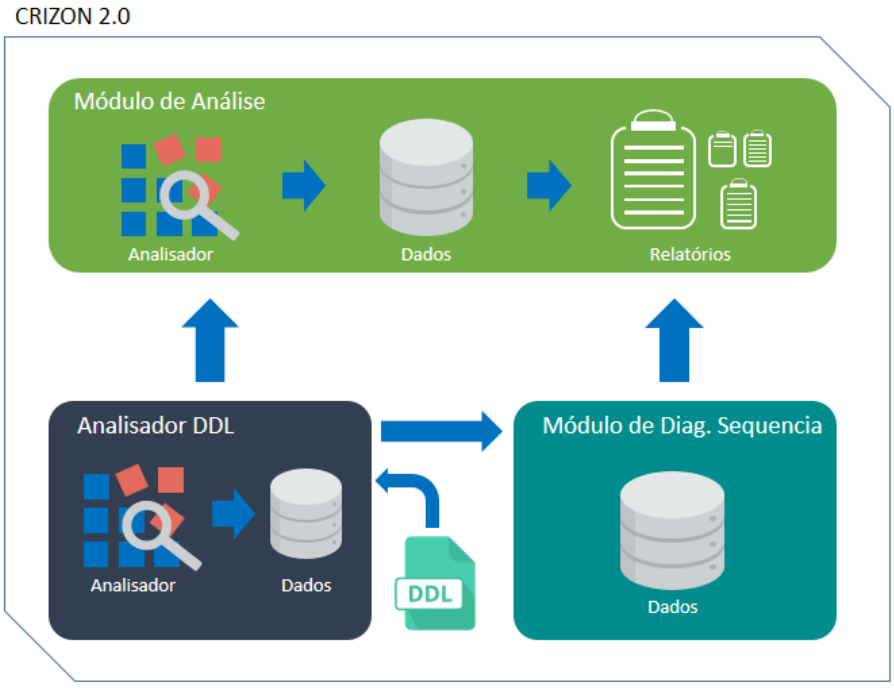


Fonte: Próprio autor

O diagrama de classe, ilustrado na Figura 17, foi gerado a partir do Visual Studio e demonstra as funcionalidades implementadas para realizar as funções disponíveis no sistema. Esse diagrama é dividido em classes, onde cada classe é responsável por um determinado módulo do sistema, são eles: *InterpreterDDL*, *SequenceDiagram* e *FunctionPoint*. Todas as classes existentes neste diagrama possuem um atributo em comum, que é a classes do banco de dados, responsável pelas operações e acesso as informações do banco, e usuário autenticado, que representa a classe do usuário autenticado no sistema. Além disso, as classes também possuem um conjunto de operações que são responsáveis por executar uma determinada ação, sendo que algumas dessas operações necessitam de parâmetros de entrada para sua execução.

- Arquitetura do Software - envolve a descrição de elementos arquiteturais dos quais os sistemas serão construídos, interações entre esses elementos, padrões que guiam suas composições e restrições sobre estes padrões (LOBATO apud GARLAN, p.13, 2011). Antes de desenvolver a arquitetura de um software, é preciso determinar qual visão será utilizada para o desenvolvimento, de forma que cada visão "separa diferentes aspectos em visões separadas com o objetivo de gerenciar complexidade" (LEITE online, 2007), além disso, cada visão representa diferentes conceitos da engenharia. A visão da arquitetura desenvolvida foi a de desenvolvimento/estrutural, que ilustra a estrutura do software dividindo-a em módulos de desenvolvimento. A Figura 18 apresenta a visão de desenvolvimento/estrutural dos módulos propostos neste trabalho.

Figura 18 - Arquitetura dos módulos propostos a partir da visão de desenvolvimento/estrutural.



Fonte: Próprio autor

Na Figura 18 é apresentada a arquitetura estrutural dos módulos, que é dividida em três partes, sendo elas:

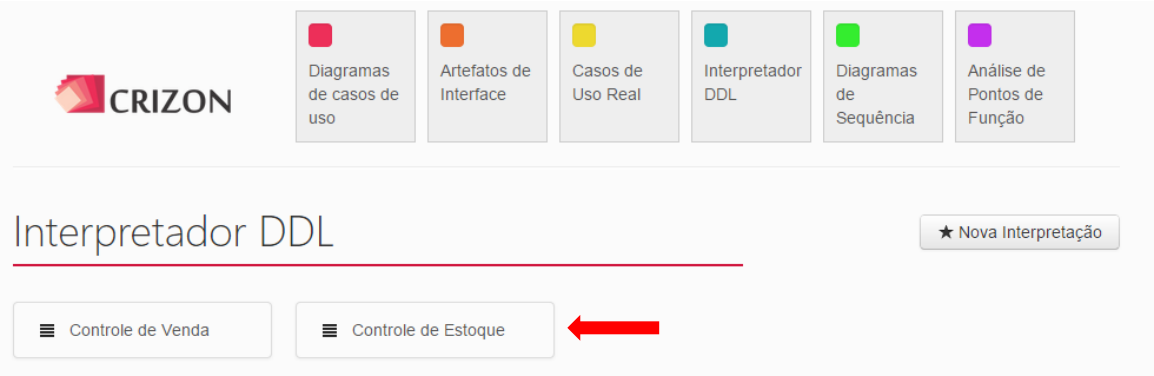
- Analisador DDL – é o módulo responsável por interpretar o *script* DLL da SQL, documento de entrada fornecido pelo usuário; e identificar as entidades e colunas existentes, bem como seus relacionamentos, gerando assim os dados necessários para auxiliar a construção dos diagramas e a realização da análise de Pontos de Função;
- Módulo de Diagrama de Sequência – representa o módulo que possibilita o usuário construir os diagramas de sequência que pertencem ao software a ser analisado para que, posteriormente, seja utilizado como entrada para análise de Pontos de Função. Esse módulo também recebe informações providas do Analisador DDL, possibilitando a seleção das variáveis que transitam nas mensagens de cada diagrama; e,
- Módulo de Análise – módulo referente à análise de Pontos de Função, tendo como fonte de entrada as informações providas dos módulos anteriores que serão utilizadas para a realização da análise. Para isso, esse módulo contém um script que analisará tais informações que, ao final, possibilitará a inferência da quantidade de Pontos de Função.

Após ser apresentado os artefatos desenvolvidos na fase de planejamento, que documentam em uma forma mais detalhada suas funcionalidades, bem como a visão geral de cada módulo e a arquitetura do software, é preciso apresentar detalhadamente como decorreu o processo de implementação do sistema. Dessa forma, as próximas seções (4.1.3, 4.1.4 e 4.1.5) apresentam como foram implementados os três módulos do sistema. Vale ressaltar que as telas adicionadas seguem o mesmo padrão da aplicação CRIZON v.1, desenvolvida pelo acadêmico William Almeida (RODRIGUES, 2012), assim como foram utilizados *scripts* e imagens desenvolvidos pelo mesmo.

4.1.3. Módulo Analisador DDL

O módulo analisador DDL é responsável por receber o script DDL da SQL do usuário através de um formulário, interpretá-lo e construir as Entidades e Dados contidos no script, eliminando assim a necessidade do usuário realizar esse processo manualmente. Esta fase se inicia ao criar uma nova interpretação e, após a realização da análise, são armazenadas todas informações providas das inferências realizadas. A Figura 19 ilustra a tela de listagem de análises realizadas.

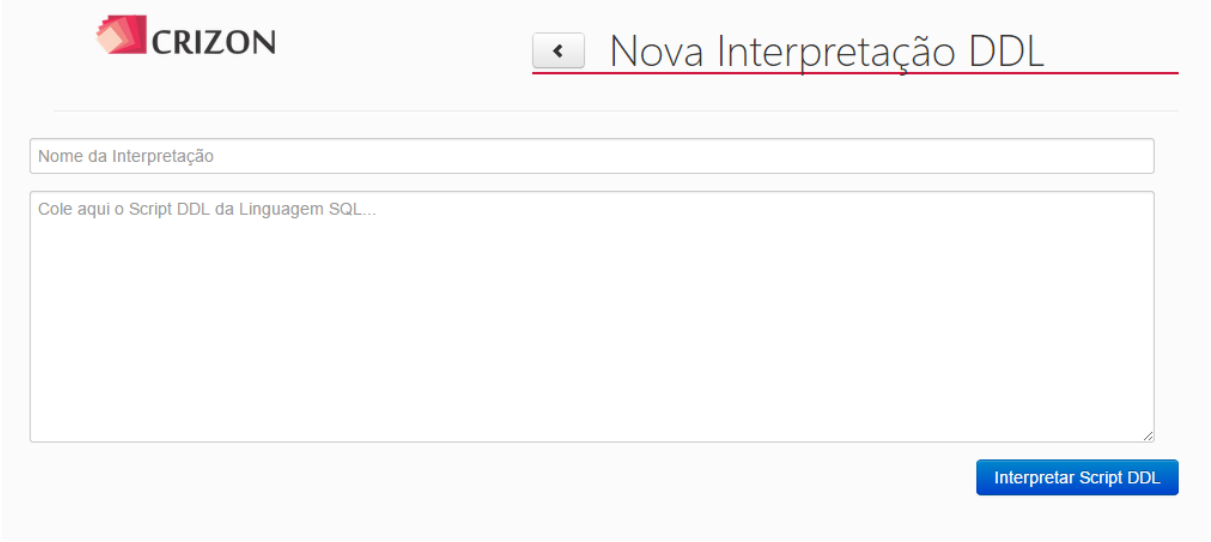
Figura 19 - Listagem de interpretações realizadas



Fonte: Próprio autor

De acordo com a Figura 19, é possível observar a listagem de interpretações realizadas, indicado pela seta na figura. Entretanto, para que seja listada a interpretação é preciso, primeiramente, cadastrá-la a partir do botão “Nova Interpretação”. Após acessar o botão, é exibido a tela de cadastro da nova interpretação, ilustrada pela Figura 20.

Figura 20 - Nova interpretação



Fonte: Próprio autor

Na tela de nova interpretação, ilustrada pela Figura 20, o usuário informará o nome que deseja conceder à interpretação a ser realizada, bem como a inserção do *script* no campo seguinte. Ao preencher todos os campos, o usuário acessará o botão "Interpretar *Script* DDL", que é responsável por iniciar o processo de interpretação. Esse processo consiste na busca pelas palavras-chaves que estruturam o banco de dados de acordo com linguagem SQL. A Figura 21 exibe uma parte do código desenvolvido responsável por encontrar as tabelas existentes no *script* DDL.

Figura 21 - *Script* responsável por buscar as tabelas

```
1 reference
private List<int> SearchCreateTable(string Codigo) //Buscando os create table
{
    List<int> indiceCreateTable = new List<int>();
    int start = 0;
    int posicao = Codigo.IndexOf("create table", start);
    while (posicao != -1)
    {
        indiceCreateTable.Add(posicao);
        start = posicao + 1;
        posicao = Codigo.IndexOf("create table", start);
    }
    return indiceCreateTable;
}
```

Fonte: Próprio autor

É possível observar, a partir da Figura 21, que o método “SearchCreateTable” recebe por parâmetro o código inserido pelo usuário, para que seja possível processar essa busca. Além disso, o código já é recebido com todos os caracteres minúsculos, para facilitar a busca das palavras desejadas. A primeira linha do código é responsável por criar uma lista de índices que, posteriormente, será preenchida com os índices que contêm a criação de uma nova tabela. Já a segunda linha cria uma variável denominada *start*, essa variável é responsável por guardar o índice referente à próxima posição que será buscada no código. Em seguida, é inserida na variável *posição* a posição no código que possui as palavras “*create table*”, que representam a criação de uma nova tabela. Após essa busca, é realizado um laço de repetição desde que seja encontrada a palavra no código informado pelo usuário. Essa posição encontrada é guardada na lista criada anteriormente, que será enviada para o método mãe de análise. Esse método é chamado após o usuário enviar as informações para o método de análise, que posteriormente instancia o método de busca das tabelas. A Figura 22 apresenta o código responsável por encontrar as linhas ou colunas das tabelas existentes.

