UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

FERRAMENTA PARA AUXÍLIO NO ENSINO DE ORIENTAÇÃO A OBJETOS

CLAOR ÉDINO BAUER

BLUMENAU
2015

2015-1/05

CLAOR ÉDINO BAUER

FERRAMENTA PARA AUXÍLIO NO ENSINO DE ORIENTAÇÃO A OBJETOS

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Ciência da Computação do Centro de Ciências Exatas e Naturais da Universidade Regional de Blumenau como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Prof. Marcel Hugo, Mestre

BLUMENAU
2015

2015-1/05

FERRAMENTA PARA AUXÍLIO NO ENSINO DE ORIENTAÇÃO A OBJETOS

Por

CLAOR ÉDINO BAUER

Trabalho de Conclusão de Curso aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II pela banca examinadora formada por:

Presidente:	Prof. Marcel Hugo, Mestre – Orientador, FURB
Membro:	Prof. Everaldo Artur Grahl, Mestre – FURB
Membro:	Prof. Matheus Carvalho Viana, Doutor – FURB

Blumenau, 30 de junho de 2015

Dedico este trabalho a meu pai pela paciência que teve comigo e pela a ajuda fornecida, e aos meus amigos pelo incentivo e ânimo que me deram durante o desenvolvimento deste projeto.

AGRADECIMENTOS

A meu pai, pelo apoio e paciência que ele tem tido comigo durante toda a minha vida, em especial nos últimos anos.

Aos meus amigos por sempre me incentivarem e me animarem mesmo quando eu estava para baixo.

Ao meu orientador pela ajuda e colaboração que ele deu para o desenvolvimento e conclusão deste projeto.

O conceito sobre um herói, é que mesmo quando não parece haver uma luz no fim do túnel, ele vai continuar cavando, ele vai continuar tentando fazer o certo e compensar o que se passou, só porque esse é quem ele é.

Joss Whedon

RESUMO

O paradigma da Orientação a Objetos na maioria dos casos faz parte do currículo de ensino de programação. No entanto as metodologias e até mesmo as ferramentas utilizadas para o ensino desse paradigma, muitas vezes não ajudam o suficiente e o aprendizado ainda é bastante afetado. Este trabalho tem como objetivo disponibilizar uma ferramenta para dispositivos móveis que possa auxiliar professores e alunos no ensino do paradigma de Orientação a Objetos. A ferramenta, batizada de *OO Programming Lab*, foi desenvolvida para o sistema operacional Windows Phone e possui uma interface gráfica que se comunica com serviços disponibilizados com tecnologia WCF para compilar as classes, criar objetos e armazenar de dados. Para o desenvolvimento foi utilizada a técnica de *Test Driven Development* (TDD), que se mostrou útil para manter o código desenvolvido coeso e correto, apesar das alterações ao longo do processo. Os alunos podem interagir com a ferramenta criando novas classes e métodos e atributos para essas classes. Com os objetos criados, é possível fazer a chamada de seus métodos e também inspecionar o valor de seus atributos. Todos os dados criados durante a execução da ferramenta ficam salvos no servidor e são carregados quando a ferramenta é inicializada. Ao final do trabalho foi realizada uma validação com um grupo de alunos, que avaliou de forma positiva esta ferramenta.

Palavras-chave: Ensino de orientação a objetos. Dispositivos móveis. WCF.

ABSTRACT

The paradigm of Object Orientation in most cases is part of the programming education curriculum. However the methodologies and even the tools used for teaching this paradigm often do not help enough and learning is still quite affected. This document aims to present a tool for mobile devices that can assist teachers and students in learning Object-Oriented paradigm. The tool, called OO Programming Lab, has been developed for Windows Phone operating system. It has a graphical interface that communicates with services available through WCF technology to compile the classes, create objects and store data. For development it has been used the technique Test Driven Development (TDD), which proved to be useful for keeping code developed cohesive and correct despite changes throughout the process. Students can interact with the tool by creating new classes and methods and attributes for these classes. With the created objects, it is possible to call their methods and also inspect the value of its attributes. All data created during the execution of the tool is saved on the server and loaded when the tool is started. At the end of the work, a validation was performed with a group of students who evaluated this tool positively.

Key-words: Teaching object orientation. Mobile devices. WCF.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

API – *Application Program Interface*

IIS – *Internet Information Services*

OO – Orientação a Objetos

TDD – *Test Driven Development*

WCF – *Windows Communication Foundation*

XAML – *Extensible Application Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Segundo Kolling (2011, p. 1), a Orientação a Objetos (OO) tem, nos últimos anos, tornado-se o paradigma de programação mais utilizado. Ela é amplamente utilizada na educação e na indústria e quase toda universidade ensina Orientação a Objetos em algum ponto do seu currículo. A comunidade de software em geral concorda que ensinar OO é benéfico, pois ela suporta os conceitos que têm sido ensinados por muitos anos, tais como programação estruturada, modularização e *design* de programas.

> A OO também suporta técnicas para resolver problemas que apenas mais recentemente entraram no currículo: programação em times, manutenção de grandes sistemas e reutilização de programas. Em resumo, Orientação a Objetos aparenta ser uma boa ferramenta para ensinar as metodologias de programação que se consideram importantes (KOLLING, 2011, p. 1).

Ao observar a forma como a análise e o projeto de sistemas vêm sendo ensinados e praticados em muitos lugares, pode-se verificar que muitos profissionais simplesmente adotam uma linguagem orientada a objetos, ou algum fragmento de processo orientado a objetos, sem ter realmente muita noção do que estão fazendo (WAZLAWICK, 2004, p. 18). Ainda segundo Wazlawick (2004, p. 18), de nada adianta realizar pesados investimentos em ferramentas CASE orientadas a objetos sem que se compreenda com clareza a forma de pensar desse modelo de programação.

Desde 1989, Beck e Cunningham (1989, p. 1) já citavam as dificuldades no ensino da Orientação a Objetos, afirmando que “o problema mais difícil em ensinar programação orientada a objetos é fazer com que o estudante desista da ideia de controle global que é possível com programação procedural, e agregar a ideia de objetos cumprindo suas tarefas”. Programadores novatos se confundem com regressões a pensamentos globais: acesso a variáveis globais, ponteiros desnecessários e dependência inadequada sobre a execução de outros objetos. Quinze anos depois, Galhardo (2004, p. 2), afirmou que, normalmente, o percurso a ser percorrido pelo aluno durante o ensino da programação, inicia-se com a aprendizagem da programação imperativa, pois a mesma possibilita que a lógica de programação possa ser explorada com maior facilidade. Quando o aluno começa a ter contato com uma linguagem orientada a objetos, ele sente dificuldades em transpor as barreiras que diferenciam os estilos de programação imperativa e orientada a objetos.

Tendo-se em mente essa dificuldade no ensino de Orientação a Objetos, foi desenvolvido neste trabalho uma ferramenta para auxiliar professores e alunos no ensino do paradigma de Orientação a Objetos. A ferramenta permite aos usuários acompanhar a execução de código orientado a objeto em um ambiente interativo, onde vários alunos podem

participar de um grupo (sala de estudos), desenvolvendo classes, criando os respectivos objetos e fazendo-os interagir.