Figura 22 - Script responsável por encontrar as colunas das tabelas existentes

```
//verificando as virgulas
List<string> linhasEntitys = new List<string>();
int posiVirgula = 0;
for (int pos = 0; pos < CodeEntity.Length; pos++)
{
    if (CodeEntity[pos] == ',')
    {
        int x = 0;
        int y = 0;
        if (!Int32.TryParse(CodeEntity[pos - 1].ToString(), out x) && !Int32.TryParse(CodeEntity[pos + 1].ToString(), out y))
        {
            linhasEntitys.Add(CodeEntity.Substring(posiVirgula, pos - posiVirgula).Trim());
            posiVirgula = pos + 1;
        }
    }
    if (CodeEntity.Length - 1 == pos)
    {
        linhasEntitys.Add(CodeEntity.Substring(posiVirgula, CodeEntity.Length - posiVirgula).Trim());
    }
}
```

Fonte: Próprio autor

O script ilustrado na Figura 22 apresenta a busca pelas colunas existentes nas tabelas encontradas anteriormente. Para isso, é preciso analisar caractere por caractere do *script* DDL que, anteriormente, foi particionado em uma lista de tabelas, na qual foi utilizado como critério

de busca as palavras *create table* para início da tabela e a busca pelo parêntese final que encerra a criação da tabela. Após esse particionamento, é realizada uma verificação nas linhas da tabela, buscando no código onde possui vírgulas, pois cada linha é separada da sua sucessora por uma vírgula. Porém, é preciso verificar se a virgula encontrada não representa um tipo de variável, como no exemplo a seguir:

```
create table Contato (Id INT NOT NULL IDENTITY(1,1), Nome VARCHAR(200))
```

Neste exemplo, a coluna denominada “Id” irá incrementar de um a um o número a cada registro novo, para isso, foi adicionada uma virgula ao denominar esse tipo de variável. No *script* ilustrado na Figura 22, o segundo condicional é responsável por verificar se essa virgula encontrada não representa uma denominação de tipo de variável, tendo assim uma busca correta pelas colunas da tabela. Caso a linha não represente essa denominação, é inserida a linha em uma lista que, posteriormente, será analisada para abstrair o nome e tipo da coluna. Já o último condicional é responsável por verificar a última linha do *script*, já que posteriormente a essa última linha não existirá vírgulas. O nome da tabela é obtido a partir da busca pelas palavras *create table* e o primeiro parêntese após essas palavras, guarda entre si o nome da tabela. A Figura 23 ilustra o código responsável por inserir as colunas encontradas.

Figura 23 - Script responsável por inserir as colunas das tabelas encontradas

```
//inserindo as variriáveis
for (int x = 0; x < linhasEntitys.Count(); x++)
{
    string[] arrayLinha = linhasEntitys[x].Split(' ');
    if (arrayLinha[0] != "constraint") //pegando somente as variaveis
    {

        LineEntitie le = new LineEntitie();
        le.ID = Guid.NewGuid();
        le.IDEntities = ent.ID;
        le.Name = arrayLinha[0];
        le.TypeLineEntities = arrayLinha[1];
        if (linhasEntitys[x].Contains("primary key"))
        {
            le.TypeFieldEntities = "Chave Primária";
            le.Valide = "NÃO";
        }
        else
        {
            le.TypeFieldEntities = "Normal";
            le.Valide = "SIM";
        }
        //Adicionando na lista de variáveis
        le.RegisterDate = DateTime.Now;
        le.ModificationDate = DateTime.Now;
        db.LineEntities.Add(le);
        db.SaveChanges();
    }
}
```

Fonte: Próprio autor

Após encontrar as colunas, é preciso inseri-las no banco de dados para que o usuário possa visualizá-las ou excluí-las, caso necessário. A Figura 23 exibe o laço de repetição que lê a lista de linhas/colunas encontradas e analisa cada uma. Primeiramente, é criada outra lista que guardará todas as palavras da linha; o critério de criação desta lista é a existência de espaço em branco entre palavras. Além disso, é inserido um condicional que possibilita a análise somente de colunas, eliminando qualquer outra denominação do script, pois os mesmos serão analisados posteriormente. Para encontrar o nome da coluna, é buscada a primeira palavra da lista de palavras, como determinado pela linguagem SQL. Em seguida, é armazenado o tipo da variável, que representa a segunda palavra da lista de palavras e, posteriormente, verificado se a linha a ser inserida é uma chave primária ou secundária, denominando também sua validade para Análise de Pontos de Função. A Figura 24 apresenta a análise realizada nas definições de constantes do script DDL.

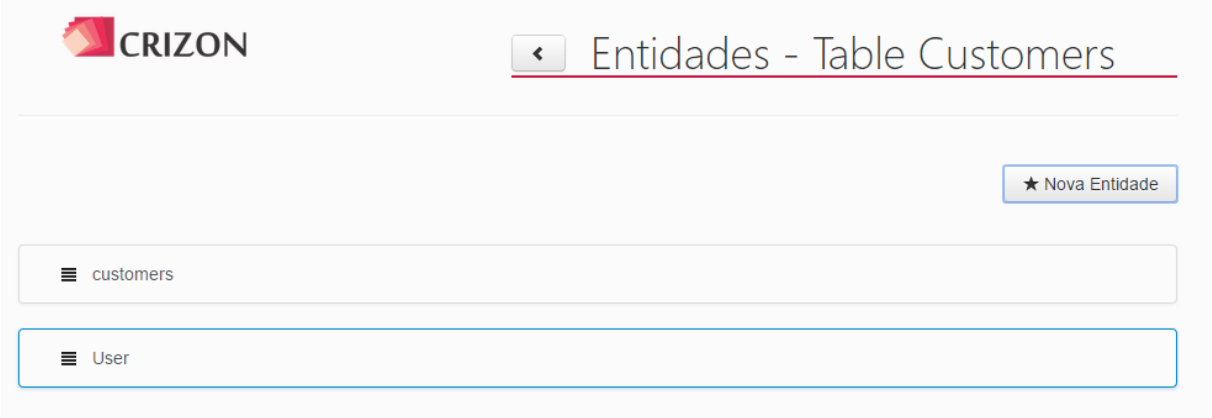
Figura 24 - Script de análise das constantes

```
//lendo as primary e foreign key
for (int x = 0; x < linhasEntitys.Count(); x++)
{
    string[] arrayLinha = linhasEntitys[x].Split(' ');
    if (arrayLinha[0] == "constraint") //pegando somente as variaveis constraint
    {
        foreach (var item in db.LineEntities.ToList())
        {
            if (linhasEntitys[x].Contains(item.Name))
            {
                if (linhasEntitys[x].Contains("primary key"))
                {
                    item.TypeFieldEntities = "Chave Primária";
                    item.Valide = "Não";
                }
                else if (linhasEntitys[x].Contains("foreign key"))
                {
                    item.TypeFieldEntities = "Chave Estrangeira";
                    item.Valide = "SIM";
                }
            }
        }
    }
}
```

Fonte: Próprio autor

A Figura 24 ilustra o código responsável por analisar as constantes existentes, na qual é realizado um laço de repetição entre as linhas desse *script*, buscando as constantes existentes, que podem determinar a existência de uma variável como chave primária ou secundária. Caso seja encontrada uma chave primária ou secundária, é alterado o tipo da variável, bem como sua validade de acordo com a técnica de Pontos de Função. A mesma lógica realizada nos processos anteriores responsáveis por analisar a criação de uma nova tabela, que é utilizada para analisar a alteração das tabelas podendo alterar o estado de uma variável. O resultado de todo esse processo de análise é gravado no banco de dados e exibido para o usuário na tela de listagem de tabelas, como ilustrado na Figura 25.

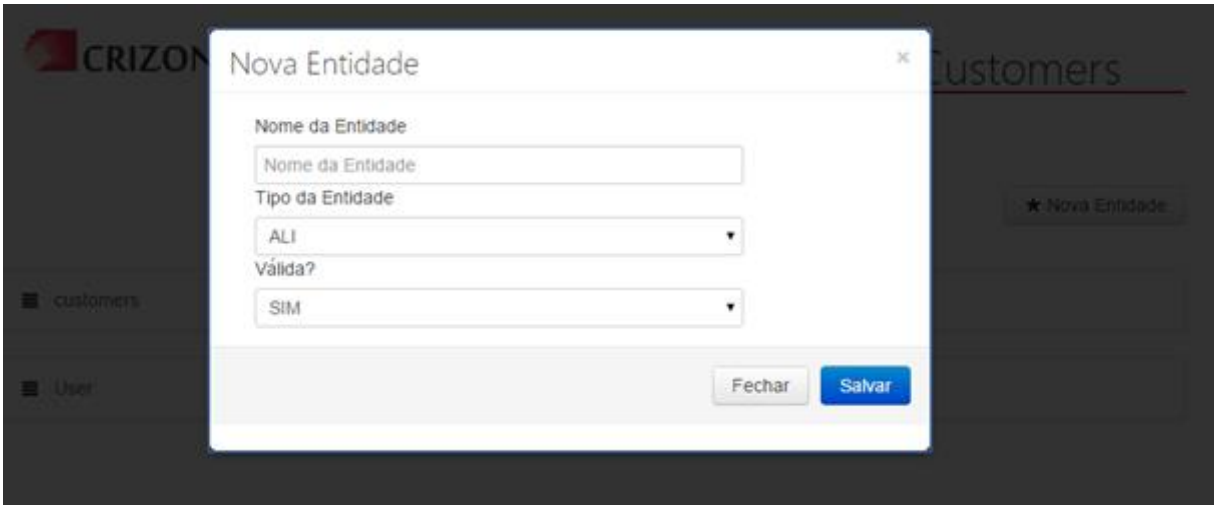
Figura 25 - Listagem de tabelas de uma interpretação



Fonte: Próprio autor

Essa listagem, ilustrada pela Figura 25, apresenta todas as tabelas encontradas e possibilita que o usuário insira uma tabela/entidade manualmente a partir do botão “Nova Entidade”. Essa ação pode ser realizada caso o usuário precise de uma tabela que não estava no *script* ou que surgir de uma nova necessidade. A Figura 26 ilustra a tela de criação de uma nova entidade/tabela.

Figura 26 - Tela de criação de uma nova entidade



Fonte: Próprio autor

Após acessar o botão, será exibida a tela de criação de uma nota entidade, como ilustrada na Figura 26. Essa tela possibilita ao usuário inserir o nome, tipo e validade dessa tabela de acordo com os conceitos de APF. Após o preenchimento dessas informações a tabela é gravada

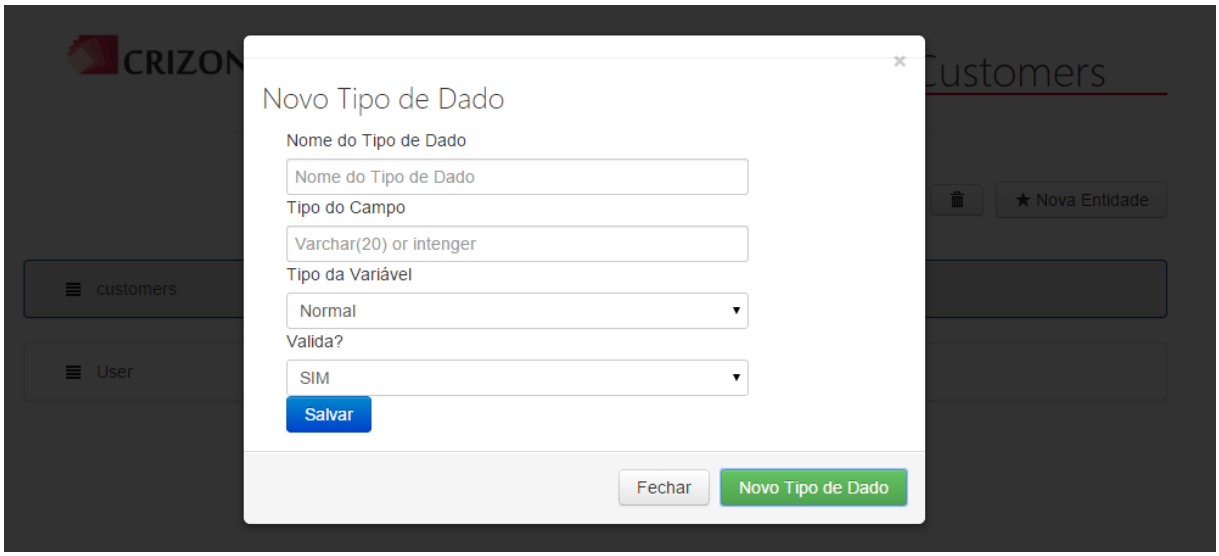
e listada juntamente com as demais. Para acessar as informações contidas na tabela, o usuário poderá pressionar um duplo clique em cima do nome da tabela ou clicar no nome da tabela e clicar no botão de visualizar com um ícone de um olho. Ao realizar essa ação é exibida a tela que apresenta as informações/colunas da tabela escolhida, como ilustrado na Figura 27.

Figura 27 - Tela de listagem dos tipos de dados/colunas da tabela

Fonte: Próprio autor

A tela de listagem das colunas, ilustrada pela Figura 27, exibe detalhadamente as colunas existentes, bem como seu tipo, campo e validade, além de possibilitar a inserção ou exclusão de uma coluna. Do mesmo modo da inserção de uma nova tabela, a inserção de uma nova coluna é possível desde que o usuário informe alguns dados necessários para análise, de acordo com conceitos da técnica de APF. A Figura 28 ilustra a tela de criação de uma nova coluna/tipo de dado.