A ferramenta foi desenvolvida para *smartphones* por estes serem mais acessíveis em comparação com um computador *desktop* ou *notebook*. O próprio usuário ou aluno já possui o dispositivo ao invés de a instituição ter de fornecer o equipamento. Isso, por sua vez, também possibilita a utilização da ferramenta em sala de aula, sem a necessidade de levar os alunos para um laboratório, além de permitir acesso ao ambiente (sala de estudo) em qualquer lugar, desde que possua conexão com o servidor, esteja ele conectado em uma rede local ou hospedado na internet.

O Windows Phone foi escolhido como a plataforma alvo de desenvolvimento desta ferramenta. Conforme pode ser visto nas estatísticas dos últimos anos em Statista (2015), o Windows Phone vem assumindo uma parcela cada vez maior do mercado de smartphones. Além disso, este sistema operacional ainda foi pouco explorado em trabalhos acadêmicos. Por este motivo, foi escolhido desenvolver esta ferramenta para Windows Phone utilizando a linguagem C#.

Durante o desenvolvimento da ferramenta foram utilizados os conceitos de *Test Driven Development* (TDD), que é uma técnica de desenvolvimento em que apenas se escreve código funcional após ter escrito um teste automatizado. Isso faz com que o código escrito tenha alta coesão e um baixo nível de acoplamento entre os componentes do programa, pois somente assim ele poderá ser facilmente testado (BECK, 2002, p. 8).

1.1 OBJETIVOS

O objetivo deste trabalho é disponibilizar uma ferramenta para dispositivos móveis que possa auxiliar professores e alunos no ensino e aprendizado do paradigma de Orientação a Objetos.

Os objetivos específicos do trabalho são:

a) auxiliar os usuários no aprendizado dos conceitos de Orientação a Objetos pela observação da execução de código OO;
b) auxiliar professores no ensino de Orientação a Objetos;
c) permitir que diferentes usuários possam interagir entre si, criando e utilizando código fonte de classes em conjunto;
d) fazer testes com um grupo de usuários para validar a ferramenta;
e) Utilizar os conceitos de TDD ao long do desenvolvimento da ferramenta.

1.2 ESTRUTURA

No Capítulo 2 é apresentada a revisão bibliográfica abordando os principais assuntos e tecnologias referentes ao trabalho, bem como trabalhos correlatos. No Capítulo 3 é relatada a fase de desenvolvimento do aplicativo, as principais ferramentas e tecnologias usadas e as dificuldades encontradas. No Capítulo 4 são relatados os resultados obtidos e a conclusão a que se chegou, assim como sugestões de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os conceitos envolvidos no desenvolvimento do trabalho. Na Seção 2.1 são apresentados alguns conceitos básicos da Orientação a Objetos. Na Seção 2.2 são apresentados alguns relatos e metodologias atuais do ensino de Orientação de Objetos. Na Seção 2.3 é comentado a respeito de reflexão computacional em C#. Na Seção 2.4 é comentado sobre o processo *Test Driven Development*. Na Seção 2.5 é dada uma introdução ao WCF. Na Seção 2.6 é feito um comentário sobre XAML. Por fim, na Seção 2.7 são apresentadas algumas ferramentas e trabalhos correlatos na área.

2.1 ORIENTAÇÃO A OBJETOS

A Orientação a Objetos é uma abordagem para o desenvolvimento de software, na qual a estrutura do software é baseada em objetos que interagem entre si para realizar uma tarefa. Essa interação assume a forma de mensagens sendo transmitidas entre os objetos (CLARK, 2003, p. 3).

Os conceitos da Orientação a Objetos iniciaram em meados dos anos 1960 com uma linguagem de programação chamada Simula e evoluíram nos anos 1970 com o advento do Smalltalk (CLARK, 2003, p. 4). Embora os desenvolvedores de software não tivessem adotado completamente esses primeiros avanços, as metodologias orientadas a objetos continuaram a se desenvolver. Na década de 1980 houve um ressurgimento do interesse nas metodologias orientadas a objetos devido à evolução do Smalltalk e crescimento do uso de dialetos da linguagem C orientados a objetos. Especificamente, as linguagens C++ e Eifel tornaram-se populares entre os programadores de computação. A Orientação a Objetos continuou a crescer em popularidade nos anos 1990, mais notavelmente com o advento do Java e o número de adeptos que ele atraiu. Em 2002, com o Visual Studio .NET 2002, a Microsoft introduziu uma nova linguagem orientada a objetos, C# (pronunciada como C-sharp) e reconstruiu o Visual Basic para que ele fosse, de fato, uma linguagem orientada a objetos (CLARK, 2003, p. 4).

O paradigma da Orientação a Objetos surgiu para resolver uma série de problemas que os programadores estavam encontrando com paradigmas de programação já existentes. Por exemplo, para fazer manutenção, raramente é prático iniciar a implementação do zero ou reimplementar o programa inteiro. Ao invés disso, deve-se ajustar as modificações dentro da estrutura existente e, para isso, é necessário que a arquitetura do programa acomode mudanças. Em particular, os módulos do programa devem ser independentes ao ponto que

uma mudança possa ser feita numa parte do programa sem que seja necessário mudar todas as outras partes (GUTTAG; LISKOV, 2006, p. 2).

Segundo McLaughlin, Pollice e West (2007, p. 125), um software bem projetado é fácil de alterar e de estender. Utilizando os princípios da Orientação a Objetos, como encapsulamento e herança, pode-se tornar o software mais flexível. Esse é um ponto importante da Orientação a Objetos comparado a outros paradigmas de programação.

De acordo com Coplien (2003, p. 4), as mais populares linguagens de programação usam classes como sua estrutura básica. Classes são a estrutura básica da Orientação a Objetos no C++ e elas são indispensáveis no Java e no Smalltalk. Uma classe define o modelo de um objeto, ou seja, todas as características que o objeto contém foram definidas pela classe. É importante considerar que uma classe não representa nenhum objeto em particular, pois é só um modelo (MENDES, 2009, p. 29).

Outra característica importante da Orientação a Objetos é a coesão do código. A coesão mede o grau de conectividade entre os elementos de um único módulo, classe ou objeto. Quanto maior a coesão do seu software, mais bem definidas e relacionadas serão as responsabilidades de cada classe individual em sua aplicação. Cada classe possui um conjunto muito específico de ações intimamente relacionadas executadas por ela (MCLAUGHLIN; POLLICE; WEST, 2007, p. 125).

O paradigma de Orientação a Objetos traz um enfoque diferente da programação estruturada, no sentido de adotar formas mais próximas do mecanismo humano para gerenciar a complexidade de um sistema. Nesse paradigma, o mundo real é visto como sendo constituído de objetos autônomos, concorrentes, que interagem entre si. Cada objeto tem seu próprio estado (atributos) e comportamento (métodos), semelhante a seu correspondente no mundo real (MENDES, 2009, p. 18).

Outro recurso importante do paradigma orientado a objetos é a encapsulação. A encapsulação ou encapsulamento é o processo no qual nenhum acesso direto é concedido aos dados; ao contrário, é ocultado. Para ter acesso aos dados, é necessário interagir com o objeto responsável pelos dados. Encapsulando os dados, os dados de seu sistema tornam-se mais seguros e confiáveis. Toma-se conhecimento de como os dados estão sendo acessados e quais operações estão sendo executadas nos dados. Isso torna a manutenção do programa mais fácil e também simplifica o processo de depuração (CLARK, 2003, p. 7).

2.2 ENSINO DE ORIENTAÇÃO A OBJETOS

O aprendizado de Orientação a Objetos tem se tornado um desafio para professores e alunos. Por se tratar de um raciocínio abstrato, torna-se difícil a compreensão e a justificativa clara da sua utilização para os alunos. Porém, com o aumento da complexidade e distribuição dos sistemas, torna-se premente o uso de técnicas, em particular a Orientação a Objetos, que permitam o uso de características da engenharia como análise, projeto, e acompanhamento em desenvolvimento de software (CARDOSO, 2002, p.6).

As ferramentas e técnicas desenvolvidas para o ensino de Programação Orientada a Objetos não são tão maduras quanto as ferramentas de ensino de programação estruturada. O resultado disso é uma maior dificuldade em ensinar Orientação a Objetos para alunos novatos em programação (CAMINHA, 2012b, p.12).

A maioria dos alunos que começam a estudar a linguagem Java tem dificuldade para entender a diferença entre uma classe e um objeto, bem como, qual o real significado de um atributo e de um método (MENDES, 2009, p. 27). Dentre as dificuldades, as que mais se destacam são a falta de motivação dos alunos com o tema de programação, a dificuldade com a sintaxe e recursos específicos de cada linguagem e a ausência de metodologia para transformação de um problema em um conjunto de classes que representariam sua respectiva solução (CAMINHA, 2012a).

A importância do ensino da programação está não só no conteúdo abordado, mas também na capacidade de transpor os conceitos estudados para uma linguagem de programação de forma prática. O desenvolvimento de ferramentas que atuem como apoio ao ensino da programação tem sido uma alternativa amplamente adotada dentro do ensino de linguagens de programação. Tais ferramentas têm por objetivo enriquecer o processo de ensino-aprendizagem e motivar os alunos apresentando-lhes diferentes formas de visualizar os conceitos estudados (GALHARDO, 2004, p. 1).

O uso da tecnologia no ensino abre espaço para um novo cenário que enriquece o processo de ensino-aprendizagem. Nesse novo cenário, professor e aluno devem interagir, participando dinamicamente do processo, construindo juntos o caminho da aprendizagem. O aluno deixa de ser um objeto estático e o professor passa a abordar os conceitos de uma forma mais dinâmica (GALHARDO; ZAINA, 2004, p. 6).

Conforme Naps et al. (2002, p. 1), um algoritmo pode ser observado de várias maneiras diferentes, por exemplo, controle de fluxo no código fonte ou o estado das estruturas de dados. Dando esses recursos de observação a um aluno pode facilitar o seu entendimento

do algoritmo. Dessa maneira, o aluno pode relacionar as ações do algoritmo com o código do programa.

2.3 REFLEXÃO COMPUTACIONAL

A reflexão computacional é a capacidade de um programa examinar a si mesmo e seu ambiente e mudar seu comportamento conforme necessário. Para executar esse auto exame, o programa necessita ter uma representação de suas informações, chamados metadados. No contexto de Orientação a Objetos, os metadados são organizados em objetos, chamados meta-objetos. O auto exame durante a execução é chamado introspecção (FORMAN; FORMAN, 2005, p. 3).

Especificamente em C#, que foi a limgagem utilizada no desenvolvimento deste trabalho, a reflexão provê objetos (do tipo `Type`) que descrevem *assemblies*, módulos e tipos. É possível utilizar a reflexão para criar dinamicamente uma instância de um tipo, vincular o tipo a um objeto existente, ou obter o tipo de um objeto existente e invocar seus métodos ou acessar seus campos e propriedades. Também é possível, através da reflexão, acessar os atributos de um objeto existente (MSDN, 2013a).

Ainda segundo o MSDN (2013b), *assemblies* contêm módulos, módulos contêm tipos, e tipos contêm membros. A reflexão provê objetos que encapsulam *assemblies*, módulos e tipos. Usos típicos da reflexão incluem os seguintes:

a) usar a classe `Assembly` para definir e carregar *assemblies*, carregar módulos que estão listados no manifesto do *assembly* e obter um tipo desse *assembly* e criar instâncias destes tipos;
b) usar a classe `Module` para descobrir informações como o *assembly* que contém o módulo desejado e as classes que o módulo contêm;
c) usar a classe `ConstructorInfo` para descobrir informações como nome, parâmetros, modificadores de acesso e detalhes de implementação (como abstrato ou virtual) de um construtor;
d) usar a classe `MethodInfo` para descobrir informações como o nome, tipo de retorno, parâmetros, modificadores de acesso e detalhes de implementação de um método;
e) usar a classe `FieldInfo` para descobrir informações como o nome, modificadores de acesso e detalhes de implementação (como campos estáticos) de um campo e obter ou setar o valor desse campo.

De acordo com C-Sharp Tutorials (2015), a classe `Type` é o fundamento da reflexão. Ela serve para obter informações em tempo de execução a respeito de um *assembly*, um módulo ou um tipo. Felizmente, obter a referência à classe `Type` de um objeto é muito simples, já que toda classe herda da classe `Object` e, portanto, possui um método `GetType()`. Se for necessário informação sobre um tipo não instanciado, pode-se usar o método `typeof()` disponível globalmente, que irá retornar as informações do tipo solicitado.

Ao obter detalhes de métodos de um tipo, a informação é retornada usando objetos `MethodInfo`. Assim como quando se faz reflexão de outros tipos de membros, pode-se recuperar detalhes de um único método com base em seu nome, opcionalmente fornecendo *binding flags*. É possível fazer isso usando o método `GetMethod()` da classe `Type`. Para recuperar os dados de vários métodos, ao mesmo tempo, utiliza-se o método `GetMethods()`. Quando usado sem argumentos, todos os métodos públicos da classe são retornados em uma matriz de `MethodInfo`. Para fazer reflexão sobre métodos não-públicos pode-se adicionar *binding flags* (CARR, 2012b, p.1).

Para fazer a chamada de um método, primeiramente, deve-se criar o objeto que contém o método para executar. O objeto é criado usando o método `CreateInstance()` da classe `Activator`. Em seguida, obtém-se um objeto MethodInfo para o método em questão utilizando o método `GetMethod()` da classe `Type`. Finalmente, executa-se o método chamando o membro `Invoke()` do `MethodInfo` (CARR, 2012a, p.1). No Quadro 1 é demonstrado um código fonte exemplificando isto.

Quadro 1 – Invocação de método via reflexão

```
Type testType = typeof(Test);
object testInstance = Activator.CreateInstance(testType);

MethodInfo toInvoke = testType.GetMethod("ShowMessage");
toInvoke.Invoke(testInstance, null);
```

Fonte: Carr (2012a, p.1).