Figura 28 - Tela de cadastro de uma nova coluna/tipo de dado



Fonte: Próprio autor

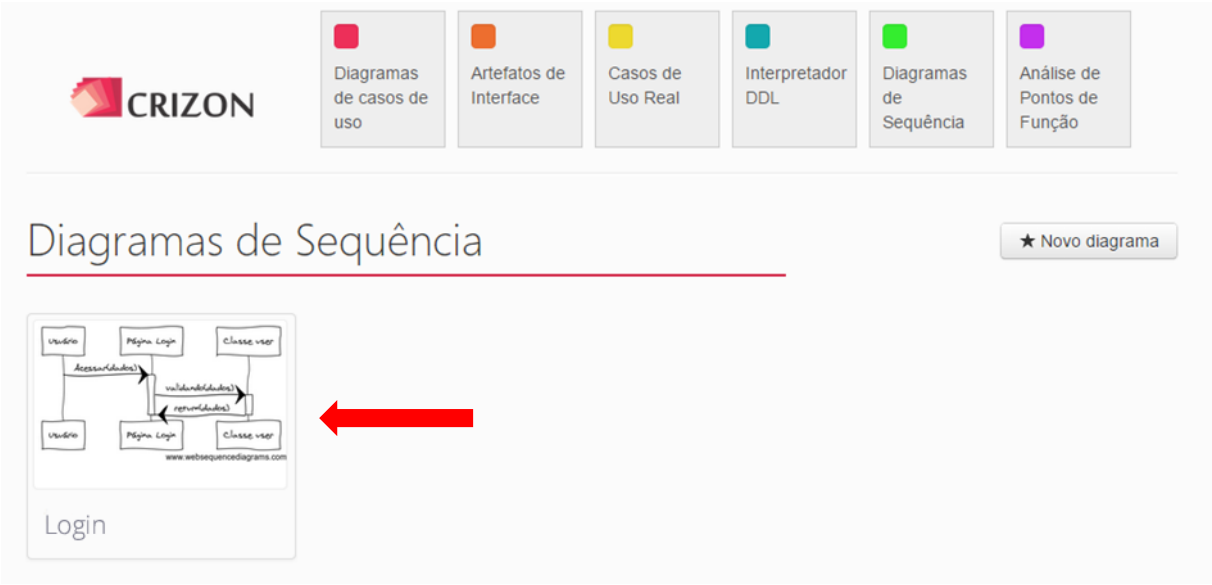
Caso necessite criar uma nova coluna, o usuário terá essa possibilidade preenchendo as informações necessárias, como ilustrado na Figura 28. Após o detalhamento do módulo responsável por realizar a interpretação do *script* DDL da SQL, é necessário apresentar o módulo de diagrama de sequência, responsável por possibilitar ao usuário construir seus diagramas de acordo com as funcionalidades existentes no software. Além disso, os diagramas utilizam como fonte de entrada as variáveis que participam da ação, que são providas do módulo de interpretação do *script* DDL. Na próxima seção (4.1.4), será apresentado detalhadamente o módulo de Diagrama de Sequência.

4.1.4. Módulo de Diagrama de Sequência

O Módulo de Diagrama de Sequência é responsável por possibilitar ao usuário construir diagramas de sequências que representem as funcionalidades existentes no software que será analisado. Esses diagramas, por sua vez, possibilitam mapear as transações existentes no sistema para, ao final, utilizar essas informações em prol da métrica do software, a partir da técnica de Análise de Pontos de Função. O usuário tem a possibilidade de criar quantos diagramas forem necessários para representar todas as ações existentes, podendo até mesmo

salvar as imagens dos diagramas, caso necessite. A Figura 29 apresenta a tela de listagem dos diagramas construídos.

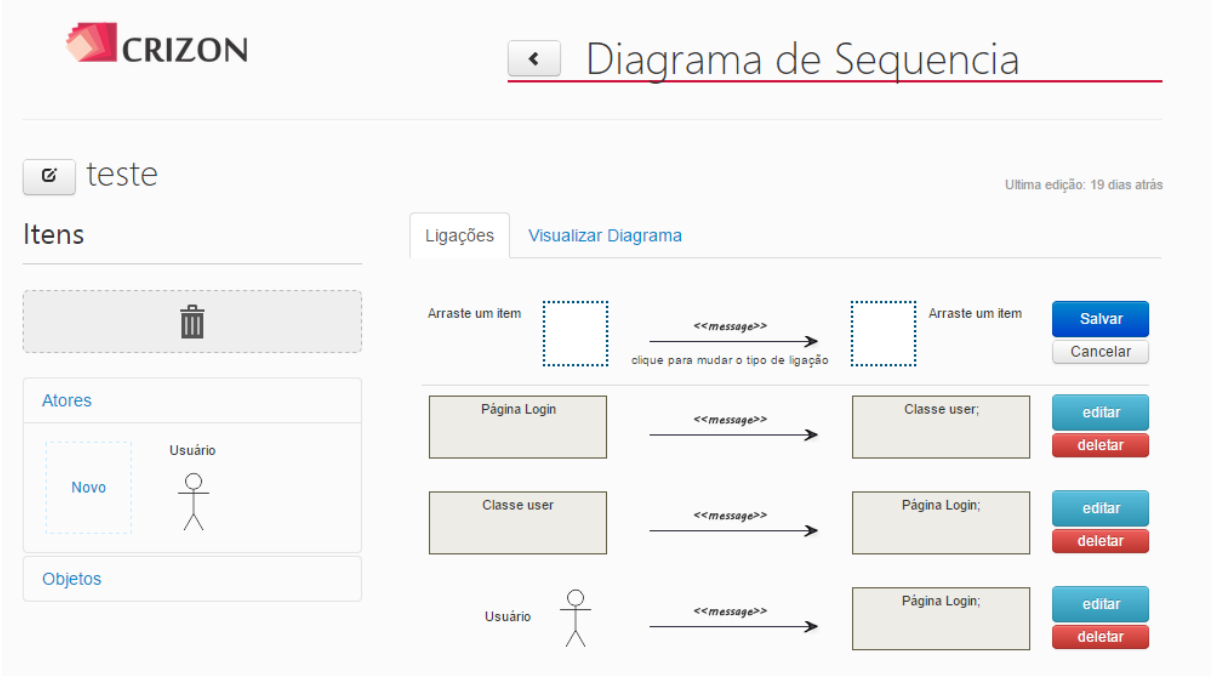
Figura 29 - Tela de listagem dos diagramas construídos



Fonte: Próprio autor

De acordo com a Figura 29, é possível observar a listagem dos diagramas, bem como uma imagem que ilustra o seu fluxo. Ao clicar em cima de um diagrama, são habilitados botões para excluir e visualizar o diagrama. Para criar um novo diagrama basta acessar o botão “Novo diagrama” presente na tela no canto superior direito. Ao criar um novo diagrama, o usuário poderá adicionar um nome ao diagrama, bem como visualizar e editar as informações contidas no diagrama, como é ilustrado na Figura 30.

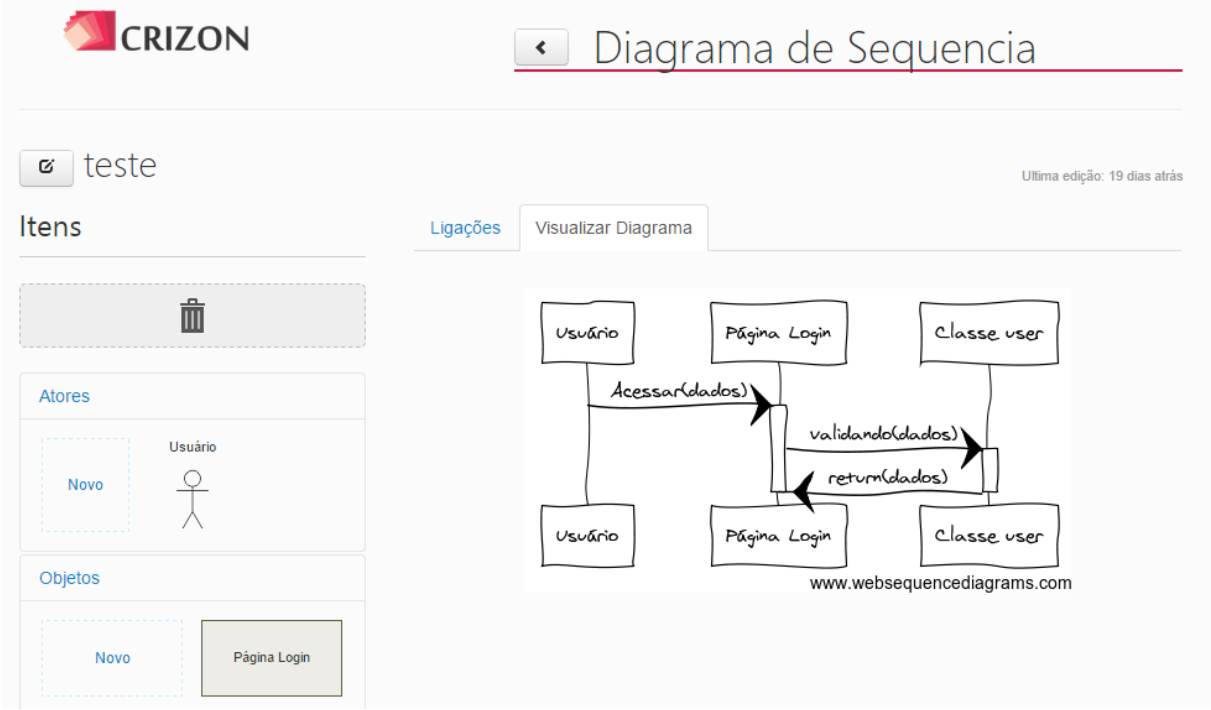
Figura 30 - Tela de edição do diagrama de sequência



Fonte: Próprio autor

A tela de edição, ilustrada pela Figura 30, possibilita a criação de atores que participam das ações existentes; dos objetos, na qual transitam as mensagens da ação; e Ligações, que representam o elo de comunicação entre um ator-objeto. Além disso, é possível visualizar o diagrama a qualquer momento, sua atualização é simultânea de acordo com a criação das ligações existentes, como ilustrado na Figura 31.

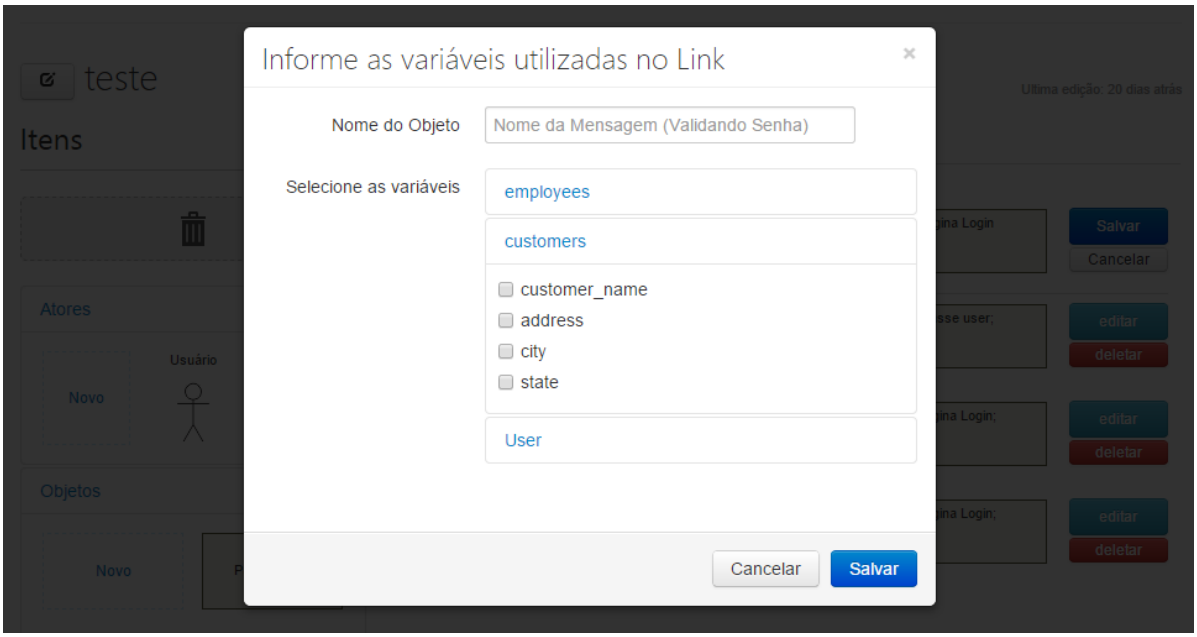
Figura 31 - Tela de visualização do diagrama de sequência



Fonte: Próprio autor

Com as ligações cadastradas, a imagem do diagrama é construída através de uma integração com a *Application Programming Interface* - API do *Web Sequence Diagrams*, sistema que possibilita a criação do diagrama de sequência em sua plataforma, além de disponibilizar um *webservice* para a construção de diagramas a partir de suas definições, retornando assim uma imagem de acordo com as ligações enviadas via requisição. Para criar uma nova ligação, é preciso informar primeiramente quem é o remetente, destinatário e o tipo de comunicação que está sendo criada nesta ligação, além de informar o nome da mensagem e, caso haja variáveis sendo transportadas na ligação, informar quais das variáveis são transportadas. Essas variáveis são obtidas a partir da interpretação do *script* DDL. A Figura 32 ilustra a tela de cadastro da nova ligação.

Figura 32 - Tela de cadastro de nova ligação



Fonte: Próprio autor

Após o preenchimento das informações da tela de cadastro, ilustrada na Figura 32, a ligação é gravada na base de dados e pode ser visualizada e editada. Por fim, as informações do módulo de interpretação do script DDL juntamente com o módulo do diagrama de sequência, são fontes de entrada para o módulo de análise de Pontos de Função, que analisará as informações obtidas desses módulos para determinar a quantidade de Pontos de Função do software a ser analisada. Para isso, a seção 4.1.5 apresenta a implementação do módulo de Análise de Pontos de Função.

4.1.5. Módulo de Análise de Pontos de Função

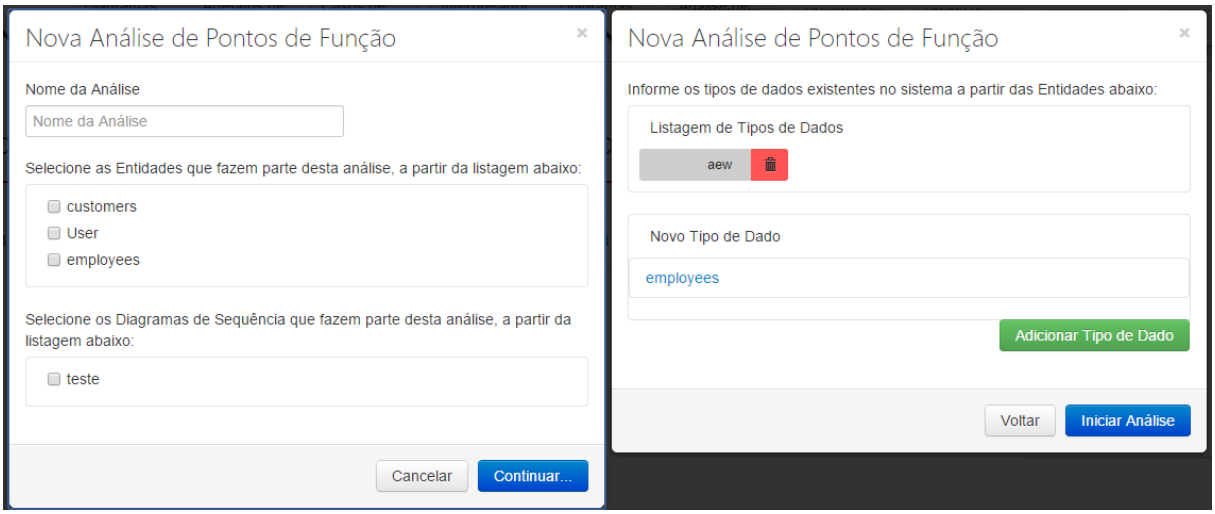
Com a conclusão da interpretação do *script* DDL e a construção dos diagramas de sequência é possível realizar a Análise de Pontos de Função do software. Para isso, é preciso somente acessar o menu, o qual será carregado com a lista de análises realizadas. A Figura 33 ilustra a tela de listagem de análises.

Figura 33 - Tela de listagem de Análises de Pontos de Função

Fonte: Próprio autor

Conforme a Figura 33, a tela de listagem de Análises de Pontos de Função possibilita a criação de uma nova análise a partir do botão “Nova Análise”, além de possibilitar visualizar e excluir as análises já realizadas. Para criar uma nova análise, é preciso preencher algumas informações necessárias para que seja possível realizar a análise mais correta do software. A Figura 34 ilustra a tela de cadastro de uma nova análise.

Figura 34 - Tela de cadastro de uma nova Análise de Pontos de Função



Fonte: Próprio autor

A partir da Figura 34, é possível observar os campos necessários para a realização da Análise de PF; o cadastro de uma nova análise é dividido em duas etapas, são elas:

- Etapa de seleção – é a etapa em que o usuário informa/seleciona quais entidades e diagramas que participaram da análise, bem como o preenchimento do nome a ser concedido a análise.
- Etapa de Tipo de Registro – é a etapa que o usuário informará, caso existam, os tipos de dados presentes nas entidades selecionadas na etapa anteriormente. Para isso, o usuário selecionará os tipos de dados/colunas que representam uma informação/tipo de registro e salvará.

Após essas etapas, o usuário acessará o botão “Iniciar Análise”, que inicia a Análise de Pontos de Função de acordo com os dados informados. Essa análise é dividida em duas fases, são elas:

- Fase de análise dos Dados – nesta fase são analisadas todas as entidades/tabelas e tipos de dados/colunas existentes, verificando suas validações perante os conceitos da técnica. Para isso, são realizados diversos procedimentos para, ao final, determinar os fatores que influenciam na determinação dos Pontos de Função do Software. O primeiro procedimento realizado é a criação de listas para guardar as entidades e tipos de dados selecionados para análise, verificando se os mesmos estão contidos nos diagramas selecionados e se há alguma entidade que esteja presente em um dos diagramas selecionados e não esteja presente nas entidades selecionadas; caso aconteça, é exibido um erro ao usuário. A Figura 35 ilustra o script de verificação das entidades/diagramas.

Figura 35 - Script de verificação dos diagramas e entidades selecionadas

```
//verificando se as entidades relacionadas pelas variáveis do diagrama
//são as mesmas entidades selecionadas para análise
List<Entitie> verifyEntities = new List<Entitie>();
foreach (var item in listaVariaveis)
{
    LineEntitie verifyLineEntities = db.LineEntities.Find(item);
    Entitie verifyEntitie = db.Entities.Find(verifyLineEntities.IDEntities);
    //caso a entidade da variavel não esteja contida dentro da lista da variável
    if (!Entidades.Contains(verifyEntitie)){
        //retorna -1 como erro na verificação
        return -1;
        messageError = "Você selecionou Entidades que não possuem Diagrama de Sequência. "
                    +"Por favor, constru-a-os para que a análise seja realizada corretamente!";
    }
}
```

Fonte: Próprio autor

Essa verificação, ilustrada pela Figura 35, é de grande importância para a consistência da análise, já que, se o usuário selecionar um diagrama que possui uma entidade envolvida na ação e não seleciona a entidade como parte da análise, há uma contradição entre os fatos, e a análise não poderá ser realizada. Outro procedimento realizado é validar todos os tipos de dados a partir das variáveis que tramitam nas ações dos diagramas selecionados; assim, tem-se as variáveis que são enviadas para o ator que, logo, são vistas pelo usuário, de acordo com os conceitos da técnica. Após essas verificações, os dados estão aptos a serem analisados e, ao final, emitir as complexidades e a quantidade de Pontos de Função desta análise.

A primeira contagem é provida dos Tipos de Dados (TD) válidos, que são dados/colunas que são vistas pelo usuário. Logo, para cada TD é atribuído um ponto que, ao final, ajuda a determinar a complexidade desta entidade, já que, cada entidade pode possuir vários TD. Em seguida, é realizada a contagem dos Tipos de Registro (TR), que são dois ou mais TD que representam uma única informação, como por exemplo: dia, mês e ano, que são três Tipos de Dados e representam uma informação (TR), que é uma data. A Figura 36 ilustra o código de verificação dos TR e TD.

Figura 36 - Script responsável por verificar os TR e TD das entidades

```
//Conta uma unidade para cada TD válido
relatorio.TD += linhaEntidades.Where(o => o.Valide.Equals("SIM")).Count();
//Conte uma unidade para cada Td Chave Estrangeira
relatorio.TD += linhaEntidades.Where(o => o.TypeLineEntities.Equals("Chave Estrangeira")).Count();

//Contando quantos Grupos possue na entidade
List<RegisterGroup> grupos = db.RegisterGroups.Where(o => o.IDEntities == item.ID).ToList();
//A partir da quantidade de grupos, denomina o valor de TR + 1
relatorio.TR += grupos.Count() == 0 ? 1 : grupos.Count() + 1;

//Contar 2 TD para cada tipo de registro
relatorio.TD += grupos.Count() > 0 ? grupos.Count() * 2 : 0;
```

Fonte: Próprio autor

É possível observar, a partir da Figura 36, o código responsável por quantificar os TD e TR existentes. Como determinado na técnica, é contado uma unidade de TR para cada entidade, mesmo que ela tenha ou não um ou mais TR; além de contar duas unidades de TD para cada TR existente. Além disso, é contado uma unidade de TD para cada chave estrangeira na entidade. Após essas contagens, o penúltimo procedimento realizado é a verificação da complexidade de cada entidade; essa verificação é realizada a partir da quantidade de TD e TR existentes. A Figura 37 ilustra o código responsável por determinar a complexidade das entidades.

Figura 37 - Script responsável por determinar a complexidade das entidades

```
//Definindo a complexidade
if (relatorio.TR == 1)
{
    if (relatorio.TD <= 50)
        relatorio.Complexidade = "Baixa";
    else
        relatorio.Complexidade = "Média";
}
else if (relatorio.TR <= 5)
{
    if (relatorio.TD < 20)
        relatorio.Complexidade = "Baixa";
    else if (relatorio.TD <= 50)
        relatorio.Complexidade = "Média";
    else //Complexidade > 50
        relatorio.Complexidade = "Alta";
}
else // TR > 5
{
    if (relatorio.TD < 20)
        relatorio.Complexidade = "Média";
    else
        relatorio.Complexidade = "Alta";
}
```

Fonte: Próprio autor

A partir da Figura 37, é possível observar que a complexidade é determinada primeiramente pela quantidade de TR existente em cada entidade e, em seguida, pela quantidade de TD existente. Essa complexidade é de grande importância, pois é a partir dela que a quantidade de PF será determinada. Após essa determinação, o procedimento final é inferir a quantidade de PF para cada entidade. A Figura 38 ilustra o código responsável por essa inferência.

Figura 38 - Script responsável por inferir a quantidade de Pontos de Função

```
//Verificar a quantidade de PF
if (item.TypeEntities.Equals("ALI"))
{
    if (relatorio.Complexidade.Equals("Baixa"))
        relatorio.PF = 7;
    else if (relatorio.Complexidade.Equals("Média"))
        relatorio.PF = 10;
    else if (relatorio.Complexidade.Equals("Alta"))
        relatorio.PF = 15;
}
else // AIE
{
    if (relatorio.Complexidade.Equals("Baixa"))
        relatorio.PF = 5;
    else if (relatorio.Complexidade.Equals("Média"))
        relatorio.PF = 7;
    else if (relatorio.Complexidade.Equals("Alta"))
        relatorio.PF = 10;
}
```

Fonte: Próprio autor

É possível observar, a partir da Figura 38, que a quantidade de Pontos de Função é determinada primeiramente pelo tipo da entidade, que representa se a entidade contém informações providas de outro sistema ou não. Após essa verificação, a inferência dos PF é realizada pela sua complexidade, que pode ser de Baixa, Média ou Alta. Por fim, as informações geradas nesta análise são gravadas na base de dados para poderem ser consultadas a partir do relatório.

- Fase de análise das Transações – essa fase é responsável por determinar a quantidade de Pontos de Função que existe no software a partir das suas transações, que nada mais é do que as ações/diagramas cadastradas no sistema. Primeiramente, são realizados procedimentos para determinação da quantidade de TD e AR existente em cada

diagrama. Um desses procedimentos é a verificação da existência de mensagens de erros, pois a existência das mesmas é somada uma unidade de TD para cada mensagem, como de acordo com os conceitos da técnica. Além disso, é somada uma unidade de TD para cada variável/coluna que tramita no diagrama, necessitando que a mesma seja uma variável válida. A Figura 39 ilustra o código responsável por realizar esses procedimentos.