2.4 TEST DRIVEN DEVELOPMENT (TDD)

De acordo com Freeman e Pryce (2010, p. 1), o TDD segue uma ideia simples: escrever o teste antes de escrever o código funcional. Os testes nesse caso, deixam de ser apenas para corrigir as falhas no programa antes de ele chegar ao usuário e passam a existir também para ajudar a equipe a entender as funcionalidades que o usuário precisa e entregar essas funcionalidades executando de forma previsível e confiável. Quando seguido corretamente, o TDD muda a maneira como se desenvolve softwares e aumenta a qualidade dos sistemas que são construídos, em particular a segurança e flexibilidade de novas funcionalidades.

Beck (2002, p. 30) afirma que o TDD gira em torno de um ciclo formado por três etapas. Na primeira etapa é escrito um teste para assegurar de que a funcionalidade que será desenvolvida funcionará. Em seguida é escrito o código da funcionalidade para fazer esse teste passar. Na terceira etapa é feita a refatoração na qual são removidos possíveis códigos duplicados. Ao terminar essa refatoração, o teste deve continuar sendo executado com sucesso. Escrever o teste antes do código também permite definir detalhes da implementação desse código antecipadamente, como por exemplo, entrada e saída de métodos.

O esforço de escrever os testes primeiro dá ao desenvolvedor um rápido *feedback* sobre a qualidade do *design* da funcionalidade que está sendo implementada. Além disso, tornar o código acessível para testes faz com que este código fique mais bem escrito e modular (FREEMAN; PRYCE, 2010, p. 1).

2.5 WINDOWS COMMUNICATION FOUNDATION

Windows Communication Foundation (WCF) é uma estrutura para a construção de aplicações orientadas a serviços específica do .NET Framework. O WCF é a implementação da Microsoft de um conjunto de padrões da indústria que definem interações de serviço, tipo de conversões, triagem, e a gestão de vários protocolos. Consequentemente, o WCF fornece interoperabilidade entre os serviços Usando WCF, pode-se enviar dados como mensagens assíncronas a partir de um terminal de serviço para outro. Um terminal de serviço pode ser parte de um serviço continuamente disponível hospedado pelo *Internet Information Services* - IIS, ou pode ser um serviço hospedado em um aplicativo. Um ponto de extremidade pode ser um cliente de um serviço que solicita dados a partir de um terminal de serviço. As mensagens podem ser tão simples como um único caractere ou uma palavra enviados como XML, ou tão complexo como um fluxo de dados binários (MSDN, 2015a).

WCF fornece um ambiente em tempo de execução para seus serviços, permitindo-lhe expor *Common Language Runtime* (CLR) como tipos de serviços e de consumir outros serviços como tipos CLR. O CLR é um ambiente de tempo de execução disponibilizado pelo .NET Framework, que executa o código e provê serviços que tornam o processo de desenvolvimento mais fácil. Compiladores e ferramentas expõem as funcionalidades do CLR e habilitam a escrita de código que se beneficia desse ambiente de execução gerenciado. Embora, em teoria, seja possível construir serviços sem WCF, na prática, a construção de serviços é significativamente mais fácil com o WCF (LÖWY, 2010, p. 1).

De acordo com Pathak (2011, p. 13), o WCF auxilia o desenvolvimento de sistemas distribuídos, tomando todas as capacidades da tecnologia dos sistemas distribuídos existentes

e permitindo que seja utilizada uma API limpa e simples. Em outras palavras, WCF traz toda a gama de recursos para sistemas distribuídos em uma única API.

Segundo Liu (2012, p.13), o WCF é uma tecnologia guarda-chuva que cobre os serviços *Web* ASMX, *.NET Remoting*, WSE, *Enterprise Service*, e *System Messaging*. Ele é projetado para oferecer uma abordagem gerenciável para computação distribuída, ampla interoperabilidade e suporte direto para a orientação a serviços. WCF suporta muitos estilos de desenvolvimento de aplicativos distribuídos, fornecendo uma arquitetura em camadas. Na sua base, a arquitetura em canais do WCF permite passar mensagens não tipadas de forma assíncrona. Construído no topo desta base estão as funcionalidades de protocolo de intercâmbio de dados seguro, confiável e transacionado, e uma ampla escolha de opções de transporte e de codificação.

2.6 EXTENSIBLE APPLICATION MARKUP LANGUAGE

Extensible Application Markup Language, ou XAML (pronuncia-se "zammel"), é uma linguagem de marcação baseada em XML desenvolvida pela Microsoft. É a linguagem responsável pela apresentação visual de aplicativos desenvolvidos na IDE *Microsoft Expression Blend* ou aplicativos para Windows Phone desenvolvidos na IDE Visual Studio. Criar um aplicativo no *Expression Blend* significa escrever código XAML, manualmente ou visualmente trabalhando na visualização *Design* do *Expression Blend* (MSDN, 2015b).

A abordagem XAML oferece suporte a um estilo de programação mais interativo, no qual é possível declarar controles de tela usando XAML e adicionar código C# para apoiá-los. Com XAML também é possível definir estilos para reduzir a duplicação na marcação (FREEMAN, 2012, p. 20). Com esses estilos é possível criar um conjunto de propriedades pra os controles e usar esses estilos para personalizar a aparência de vários controles de uma única vez. O painel `Grid` é o controle de leiaute básico em páginas de aplicativos do Windows baseadas em XAML. O `Grid` fornece um excelente equilíbrio entre esforço e controle, permitindo um gerenciamento sobre como os controles de tela estão dispostos em relação uns aos outros sem a necessidade de especificar as coordenadas exatas para cada elemento (BURNS, 2012, p. 47).

Ainda segundo Burns (2012, p. 46), o `Grid` é um membro de uma classe de controles conhecidos como controles de leiaute ou painéis, cuja finalidade é proporcionar um recipiente responsável pelo leiaute e disposição dos controles que eles contêm. O `Grid` lida com as responsabilidades de leiaute através do estabelecimento de uma ou mais linhas e colunas em que outros controles são colocados. Cada controle contido, por padrão, será na linha 0 e

coluna 0, com o desenvolvedor ou designer necessitando especificar um valor não padrão para conseguir adicionar o controle no local desejado.

Ao contrário do `Grid`, que requer que seja declarado o número de linhas e colunas inicialmente, o `StackPanel` é usado para expor seus controles contidos de forma linear ao longo ou um plano vertical ou horizontal. Outro controle comum é o `TextBox`, que é usado para exibir texto com o qual o usuário pode interagir, realizando ações como editar o texto ou copiá-lo para a área de transferência. É usado sempre que uma resposta de forma livre é necessária do usuário, como responder à pergunta "Qual é seu nome?" (BURNS, 2012, p. 52).