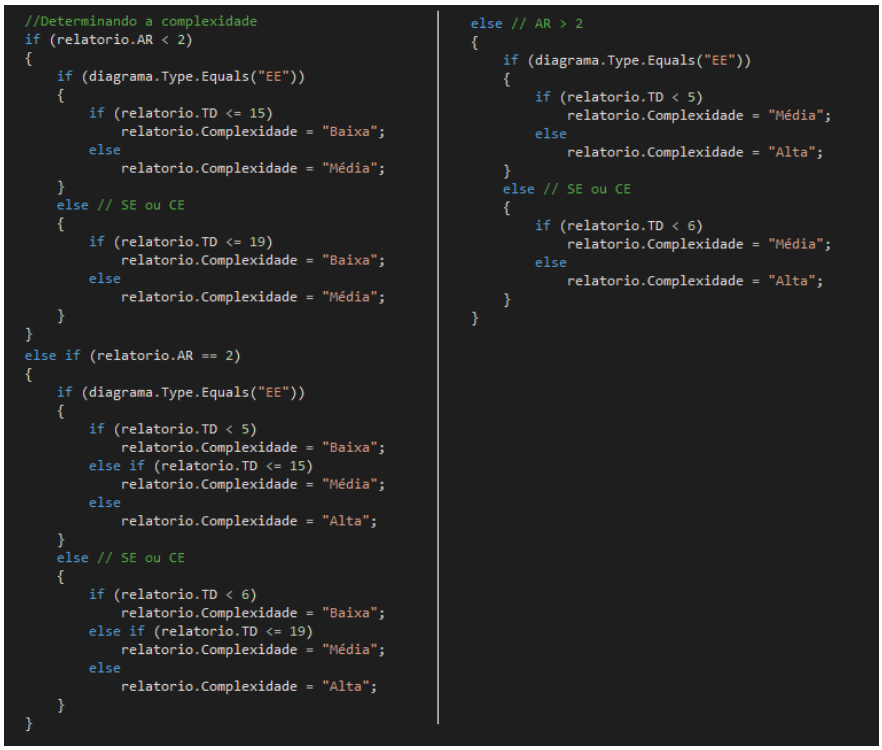
Figura 39 - Script responsável pela determinação dos TD e AR

```
//Verificando as Linhas de comunicação do diagrama
foreach (LineDiagramSequence linha in linesDiagram.Where(o => o.TypeCommunication != 1 && o.TypeCommunication != 5))
{
    // Se o tipo de comunicação for Mensagem de Erro
    if (linha.TypeCommunication == 4)
        relatorio.TD++;

    //Obtendo as variaveis utilizadas na linha de comunicação
    string[] IDLineEntities = linha.Variables.Split(';');

    foreach (var linesEntities in IDLineEntities)
    {
        //Buscando a Linha da Entidade
        LineEntitie line = db.LineEntities.Find(new Guid(linesEntities));
        //verificando se o campo é valido (visivel pelo usuário)
        if (line.Valide.Equals("SIM"))
            relatorio.TD++;
        //Verificando se sua Entidade já está presente na lista de entidades
        if (!EntitiesExistent.Contains(line.IDEntities.ToString()))
            //caso não esteja presente, é preciso adiciona-la!
            EntitiesExistent.Add(line.IDEntities.ToString());
    }
}

//Contando AR
relatorio.AR = EntitiesExistent.Count();
```

Fonte: Próprio autor

É possível observar, de acordo com a Figura 39, a contagem de uma unidade de AR para cada entidade existente no diagrama. Cada entidade só pode ser contada uma única vez, para isso, é verificada a sua existência na lista de entidades relacionadas. Após esses procedimentos, é possível determinar a complexidade da transação/diagrama de acordo com os valores inferidos para TD e AR. A Figura 40 ilustra o código responsável pela inferência da complexidade, de acordo com os conceitos da técnica.

Figura 40 - Script responsável por determinar a complexidade dos diagramas



```
//Determinando a complexidade
if (relatorio.AR < 2)
{
    if (diagrama.Type.Equals("EE"))
    {
        if (relatorio.TD <= 15)
            relatorio.Complexidade = "Baixa";
        else
            relatorio.Complexidade = "Média";
    }
    else // SE ou CE
    {
        if (relatorio.TD <= 19)
            relatorio.Complexidade = "Baixa";
        else
            relatorio.Complexidade = "Média";
    }
}
else if (relatorio.AR == 2)
{
    if (diagrama.Type.Equals("EE"))
    {
        if (relatorio.TD < 5)
            relatorio.Complexidade = "Baixa";
        else if (relatorio.TD <= 15)
            relatorio.Complexidade = "Média";
        else
            relatorio.Complexidade = "Alta";
    }
    else // SE ou CE
    {
        if (relatorio.TD < 6)
            relatorio.Complexidade = "Baixa";
        else if (relatorio.TD <= 19)
            relatorio.Complexidade = "Média";
        else
            relatorio.Complexidade = "Alta";
    }
}
else // AR > 2
{
    if (diagrama.Type.Equals("EE"))
    {
        if (relatorio.TD < 5)
            relatorio.Complexidade = "Média";
        else
            relatorio.Complexidade = "Alta";
    }
    else // SE ou CE
    {
        if (relatorio.TD < 6)
            relatorio.Complexidade = "Média";
        else
            relatorio.Complexidade = "Alta";
    }
}
```

Fonte: Próprio autor

A partir da Figura 40, é possível observar que a determinação da complexidade dos diagramas é inferida, primeiramente, pela verificação da quantidade de AR existente e, em seguida, pela quantidade de TD existente, para assim inferir a complexidade de cada diagrama. Após a determinação da complexidade, é possível mensurar a quantidade de Pontos de Função para cada diagrama. A Figura 41 ilustra o código responsável por essa inferência.

Figura 41 - Script responsável por determinar a quantidade de Pontos de Função dos diagramas

```
//Determinando PF
if (diagrama.Type.Equals("EE"))
{
    if (relatorio.Complexidade.Equals("Baixa"))
        relatorio.PF = 3;
    else if (relatorio.Complexidade.Equals("Média"))
        relatorio.PF = 4;
    else
        relatorio.PF = 6;
}
else if (diagrama.Type.Equals("SE"))
{
    if (relatorio.Complexidade.Equals("Baixa"))
        relatorio.PF = 4;
    else if (relatorio.Complexidade.Equals("Média"))
        relatorio.PF = 5;
    else
        relatorio.PF = 7;
}
else //CE
{
    if (relatorio.Complexidade.Equals("Baixa"))
        relatorio.PF = 3;
    else if (relatorio.Complexidade.Equals("Média"))
        relatorio.PF = 4;
    else
        relatorio.PF = 6;
}
```

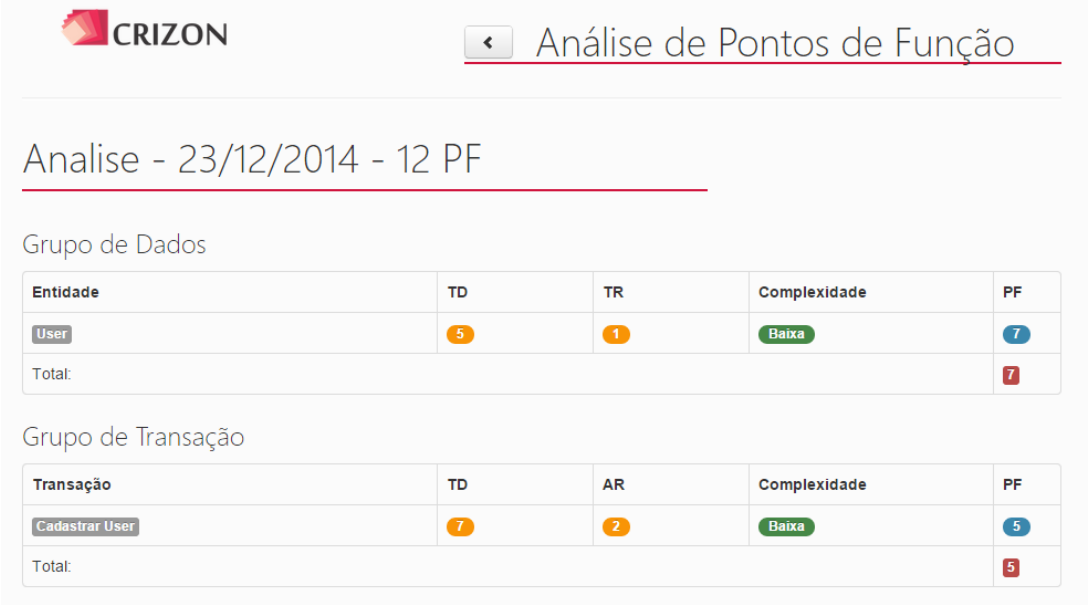
Fonte: Próprio autor

A determinação da quantidade de Pontos de Função, ilustrada pela Figura 41, é realizada a partir de duas verificações, são elas:

- Verificação do Tipo de Transação - cada diagrama representa uma ação, que pode ser caracterizada entre uma Entrada Externa (EE), que representa por exemplo um cadastro onde o usuário insere as informações para o sistema; Saída Externa (SE), que consiste em um processo que tem como resultado a extração de dados da aplicação; e, Consulta Externa (CE), que denota um processamento a partir da requisição de dados do meio externo para exibição imediata dos dados. Esses tipos de transação são de grande importância para a técnica, e é a partir desse tipo que a primeira verificação é realizada para, ao final, determinar a quantidade de Pontos de Função.
- Verificação da Complexidade – é verificada a complexidade de cada transação/diagrama, que foi determinada anteriormente a partir da quantidade de TD e AR de cada diagrama. De acordo com o tipo de transação e o grau de complexidade, é inferido um valor que determina os Pontos de Função da transação/diagrama.

Após o processamento das duas verificações, tem-se a quantidade de Pontos de Função de cada transação/diagrama analisado, podendo assim, ser realizada a verificação da quantidade de Pontos de Função geral do software. Essa contagem final é realizada somando a quantidade de Pontos de Função de todas as entidades e transações/diagramas analisados, gerando um relatório com cada inferência realizada. A Figura 42 ilustra o relatório gerado pelos resultados da análise.

Figura 42 - Relatório da análise realizada



Fonte: Próprio autor

Na Figura 42, é possível observar os dois tipos de inferências realizados, são eles: Grupo de Dados, que representa todas as Entidades/Tabelas existentes, válidas e analisadas pelo módulo, juntamente com a quantidade de TD, TR, complexidade e quantidade de PF; e, Grupo de Transação, que representa as transações existentes, analisadas a partir dos diagramas de sequências cadastrados pelo usuário, juntamente com o quantitativo dos índices de TD, AR, PF e complexidade. Finalizando a implementação de todos os módulos, é necessário realizar testes de verificação, observando o quanto próximo da análise manual o sistema chega, e se a análise realizada pelo sistema exige menos tempo em comparação com a manual. A seção 4.5 apresenta detalhadamente o processo de testes de verificação dos módulos desenvolvidos.

4.2. Testes de Verificação

Após o desenvolvimento dos módulos propostos neste trabalho, detalhados nas seções anteriores, iniciou-se a etapa de testes de verificação da ferramenta. Para tanto, foram realizadas duas análises, sendo uma sobre um software escolar, que possibilita o cadastro das escolas, alunos, professoras e disciplinas; e outra sobre um software bibliotecário, que possibilita o controle das reservas, livros, clientes e autores de livros. As análises foram realizadas a partir da ferramenta desenvolvida, construindo os diagramas de sequências de cada funcionalidade existente, juntamente com as entidades inferidas pelo interpretador de *script* DDL do banco de dados. Essa análise realizada por meio dos módulos possibilitou a automatização do processo de análise por meio da interpretação da estrutura da base de dados e de um artefato da modelagem de software.