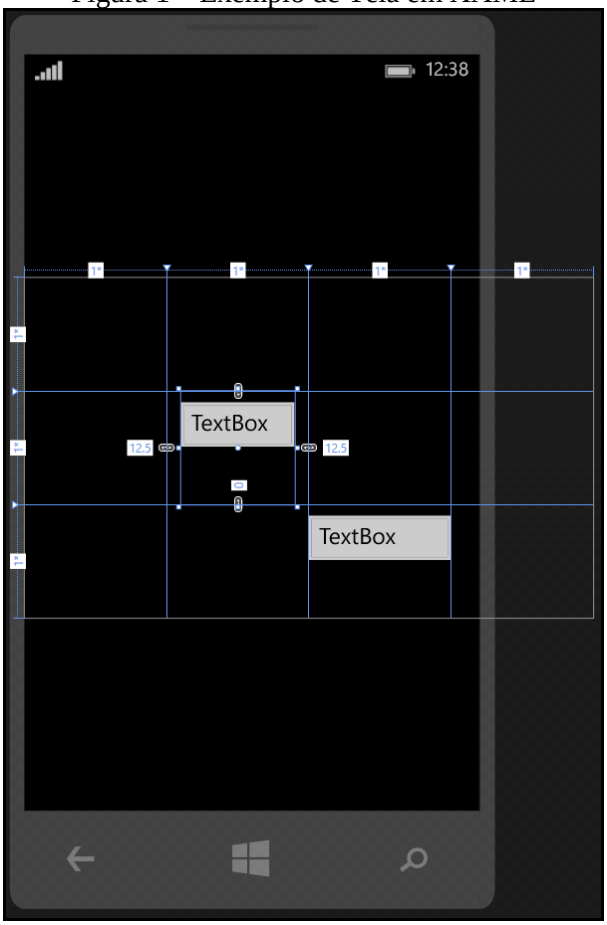
No Quadro 2 é apresentado um exemplo de como ficaria um código XAML com um `Grid` de quatro colunas por três linhas, possuindo dois `StackPanel` e dois `TextBox`, sendo que um dos `TextBox` estaria dentro de um `StackPanel`. A Figura 1 apresenta a tela criada a partir desse código.

Quadro 2 – Código exemplo de XAML

```
  <Grid x:Name="LayoutRoot" Width="500" Height="300">
      <Grid.ColumnDefinitions>
          <ColumnDefinition />
          <ColumnDefinition />
          <ColumnDefinition />
          <ColumnDefinition />
      </Grid.ColumnDefinitions>
      <Grid.RowDefinitions>
          <RowDefinition />
          <RowDefinition />
          <RowDefinition />
      </Grid.RowDefinitions>

      <StackPanel Height="100" Width="100" Grid.Column="0" Grid.Row="0"/>
      <StackPanel Height="100" Width="100" Grid.Column="1" Grid.Row="1">
          <TextBox Margin="0,10,0,0" Text="TextBox"/>
      </StackPanel>
 <TextBox Margin="0,10,0,0" Grid.Column="2" Grid.Row="2" Text="TextBox" />
</Grid>
```

Figura 1 – Exemplo de Tela em XAML



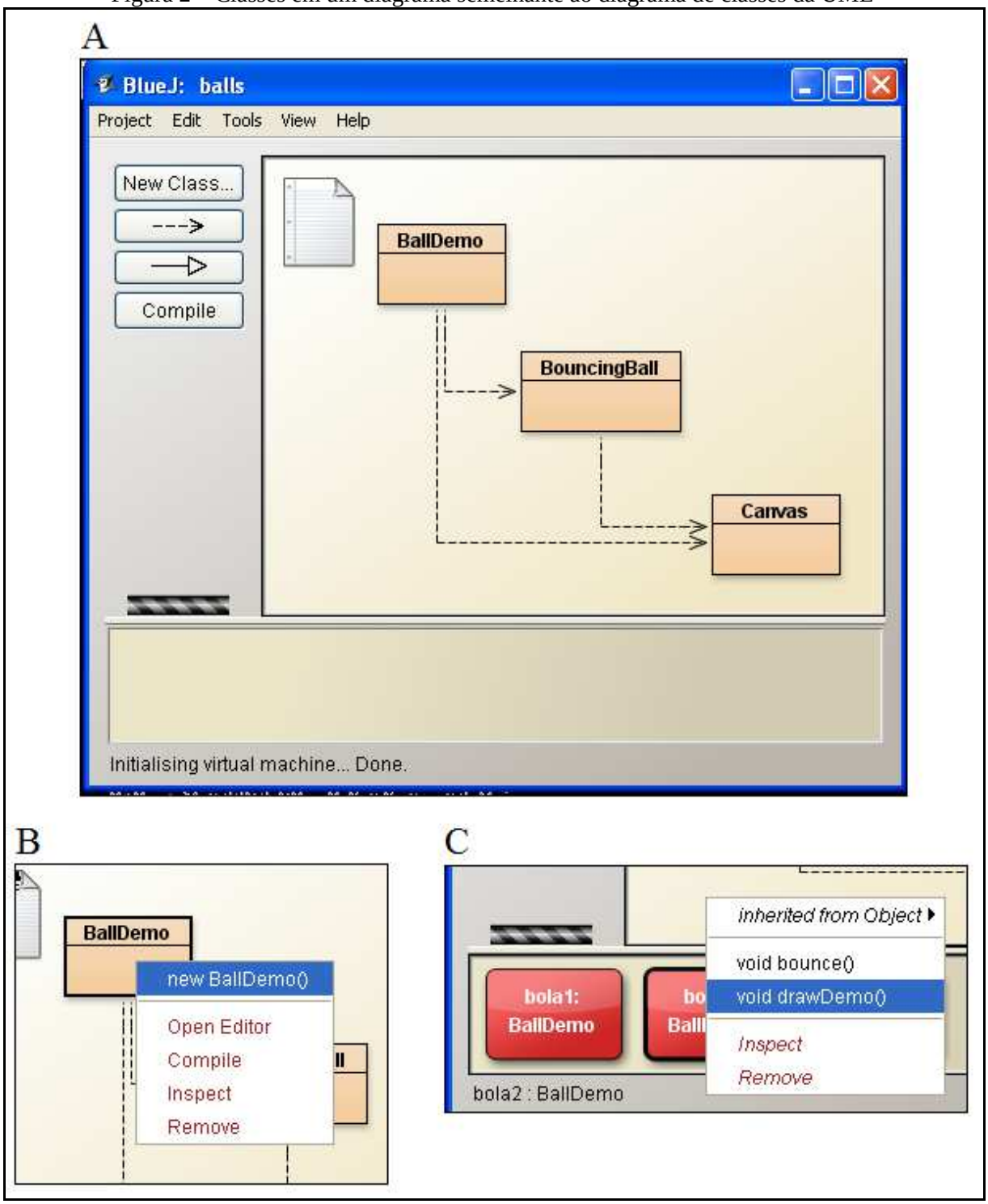
2.7 TRABALHOS CORRELATOS

A seguir são apresentados alguns trabalhos e ferramentas que tem relação com o tema e a ferramenta proposta neste trabalho.

2.7.1 BlueJ

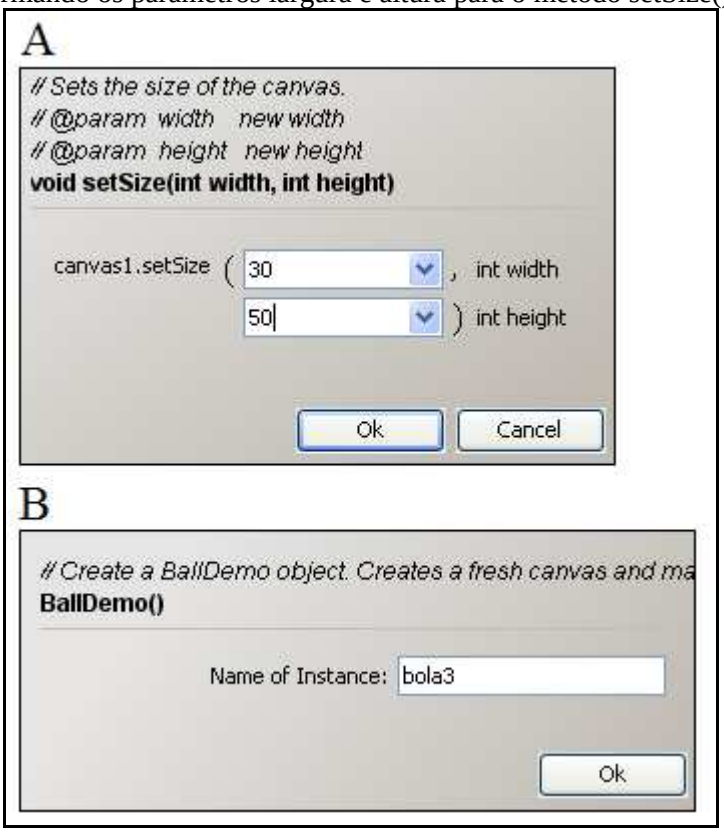
Conforme Caminha (2012a), o BlueJ é focado em interatividade, visualização e simplicidade. É possível visualizar todas as classes criadas na forma de um diagrama semelhante ao diagrama de classes da UML (Figura 2.A) e criar objetos a partir destas classes (Figura 2.B) e visualizá-los em um “painel de objetos”. Com os objetos dispostos em um painel, é possível interagir com eles visualizando seu estado e realizando chamadas de métodos diretamente (Figura 2.C), apenas com cliques do mouse.

Figura 2 – Classes em um diagrama semelhante ao diagrama de classes da UML



O programa também não precisa de um método `main`, pois qualquer método de qualquer classe pode ser executado em qualquer instante. O BlueJ fornece interface gráfica para digitação dos parâmetros de métodos (Figura 3.A) e a informação dos nomes das variáveis que armazenarão um objeto, como aparece na Figura 3.B.

Figura 3 – Informando os parâmetros largura e altura para o método setSize() de um canvas



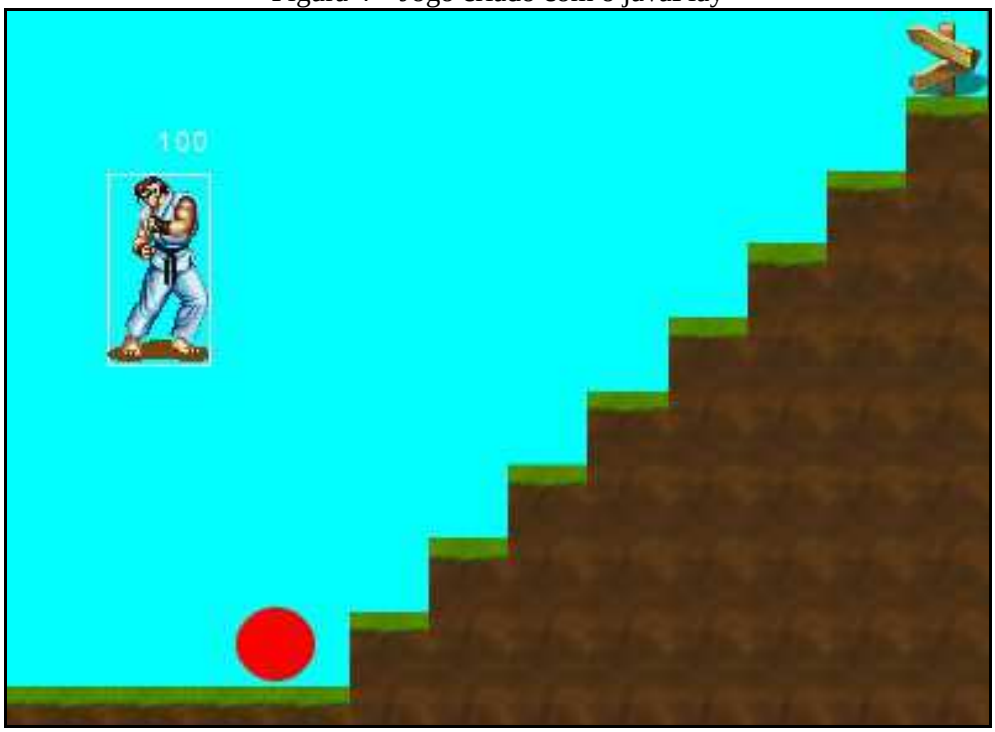
Por ser voltado para simplicidade, o BlueJ não possui muitas das ferramentas comuns em ambientes profissionais. Seu uso é especialmente voltado para o ensino em cursos introdutórios, de modo que o estudante possa colocar seu esforço no aprendizado dos principais conceitos de Programação Orientada a Objetos. O BlueJ permite que os alunos abram um projeto e a partir de suas classes, criem objetos e chamem os métodos desses objetos, sem a necessidade de trabalhar com o código fonte do programa. Ao fazer a chamada dos métodos, é aberta uma janela *pop-up* solicitando os parâmetros a serem passados e, após a chamada do método, é aberta outra janela *pop-up* com o retorno. O BlueJ foi desenvolvido para ambiente *desktop* e atualmente suporta trabalhar com qualquer exemplo de programação Java e pode ser utilizado em qualquer computador que possua o *Java Development Kit* instalado.

2.7.2 Ensino de Programação Orientada a Objetos com Jogos 2D

Este é um Trabalho de Conclusão de Curso de autoria de Caminha (2012b) que tem como objetivo documentar a experiência do autor com ensino de Orientação a Objetos baseada em jogos. O autor começa por relatar como era a experiência de ensinar Orientação a Objetos seguindo os métodos didáticos mais comuns e após faz uma comparação com o aproveitamento do conteúdo por uma turma utilizando um *framework* de Jogos 2D para

aprender Orientação a Objetos. Por meio desse *framework*, que se chama javaPlay, o autor ensinou os conceitos de Orientação a Objetos para sua turma, como Composição, Encapsulamento, Herança e Polimorfismo. Na Figura 4 é possível visualizar um jogo criado com o framework.

Figura 4 – Jogo criado com o javaPlay



Segundo Caminha (2012b, p.42), ao final do módulo, ficou evidente o aumento da motivação dos alunos, principalmente pela qualidade de muitos dos jogos apresentados ao final da disciplina. Porém, a ênfase em um grande conjunto de códigos previamente elaborados que os alunos apenas modificavam acabou por gerar dificuldade em reconhecer quando e como aplicar um determinado conceito aprendido para um problema novo. Por exemplo, os alunos tinham facilidade em entender quando usar herança para os casos semelhantes aos apresentados em sala, mas não em casos diferentes.