No processo de teste de verificação e com o intuito de verificar os pontos fracos, bem como os benefícios e o grau de automatização da ferramenta desenvolvida, foram realizados testes, comparando análises realizadas pelo processo manual pelas obtidas pela ferramenta. As análises realizadas são apresentadas nesta seção e foram desenvolvidas de acordo com algumas regras criadas (e apresentadas a seguir), que buscaram obter um resultado de teste de verificação mais coeso possível. As regras serão detalhadas a seguir:

- Cada participante realizará duas análises - sendo uma feita de forma manual e outra através da ferramenta;
- Cada análise de um mesmo participante será feita sobre um software diferente - cada participante analisará dois softwares distintos: um no processo manual e outro no processo utilizando a ferramenta; entretanto, com a quantidade de Pontos de Função semelhante, pois entende-se que, se o participante analisar o mesmo software nas duas etapas do teste, o mesmo iniciará a segunda etapa com uma bagagem de conhecimento

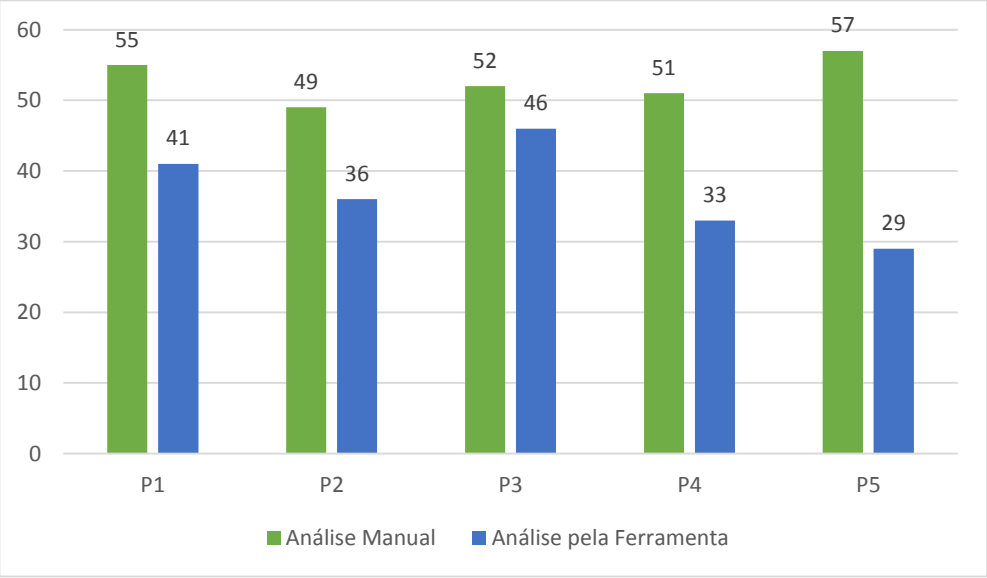
da etapa anterior e terá uma maior facilidade na realização da segunda etapa, influenciando assim no êxito do teste de verificação;

- Cada participante terá até uma hora por análise;
- Serão dez participantes;
- Ordem aleatória de análise - a escolha do tipo do processo, manual ou a partir do sistema, será determinada aleatoriamente na hora da realização do teste;
- É necessário que todos os participantes sejam da área de Tecnologia da Informação – TI, além de conhecerem o processo de modelagem e desenvolvimento de software;
- A metade dos participantes não terá conhecimento sobre a técnica – sendo necessário realizar uma explanação para que esses participantes tenham o conhecimento mínimo para realizar a análise. Essa divisão entre os participantes foi determinada para que o teste de verificação possa verificar a facilidade ou não que um usuário da área de TI, mesmo com conhecimento superficial sobre a técnica, terá em realizar a análise a partir do software;
- Cada etapa da análise será cronometrada - tanto para processo manual, quanto para o processo realizado pela ferramenta. A contabilização foi definida para mensurar o tempo gasto em cada etapa de cada fase para, ao final, verificar qual dos processos é realizado com maior agilidade.
- Cada etapa da análise será documentada - o processo manual será realizado a partir do formulário desenvolvido (Apêndice I), que discrimina cada inferência realizada, bem como os Pontos de Função do software analisado. Já o processo realizado pela ferramenta será contabilizado a partir do formulário simplificado (Apêndice II), juntamente com formulário emitido pela ferramenta, que demonstra as inferências realizadas;

Os participantes selecionados para realização do teste de verificação foram determinados pelos critérios citados acima, tendo como participantes funcionários da Fábrica de Software e acadêmicos do Centro Universitário Luterano de Palmas e servidores do setor de TI do Tribunal de Contas do Estado do Tocantins. Não houve determinação do local de realização dos testes, já que os participantes são de locais distintos, por isso, foram realizados em data, horário e locais definidos pelos participantes.

Após a realização dos testes, foi possível determinar algumas visões dos testes realizados, bem como um comparativo entre as análises realizadas manualmente e pela ferramenta. A primeira visão exibe o comparativo entre as análises realizadas pelos participantes que possuíam conhecimento sobre a técnica, como apresentado na Figura 43.

Figura 43 - Comparativo entre as análises realizadas por participantes com conhecimento da técnica.



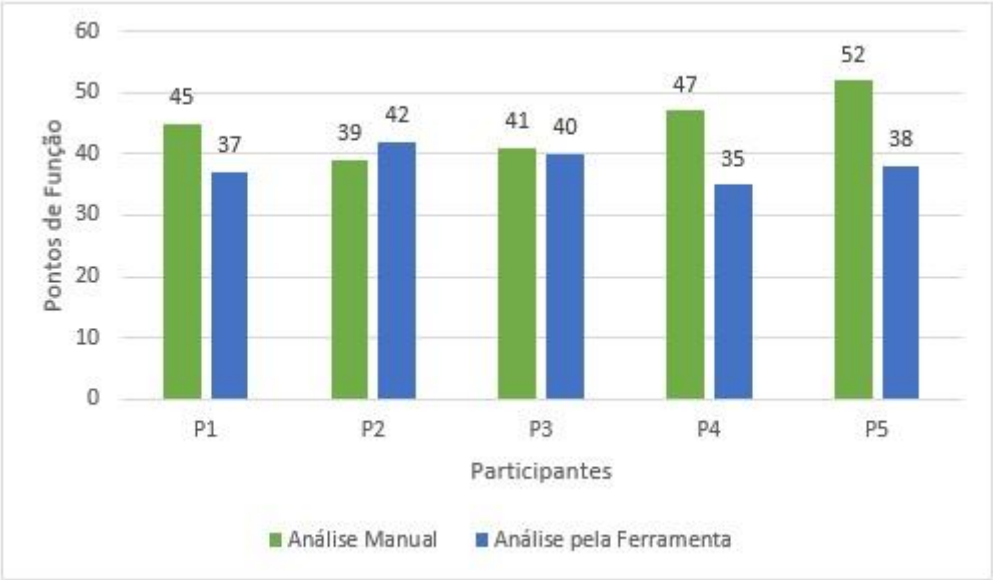
Fonte: Próprio autor

É possível observar, a partir da Figura 43, no eixo ‘x’ os participantes com as duas análises realizadas, e no eixo ‘y’ os minutos para realização das análises. O tempo necessário para realizar as análises manuais se mantiveram quase constantes na casa dos cinquenta minutos, tendo um tempo médio para realização de 52,8 minutos. Já as análises realizadas na

ferramenta apresentam uma média de 37 minutos, tendo uma diferença de aproximadamente 30% a menos do tempo gasto na análise manual. Além disso, espera-se que o tempo gasto na análise realizada na ferramenta seja menor ainda, pois os participantes não tinham conhecimento da ferramenta e gastaram alguns minutos para se familiarizar, tempo não necessário quando um usuário já tiver familiarizado com a ferramenta.

Outra visão possível é o comparativo entre os participantes sem conhecimento da técnica, que receberam treinamentos e explanações sobre a técnica de Análise de Pontos de Função, para melhor compreensão na realização das análises. A Figura 44 apresenta o comparativo entre as análises realizadas por participantes sem conhecimento da técnica.

Figura 44 - Comparativo entre as análises realizadas por participantes sem conhecimento da técnica.



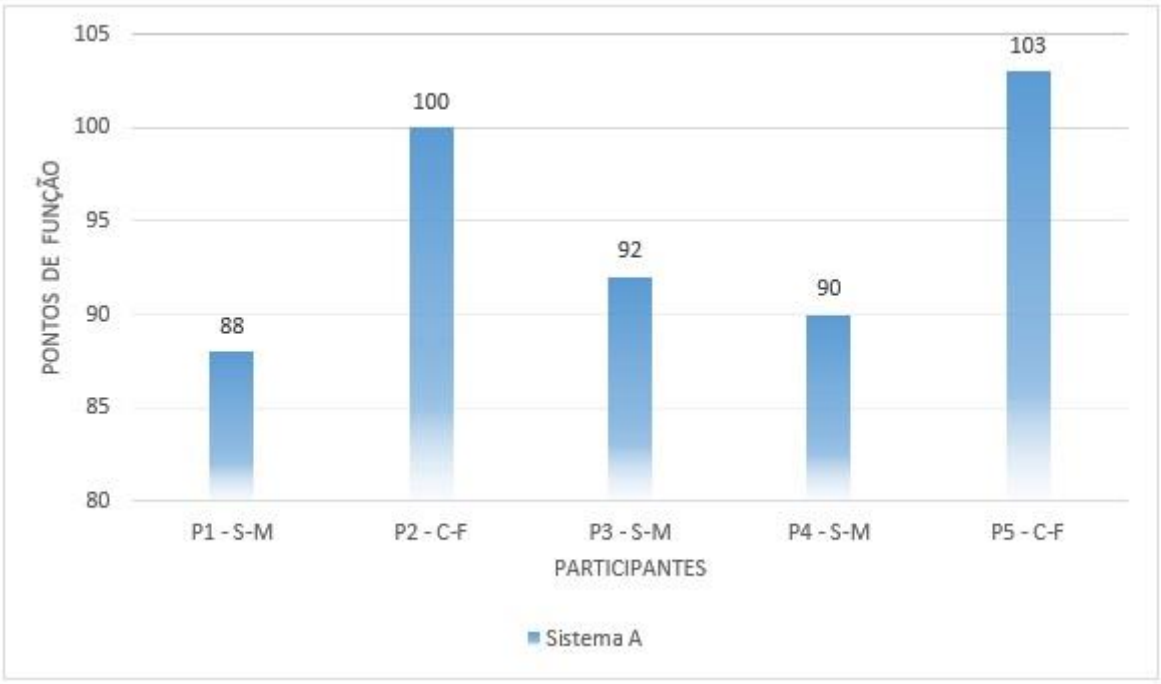
Fonte: Próprio autor

O comparativo entre as análises realizadas por participantes sem conhecimento da técnica, ilustrado pela Figura 44, possibilita observar a variação entre o tempo de realização das análises, tendo uma média de 45,8 minutos para análise realizada manualmente e 38,4 minutos para a análise realizada na ferramenta, tendo uma diferença de aproximadamente 16% menor do tempo gasto na análise manual.

Além disso, é possível perceber a diferença entre o tempo de realização das análises realizadas por participantes com e sem conhecimento sobre a técnica. Essa diferença chega a 7 minutos a mais entre análises realizadas por participantes com conhecimento, em comparativo com os participantes sem conhecimento, nas análises realizadas manualmente. Já nas análises realizadas na ferramenta a diferença chega a 1,4 minutos a mais entre análises realizadas por participantes sem conhecimento, em comparativo com participantes que possui o conhecimento. Essa diferença pode ser entendida pelo fato de que a ferramenta não exige um conhecimento sólido sobre a técnica, quanto na análise manual. Assim, os participantes que dominavam a técnica de Análise de Pontos de Função também dominavam a modelagem de sistemas e artefatos de software, o que facilitou na hora de realizar a mensuração realizada na ferramenta.

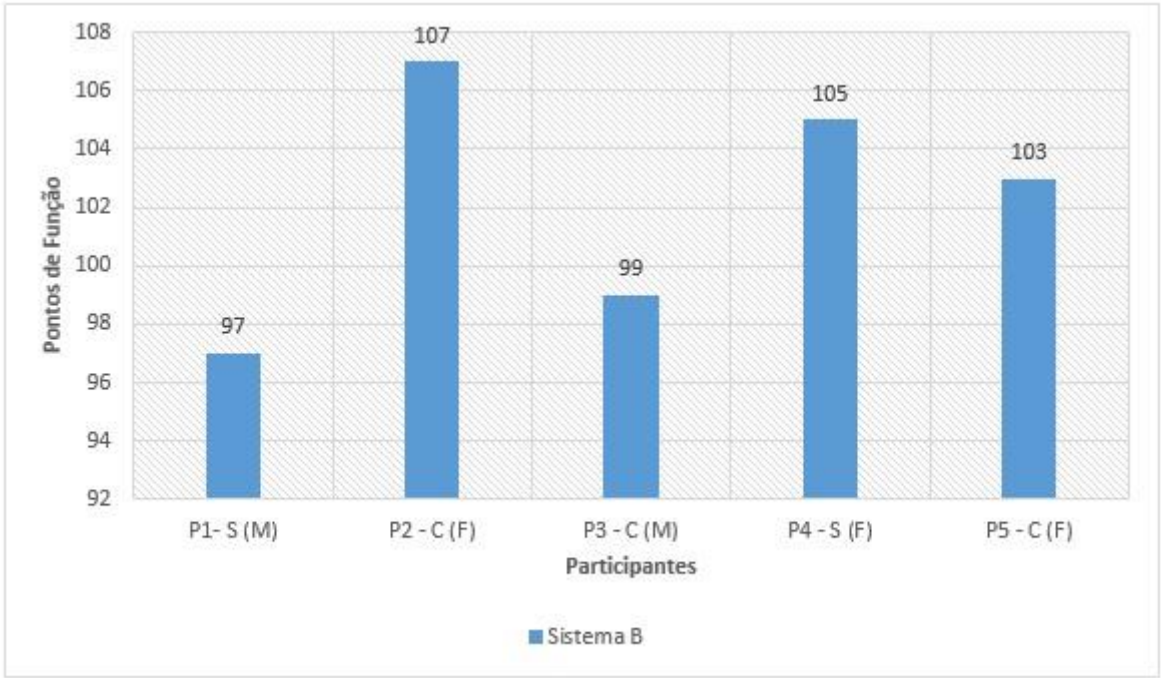
No comparativo entre os resultados das análises, tanto dos participantes com conhecimento, quanto os sem, pode-se observar uma variante entre os PF resultantes. A Figura 45 e Figura 46 ilustram o comparativo entre Pontos de Função resultantes de análises realizadas manualmente e na ferramenta, tanto com participantes que possuem conhecimento sobre a técnica, quanto os que não possuem.

Figura 45 - Comparativo entre os Pontos de Função das análises realizadas por participantes com e sem conhecimento a partir do Sistema A.



Fonte: Próprio autor

Figura 46 - Comparativo entre os Pontos de Função das análises realizadas por participantes com e sem conhecimento a partir do Sistema B.



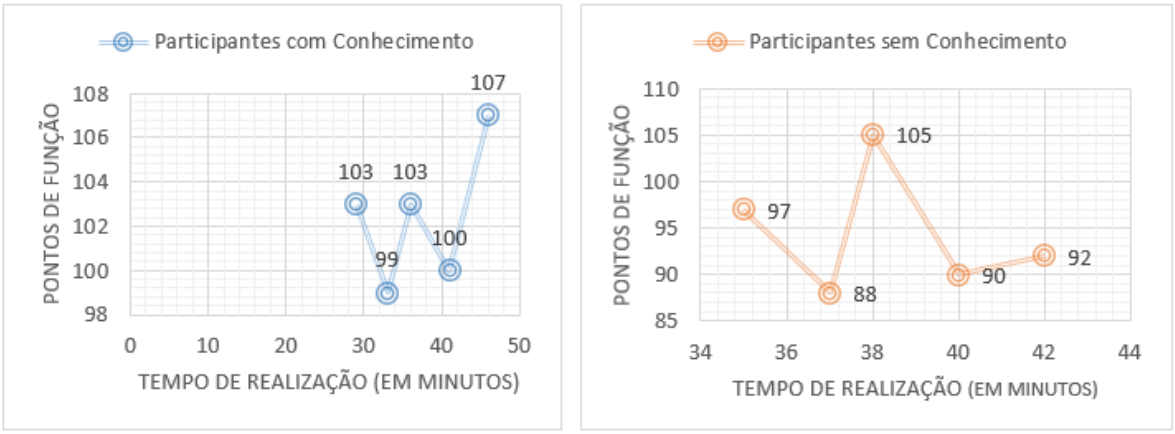
Fonte: Próprio autor

É possível observar, a partir da Figura 45 e Figura 46, no eixo ‘x’ os participantes com as duas análises realizadas, e no eixo ‘y’ a quantidade de pontos de função. Além disso, a descrição do participante apresentada nas figuras acima também ilustra se o participante possuí ou não conhecimento sobre a técnica, de acordo com a legenda “C” para com conhecimento ou “S” para sem conhecimento, e qual tipo de análise foi realizada, a partir da descrição (M) para manual e (F) para realizada na ferramenta, entre os parênteses descritos na legenda. É possível observar uma grande variação entre os resultados em Pontos de Função, essa variação pode ser entendida muitas vezes pelo esquecimento de alguma Entidade/Transação na realização da contagem de Pontos de Função, ou até mesmo por não mensurar variáveis nas transações. Essas variações ocorrem também pelo fato de que cada usuário tem uma visão distinta dos demais usuários (mesmo utilizando uma descrição detalhada do sistema), essa visão influencia diretamente na contagem, alterando o seu resultado final. Ou seja, nem sempre o mesmo sistema analisado por pessoas distintas resulta em Pontos de Função iguais; entretanto, esses resultados tendem a serem semelhantes, já que a análise foi realizada sobre o mesmo sistema.

É importante destacar que ao realizar a análise pela ferramenta, o usuário constrói o diagrama de sequência, o que faz com que exista uma variação entre as funções analisadas manualmente e no sistema, pois construindo o diagrama é possível entender e visualizar a funcionalidade como ela realmente é, e não imaginar como seria a funcionalidade a partir de uma descrição, como ocorre na contagem manual.

Outro comparativo possível é entre os pontos de função e tempo de realização da análise a partir dos participantes com e sem conhecimento sobre a técnica de Pontos de Função, ilustrado na Figura 47.

Figura 47 - Comparativo entre pontos de função e tempo de realização da análise a partir dos participantes com e sem conhecimento sobre a técnica



Fonte: Próprio autor

É possível observar, a partir da Figura 47, a pequena variação entre os pontos de função das análises realizadas com participantes com conhecimento que, comparado com os participantes sem conhecimento, possui uma variação um pouco maior. Já o tempo de realização das análises também é pouco variante entre os participantes com conhecimento, mas entre os participantes sem conhecimento obteve-se uma variação maior. Entretanto, essa variação entre o tempo comprova que os participantes com conhecimento sobre a técnica de Pontos de Função conseguem realizar a mensuração no software mais rápido, por possuir um conhecimento prévio. Na variação entre os pontos de função, percebe-se que os tempos variam, pois cada participante tem uma visão distinta ao realizar a análise de um software, possibilitando assim uma variação entre os Pontos de Função resultantes. Essa variação de PF não determina que as análises estão incorretas, somente indica que cada usuário tem uma visão diferente do mesmo software, ou seja, como não houve nenhuma análise com uma grande variação, todas as análises realizadas estão de acordo com o software proposto.

5. Considerações finais

A determinação do tamanho de um software ou o valor real que será cobrado pelas funcionalidades existentes é um processo difícil, pois é preciso trazer à tona fatores que influenciam diretamente no custo do desenvolvimento como, por exemplo: mão de obra, planejamento, modelagem, testes, riscos, dentre outros fatores.

O presente trabalho teve como objetivo desenvolver três módulos que possibilitam a mensuração do tamanho funcional dos softwares a partir da técnica de análise de Pontos de Função, utilizando como entrada os Diagramas de Sequências e o *script* DDL do banco de dados. Estes dois artefatos foram escolhidos por possibilitar a identificação dos dois grupos existentes no software, são eles:

- Grupos de Dados - representa a estrutura da base de dados que foi utilizada; para isso, foi utilizado o *script* DDL para determinação deste grupo, pois esse *script* é composto por códigos da linguagem SQL que representam a estrutura da base de dados do software, além disso, o *script* é utilizado para construção de bancos de dados que utilizam a sintaxe SQL, representando sua estrutura em forma escrita.
- Grupos de Transações - consistem no mapeamento das funcionalidades ou ações realizadas pelo software; para isso, foram utilizados os Diagramas de Sequência criados pelo usuário para determinação deste grupo, pois esse diagrama descreve a interação de um ator com a interface de usuário e interface com todas as camadas do sistema, além disso, o diagrama pertence a linguagem de modelagem da *Unified Modeling Language* (UML), que é um padrão de modelagem universal utilizada em muitas metodologias de modelagem de software.

Ao final de cada etapa listada acima resulta um valor em Pontos de Função, que são somados para definir o tamanho funcional do software como um todo. Assim, obteve-se o tamanho funcional do software analisado.

O objetivo desta ferramenta foi automatizar o processo que, anteriormente, era realizado de modo manual, necessitando muito tempo para sua conclusão. Outro problema com relação ao processo manual era que, a cada alteração, precisava-se realizar toda a contagem novamente. Agora, com a ferramenta, a análise pode ser alterada e recalculada em poucos cliques e quantas vezes for preciso, além de que a mesma é realizada a partir de um artefato de modelagem de software e da análise da estrutura do banco de dados. Ou seja, o processo que anteriormente era manual evoluiu para um processo automatizado que, a partir de artefatos de modelagem que já são desenvolvidos pela equipe para documentar o desenvolvimento do software, facilita a mensuração e elimina custos e tempo do processo de análise manual.

Algumas diferenças podem ser observadas em relação às outras ferramentas citadas nos trabalhos correlatos como, por exemplo, a utilização de um interpretador do *script* DDL para determinar os Pontos de Função dos Grupos de Dados, comparando com as ferramentas de Batista *et al.* (*online*, 2011) e Uemura *et al.* (2001) que necessitam que o usuário construa modelos e/ou diagramas para realizar essa determinação. Outra diferença é que o processo de análise de Pontos de Função realizado pela ferramenta desenvolvida neste trabalho não determina a necessidade da utilização de nenhuma metodologia, como é o caso das ferramentas de Pinel (2012) e Fraternali *et al.* (2006), que determinam que o especialista em Pontos de Função necessita utilizar uma metodologia de desenvolvimento escolhida pelos autores.

Como a proposta da ferramenta CRIZON v.2 é a análise a partir de artefato de modelagem e *script* da estrutura de dados do banco, a mensuração é realizada antes do desenvolvimento do software ou até mesmo em softwares já desenvolvidos, ao contrário da ferramenta de Pinel (2012), que é utilizada para contagens a partir de softwares já desenvolvidos. Já a ferramenta desenvolvida por Fraternali *et al.* (2006) necessita para o processo de mensuração a utilização da metodologia *WebML*, que é orientada para aplicações Web, não sendo adequada para aplicações *desktop*, *mobile* e entre outros. Já a CRIZON v.2

pode ser utilizada tanto para aplicações web, *mobile* ou *desktop,* e possui somente duas fases de análises, sendo uma automatizada pelo analisador.

Em contrapartida, um dos benefícios que pode ser apontado com essa ação é o tempo gasto na correção de um erro, na qual esse tempo na ferramenta é menor em comparação com a fase manual, pois não necessita realizar a contagem e analisar a funcionalidade novamente, já que no sistema o diagrama pode ser alterado quando for necessário e recalculado a qualquer momento. Além disso, ao construir os diagramas, o usuário tem a visão de como realmente executará a funcionalidade a partir das comunicações entre ator/objeto. Já na fase manual, essa visão não acontece e o usuário precisará recorrer ao papel para descrever uma ideia, que talvez não refletirá a funcionalidade real da ação. Ou seja, no processo manual o usuário não tem aparatos para determinar as transações e precisa buscar meios que ajudem a determiná-las; no sistema as funcionalidades são analisadas pelo diagrama de sequência construído.

Outro benefício ao utilizar a ferramenta é que, ao mesmo tempo que é criado um artefato de modelagem do sistema, pode-se ter o tamanho funcional a partir deste artefato. Assim, tem-se uma ferramenta que documenta o artefato utilizado na modelagem e analisa a partir do mesmo o tamanho em Pontos de Função. Além disso, um ponto importante é que a ferramenta não exige um grande conhecimento para realizar a análise; esse ponto já era esperado e foi descrito pelos participantes ao final do teste.

A análise de PF ao sofrer modificações, como a adição de uma nova tabela ou funcionalidade que pode ou não mudar as funcionalidades já implementadas, necessita da realização de uma nova análise, interpretando as funcionalidades e entidades existentes novamente. Já com a ferramenta, o usuário não se preocupa em analisar a nova tabela (entidade), pois isso é trabalho do interpretador, e somente precisa construir as novas funcionalidades e editar as que alteraram. Assim, a análise é realizada a partir da atualização da documentação do projeto, que está presente na ferramenta, não necessitando realizar a análise

manual e atualizar o artefato em um arquivo do *word*, como é feito por algumas equipes de desenvolvimento.

Entretanto, algumas melhorias podem ser realizadas para automatizar mais a análise, eliminando algumas informações inseridas pelo usuário, interpretando a partir dos diagramas construídos. Um exemplo é a criação do CRUD (*Create, Read, Update e Delete*) automático a partir da tabela, onde o usuário só precisaria alterar os diagramas caso as funcionalidades fugissem do padrão da tabela. Outra melhoria é buscar métodos para automatizar a busca de mensagens de erros e cálculos, podendo interpretar tais informações dos diagramas de sequência construídos. Além disso, os Tipos de Registros (TR) é determinado quando duas ou mais colunas representam uma mesma informação, esse TR é inserido manualmente no sistema, por ser uma particularidade imprevisível da tabela, mas que poderia haver um modo para tentar encontra-los.

Por fim, os resultados obtidos na fase de testes foram satisfatórios, pois comprovaram um ganho de tempo na realização da análise a partir da ferramenta, além de mostrar que o fluxo de uma funcionalidade é mais visível na construção do diagrama de sequência, levando a uma análise mais concisa. Além disso, os participantes também se expuseram ao descrever a facilidade da construção e a não necessidade de conhecimento, no qual influenciou consideravelmente na facilidade de utilizar a ferramenta.

5.1. Trabalhos futuros

A partir dos resultados deste trabalho, outros podem ser desenvolvidos, tais com:

- Tornar o sistema visível em portais destinado as empresas e equipes que utilizam a técnica para métrica de software;
- Buscar parcerias para tornar o sistema comercial;
- Integrar outras técnicas ao sistema, como a Análise de Pontos de Teste;

- Otimizar os processos e métodos de análise no contexto da técnica;
- Melhorar o método de captação de informações para tornar a análise mais fácil;
- Validar o sistema como ferramenta de aprendizagem em disciplinas que envolvam a utilização da técnica de Análise de Pontos de Função como métrica de software;
- Validar o sistema com equipes de desenvolvimentos ou empresas de software;

6. Referências bibliográficas

ALEXANDER, Alvin. **HOW TO DETERMINE YOU APPLICATION SIZE USING FUNCTION POINTS.** *Embarcadero Developer Network*, 2004. Disponível em: <http://uosis.mif.vu.lt/~adamonis/psp/1112p/How%20to%20Determine%20Your%20Application%20Size%20Using%20Function%20Points.pdf>. Acesso em 12 de agosto de 2014.