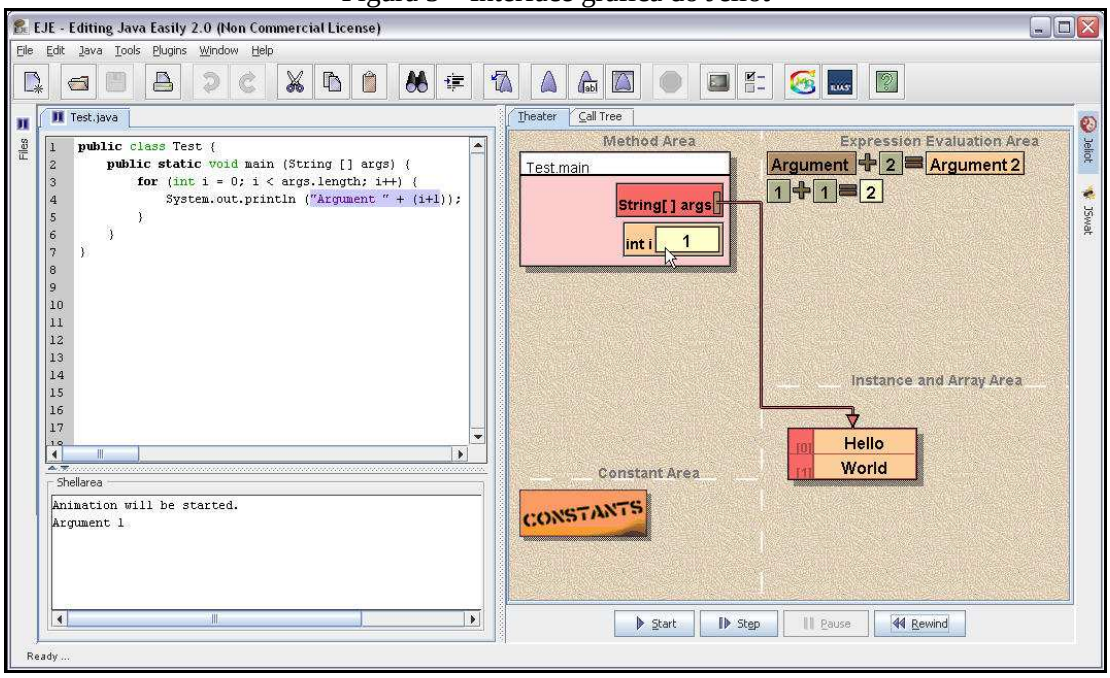
Caminha (2012b) acredita que a metodologia adotada carece de maturidade na elaboração de atividades e exemplos. Esses exemplos devem permitir que os estudantes adquiram maturidade no processo de desenvolvimento e identifiquem adequadamente onde cada conceito pode ser aplicado.

2.7.3 Jeliot

O Jeliot é um sistema de animação de programas com o objetivo de ser utilizado em introdução ao ensino de programação (JELIOT 3, 2012). Os programas são animados

automaticamente, sem requerer modificações ou anotações por parte do usuário. O Jeliot pode ser utilizado por novatos em programação, assim como por seus professores que não precisam investir em aprender como preparar as animações. A Figura 5 mostra a interface gráfica do Jeliot.

Figura 5 – Interface gráfica do Jeliot



O Jeliot foi escrito em Java para ganhar o máximo de portabilidade. Ele anima programas escritos em Java, mas não está restrito a uso apenas em cursos que lidam com Java. Ele foi utilizado com sucesso em cursos que lidavam com outras linguagens de programação, como PASCAL (JELIOT 3, 2012, p. 4).

De acordo com Ben-Ari, Levy e Uronen (2000, p. 3), que realizaram um experimento com a ferramenta durante um ano de curso, a nota média obtida pelos alunos aumentou com relação aos alunos que não usaram o Jeliot. Com o experimento realizado por Ben-Ari, Levy e Uronen (2000, p. 4), os autores do experimento concluíram que a animação é uma ferramenta que deve ser integrada nas salas de aula, além disso a interpretação de animações deve ser ensinada aos alunos. O principal benefício das animações do Jeliot é oferecer um modelo concreto de execução que todos os alunos precisam para poder entender algoritmos e programação.

2.7.4 Metodologia e Ferramenta para Ensino da Orientação a Objetos

O objetivo deste trabalho correlato apresentado por Mariane Galhardo (2004) é apresentar uma metodologia para acompanhamento do aluno durante a resolução de exercícios no paradigma orientado a objetos. Nessa metodologia o docente propõe exercícios

nos quais o aluno pode construir passo a passo um programa no paradigma citado, sem se preocupar com a estrutura da linguagem adotada. Assim que o aluno submete o exercício o professor pode verificá-lo indicando se os objetivos propostos foram atingidos, conduzindo o aluno a reforçar seus estudos em conceitos não assimilados ou propondo novos exercícios que abordem outros conceitos (GALHARDO, 2004, p. 1).

A ferramenta permite que o aluno construa seu programa por meio de interações, sem se preocupar com os aspectos estruturais da linguagem se concentrando em transpor os conceitos aprendidos para a modelagem do problema. Ao final código na linguagem Java é gerado como resultado das interações realizadas pelo aluno (GALHARDO, 2004, p. 2). O professor cadastra os exercícios dividindo os mesmos por assunto e definindo a descrição de cada um. Durante o cadastro dos exercícios, o professor define também o próximo exercício que o aluno deve resolver, caso o exercício resolvido pelo aluno esteja correto (GALHARDO, 2004, p. 3).

O aluno escolhe o exercício, resolve o mesmo e salva. Em seguida, o docente é avisado que o exercício já foi resolvido e começa o processo de correção do mesmo. A ferramenta envia automaticamente um *e-mail* ao aluno avisando que o exercício foi corrigido, assim que o docente finaliza este procedimento. Se o exercício estiver correto, o aluno passa para o próximo da sequência. Caso contrário, o aluno deve refazer o exercício para procurar sanar suas falhas de aprendizagem (GALHARDO, 2004, p. 3).

Durante o processo de resolução do exercício o aluno fornece, gradativamente, todos os dados necessários para construção do código-solução, que partem das características gerais da classe, passam pela definição dos atributos e métodos e terminam com a definição dos valores iniciais dos atributos para serem utilizados pelo construtor da classe. Ao finalizar a resolução do exercício a ferramenta gera para o aluno o código correspondente aos dados que ele informou em Java. Durante o processo de interação com a ferramenta, o aluno não necessita se preocupar com a sintaxe da linguagem. Ele concentra-se na busca da solução do exercício, apoiado nas definições conceituais de um código orientado a objetos. Porém, como resultado deste processo de interação ele vê o código gerado em uma linguagem pertencente a este paradigma (GALHARDO, 2004, p. 6).