ANDRADE, Edméia Leonor Pereira de. **PONTOS DE CASOS DE USO E PONTOS DE FUNÇÃO NA GESTÃO DE ESTIMATIVA DE SOFTWARE ORIENTADO A OBJETOS.** Dissertação (Mestrado em Gestão do Conhecimento e Tecnologia da Informação). 2004. Universidade Católica de Brasília, Brasília, 132 p. Disponível em: <http://www.bfpug.com.br/Artigos/UCP/Tese%20Edmeia.zip>. Acesso em 30 de agosto de 2014.

BATISTA, Vitor A.; PEIXOTO, Daniela. C. C.; BORGES, Eduardo P.; PÁDUA, Wilson de; RESENDE, Rodolfo F.; PÁDUA, Clarindo Isaías P. S. **REMOFP: A TOOL FOR COUNTING FUNCTION POINTS FROM UML REQUIREMENT MODELS**. *Advances in Software Engineering*, vol. 2011, p. 1-7, 2011. Disponível em: <http://downloads.hindawi.com/journals/ase/2011/495232.pdf>. Acesso em 13 de outubro de 2014.

BRAGA, Antônio. **ANALISE DE PONTOS DE FUNÇÃO.** Rio de Janeiro: Infobook, 2010. 188p.

CHAOS MANIFESTO. The Standish Group International, 2013. Disponível em: <http://www.versionone.com/assets/img/files/CHAOSManifesto2013.pdf>. Acessado em 23 de agosto de 2014.

COSTA, Elisângela Rocha da. **BANCO DE DADOS RELACIONAIS**. Dissertação (Tecnólogo em Processamento de Dados). 2011. Faculdade de Tecnologia de São Paulo, São

Paulo, 63 p. Disponível em <http://www.fatecsp.br/dti/tcc/tcc0025.pdf>. Acesso em 05 de maio de 2014.

Eclipe Soluctions Gartner. **APPENDIX H – FUNCTION POINT ANALYSIS DATA**. 2005. Disponível em: <http://www.hwcws.cahwnet.gov/projects/docs/go_forward/Appendix_H_Function_Point_Analysis_Data.pdf>. Acesso em 12 de agosto de 2014.

FARIAS, Alexsandro J. de Melo; CUNHA, Diego; TELES, Fabrício de Siqueira; RODRIGUES, Leonardo P. de Holanda; CASTRO, Pedro M. Manhães de. **GESTÃO DA TECNOLOGIA DA INFORMAÇÃO USANDO ANÁLISE POR PONTOS DE FUNÇÃO.** Universidade Federal de Pernambuco - UFPE. Agosto de 2004. Disponível em: <http://www.cin.ufpe.br/~if720/monografias_alunos/monografias2005/GestaoTI.pdf>. Acesso em 29 de agosto de 2014.

FRATERNALI, Piero; TISI, Massimo; BONGIO, Aldo. **AUTOMATING FUNCTION POINT ANALYSIS WITH MODEL DRIVEN DEVELOPMENT.** In: *Proceedings of the 2006 conference of the Center for Advanced Studies on Collaborative research*. IBM Corp., 2006. p. 18. Disponível em: <http://www.webml.org/webml/upload/ent5/1/FP06.pdf>. Acesso em 15 de outubro de 2014.

GUEDES, Gilleanes T. A. **UML 2: GUIA PRÁTICO.** São Paulo: Novatec Editora, 2º ed. 2014.

GUEDES, Gilleanes T. A. **UML 2: UMA ABORDAGEM PRÁTICA.** São Paulo: Novatec Editora, 2º ed. 2011.

HAZAN, Claudia. **ANÁLISE DE PONTOS POR FUNÇÃO: UMA ABORDAGEM GERENCIAL.** Congresso Nacional da SBC, XIX Jornada de Atualização em Informática (JAI), 2000.

I-WEB. **UML**. 2003. Disponível em: <http://www.iweb.com.br/iweb/pdfs/20031008-uml-01.pdf>. Acesso em 02 de abril de 2015.

LEITE, Jair Cavalcanti. **AS 4 + 1 VISÕES**. 2007. Disponível em <http://www.dimap.ufrn.br/~jair/as/slides/Visoes4+1.pdf>. Acesso em 26 de fevereiro de 2015.

LOBATO, Paulo Alexandre. **2DMG — UMA ARQUITETURA DE CLASSES DE MIDDLEWARE PARA A PORTABILIDADE DE APLICATIVOS GRÁFICOS 2D EM DISPOSITIVOS MÓVEIS**. Dissertação (Pós-Graduação em Ciência da Computação). 2011. Universidade Federal de Viçosa, Minas Gerais, 73 p. Disponível em <http://www.dpi.ufv.br/arquivos/ppgcc/dissertacoes/2011-ms-Paulo_Alexandre_Lobato.pdf>. Acesso em 04 de fevereiro de 2015.

LONGSTREET, David. **FUNDAMENTALS OF FUNCTION POINT ANALYSIS.** 2005. Disponível em: <http://www.softwaremetrics.com/files/Fundamentals%20of%20Function%20Point%20Analysis.pdf>. Acesso em 12 de agosto de 2014.

MEDEIROS, Emani Sales de. **DESENVOLVENDO SOFTWARE COM UML 2.0: DEFINITIVO.** São Paulo: Pearson Makron Books, 2004.

MEDEIROS, Luciano Frontino de. **BANCO DE DADOS: PRINCÍPIOS E PRÁTICAS.** Curitiba: InterSaberes, 1º ed. 2013.

MSDN, **Microsoft *Developer Network***, 2013, Online. Disponível em: <http://msdn.microsoft.com/pt-br/library/ie/6974wx4d(v=vs.94).aspx>. Acessado em 15 de abril de 2014.

OLIVEIRA, Júlio. **REQUISITOS, CASOS DE USO E DESENVOLVIMENTO.** 2009. Disponível em:

<http://sistemasecia.freehostia.com/component/jccmultilanguagecontent/article/34-engenhariasoft/73-reqs-casosuso-desenv.html>. Acesso em 02 de abril de 2015.

PMI, ***Project Management Institute***, Online. Disponível em: <https://brasil.pmi.org/brazil/AboutUS/WhatIsProjectManagement.aspx>. Acessado em 15 de abril de 2014.

PINEL, Roque E. Assumpção. **ANÁLISE DE PONTOS DE FUNÇÃO EM SISTEMAS DESENVOLVIDOS USANDO MDA.** Dissertação (Mestrado em Engenharia de Sistemas e Computação). 2012. Universidade Federal do Rio de Janeiro, Rio de Janeiro, 98 p. Disponível em: <http://objdig.ufrj.br/60/teses/coppe_m/RoqueEliasAssumpcaoPinel.pdf>. Acesso em 15 de Outubro de 2014.

PUGA, Sandra; FRANÇA, Edson; GOYA, Milton. **BANCO DE DADOS: IMPLEMENTAÇÃO EM SQL, PL/SQL E ORACLE 11G.** São Paulo: Pearson Education do Brasil, 2013.

REINALDO, Werley Teixeira; FILIPAKIS, Cristina D'Ornellas. **ESTIMATIVA DE TAMANHO DE SOFTWARE UTILIZANDO APF E A ABORDAGEM NESMA.** In: *Encontro de Computação e Informática do Tocantins*, 11., 2009, Palmas. **Anais**... Palmas: Centro Universitário Luterano de Palmas, 2009. Disponível em: <http://www3.ulbra-to.br/eventos/encoinfo/2009/Anais/Estimativa_de_Tamanho_de_Software_Utilizando_APF_e_a_Abordagem_NESMA.pdf >. Acesso em 29 de agosto de 2014.

RIBEIRO, Leandro. **O QUE É UML E DIAGRAMAS DE CASO DE USO: INTRODUÇÃO PRÁTICA À UML.** Devmedia, [s.d.]. Disponível em: <http://www.devmedia.com.br/o-que-e-uml-e-diagramas-de-caso-de-uso-introducao-pratica-a-uml/23408>. Acesso em 02 de abril de 2015.

RODRIGUES, William Almeida. **CRIZON – UMA FERRAMENTA PARA MODELAGEM DE SOFTWARE.** Centro Universitário Luterano de Palmas, Palmas, Tocantins, 2012.

SEABRE, Rodolfo Moacir Júnior. **ANÁLISE E PROJETO ORIENTADO A OBJETOS USANDO UML E O PROCESSO UNIFICADO.** Dissertação (Bacharelado em Ciência da Computação). 2004. Universidade Federal do Pará, 113 p. Disponível em: <http://cultura.ufpa.br/cdesouza/teaching/cedai/APOOUMLP.pdf>. Acesso em 12 de setembro de 2014.

SOBRE ANÁLISE DE PONTOS DE FUNÇÃO. [s.d.]. Disponível em: <http://www.ifpug.org/about-ifpug/about-function-point-analysis/?lang=pt>. Acesso em 1 de dezembro de 2014.

STADZISZ, Paulo Cézar. **PROJETO DE SOFTWARE USANDO A UML.** Centro Federal de Educação Tecnológica do Paraná, 2002. Disponível em: <http://www.etelg.com.br/paginaete/downloads/informatica/apostila2uml.pdf>. Acesso em 12 de setembro de 2014.

TACLA, Cesar Augusto. **ANÁLISE E PROJETO OO & UML 2.0.** Paraná: Universidade Tecnológica Federal do Paraná, 2007. Disponível em: <http://www.dainf.ct.utfpr.edu.br/~tacla/UML/Apostila.pdf>. Acesso em 6 de outubro de 2014. [Apostila do Curso de Tecnologia de Orientação a Objetos e JAVA].

TEPFENHART, William M. **UML E C++: GUIA PRÁTICO DE DESENVOLVIMENTO ORIENTADO A OBJETOS.** Tradução: Celso Roberto Paschoa, Revisão Técnica: José David Furlan. São Paulo: MAKRON BOOK, 2001.

UEMURA, Takuya; Kusumoto, Shinji; Inoue, Katsuro. **FUNCTION-POINT ANALYSIS USING DESIGN SPECIFICATIONS BASED ON THE UNIFIED MODELLING LANGUAGE**. *Jornal of Software Maintenance and Evolution: research and Practice*, vol. 13,

n. 4, p. 223-243, 31 ago. 2001. Disponível em: <http://www.researchgate.net/publication/220674140_Function-point_analysis_using_design_specifications_based_on_the_Unified_Modelling_Language/links/09e4150aa32b36a9b5000000>. Acesso em 14 de outubro de 2014.

VARGAS, Thânia C. de Souza. **A HISTÓRIA DE UML E SEUS DIAGRAMAS.** Santa Catarina: Universidade Federal de Santa Catarina, [s.d.]. Disponível em: <https://projetos.inf.ufsc.br/arquivos_projetos/projeto_721/artigo.tcc.pdf>. Acesso em 12 de setembro de 2014.

VAZQYEZ, Carlos Eduardo; SIMÕES, Guilherme Siqueira; ALBERT, Renato Machado. **ANÁLISE DE PONTOS DE FUNÇÃO: MEDIÇÃO, ESTIMATIVAS E GERENCIAMENTO DE PROJETOS DE SOFTWARE.** São Paulo: Érica, 10º edição revisada, 2010.

7. Apêndices

I. Ficha de teste manual

CRIZON

Ficha de teste manual

Nome do Participante:			
Data e Hora do Início:		Hora do Fim do Teste:	
Local da Realização:			
Tipo de Participante:	☐ Não Conheço a Técnica	☐ Conheço a Técnica	

Análise realizada Manualmente	
Total de PF:	

Determinação da Complexidade de Dados					
Nome	Tipo	TD	TR	Complexidade	PF
Total					
Hora Início:		Hora Fim:			

Determinação da Complexidade de Transação					
Nome	Tipo	TD	AR	Complexidade	PF
Total					
Hora Início:		Hora Fim:			

_____

Participante

II. Ficha de teste automatizado

Ficha de Teste CRIZON 2.0

Nome do Participante:			
Data e Hora do Início:		Hora do Fim do Teste:	
Local da Realização:			
Tipo de Participante:	☐ Não Conheço a Técnica	☐ Conheço a Técnica	

Análise realizada na ferramenta	
Total de PF:	

Determinação da Complexidade de Dados						
Nome		Tipo	TD	TR	Complexidade	PF
Total						
Hora Início:		Hora Fim:				

Determinação da Complexidade de Transação						
Nome		Tipo	TD	AR	Complexidade	PF
Total						
Hora Início:		Hora Fim:				

_____

Participante