Essa ferramenta acompanha o aluno passo a passo no desenvolvimento de uma classe gerada a partir de um problema proposto e possibilita o professor acompanhar esse processo através da *Web*. Assim o aluno pode ter um retorno mais rápido e pode dar continuidade na sequência de exercícios que foi proposta pelo professor através da mesma ferramenta. Uma característica dessa ferramenta é a dependência de *e-mail* para informar ao aluno sobre a correção dos problemas. Outra característica é que ela não pode ser utilizada em sala de aula por necessitar de computadores para os alunos, sendo necessário um laboratório com vários computadores para utilização.

3 DESENVOLVIMENTO

Neste capítulo são apresentadas as etapas envolvidas no desenvolvimento da ferramenta. Na Seção 3.1 são apresentados os requisitos funcionais e não funcionais do trabalho. Na Seção 3.2 é feita a especificação da ferramenta através de diagramas de uso e de classe. Por fim, na Seção 3.3 são apresentados detalhes da fase de implementação da ferramenta.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

A ferramenta desenvolvida nesse trabalho foi batizada de *OO Programming Lab* e tem como objetivo permitir que o aluno crie classes e objetos através de simples cadastros na tela de seu *smartphone*. Deve permitir também que sejam inseridos atributos e métodos nessas classes criadas. Outra funcionalidade da ferramenta é que ela deve permitir que o aluno visualize os valores dos atributos de um objeto criado, e também chame os métodos desse objeto.

Os requisitos funcionais são:

a) permitir o cadastro de usuários (alunos ou professores);
b) permitir a um professor criar uma sala de estudos;
c) permitir a um professor atribuir um líder(aluno) a uma sala de estudos;
d) permitir que um professor faça *login*;
e) permitir que um aluno faça *login* e se integre a uma sala de estudos já criada;
f) permitir a qualquer aluno criar uma classe, com seus métodos e atributos, ou aprimorar uma classe existente em sua sala de estudos;
g) permitir ao aluno-líder excluir classes, métodos e atributos;
h) permitir a um aluno criar um objeto a partir das classes existentes e compartilhá-lo na sala, bem como destruir seus objetos da sala;
i) permitir a um aluno fazer a chamada de um método de algum objeto da sala, incluindo parâmetros, e receber seu retorno;
j) permitir acompanhar a execução de um método, identificando os demais métodos chamados e seus respectivos objetos (similar a um *stack trace*);
k) permitir ao aluno observar o estado de qualquer objeto da sala;
l) permitir que sejam definidas cores para as classes, para fácil identificação dos objetos criados a partir de determinada classe;
m) permitir ao professor salvar e recuperar as classes criadas em uma sala;

n) guardar toda a interação de uma execução da solução desenvolvida para posterior análise do professor.

Os requisitos não funcionais são:

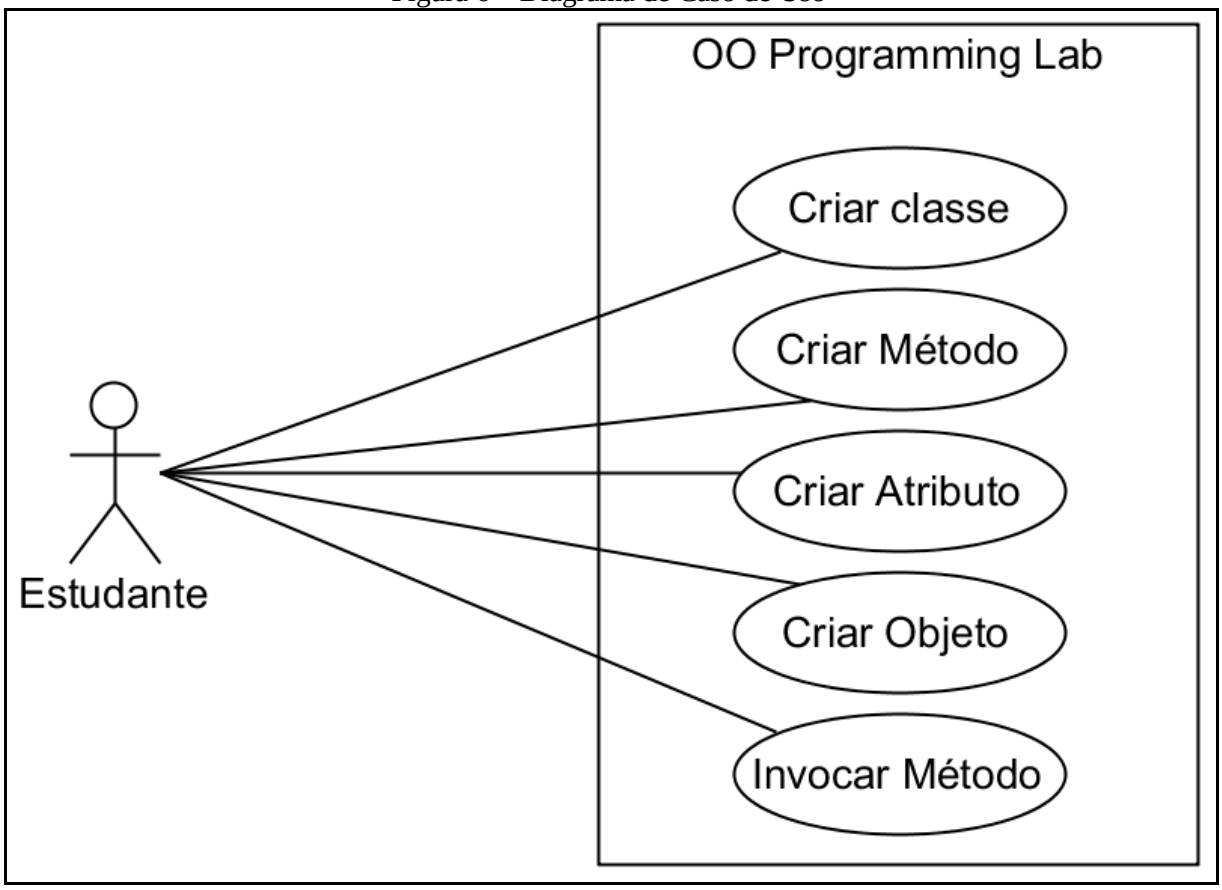
a) ser compatível com Windows Phone e Android;
b) ser implementado em C# utilizando a IDE Visual Studio;
c) utilizar o Xamarin para portar a ferramenta para Android;
d) utilizar o Visual Studio para criar os diagramas de classes e de casos de usos;
e) utilizar o Microsoft SQL Server para manter os dados.

3.2 ESPECIFICAÇÃO

O diagrama de casos de uso apresentado nesta seção demonstra a funcionalidade da ferramenta do ponto de vista do estudante e os diagramas de classes apresentam a representação lógica dos dados da ferramenta. Os diagramas de classes foram gerados utilizando o Visual Studio 2013 Community e o diagrama de casos de uso foi gerado utilizando a ferramenta Visual Paradigm 12.1.

Na Figura 6 está representado o diagrama de casos de uso, mostrando as interações entre o ator Estudante e a ferramenta *OO Programming Lab* para Windows Phone.

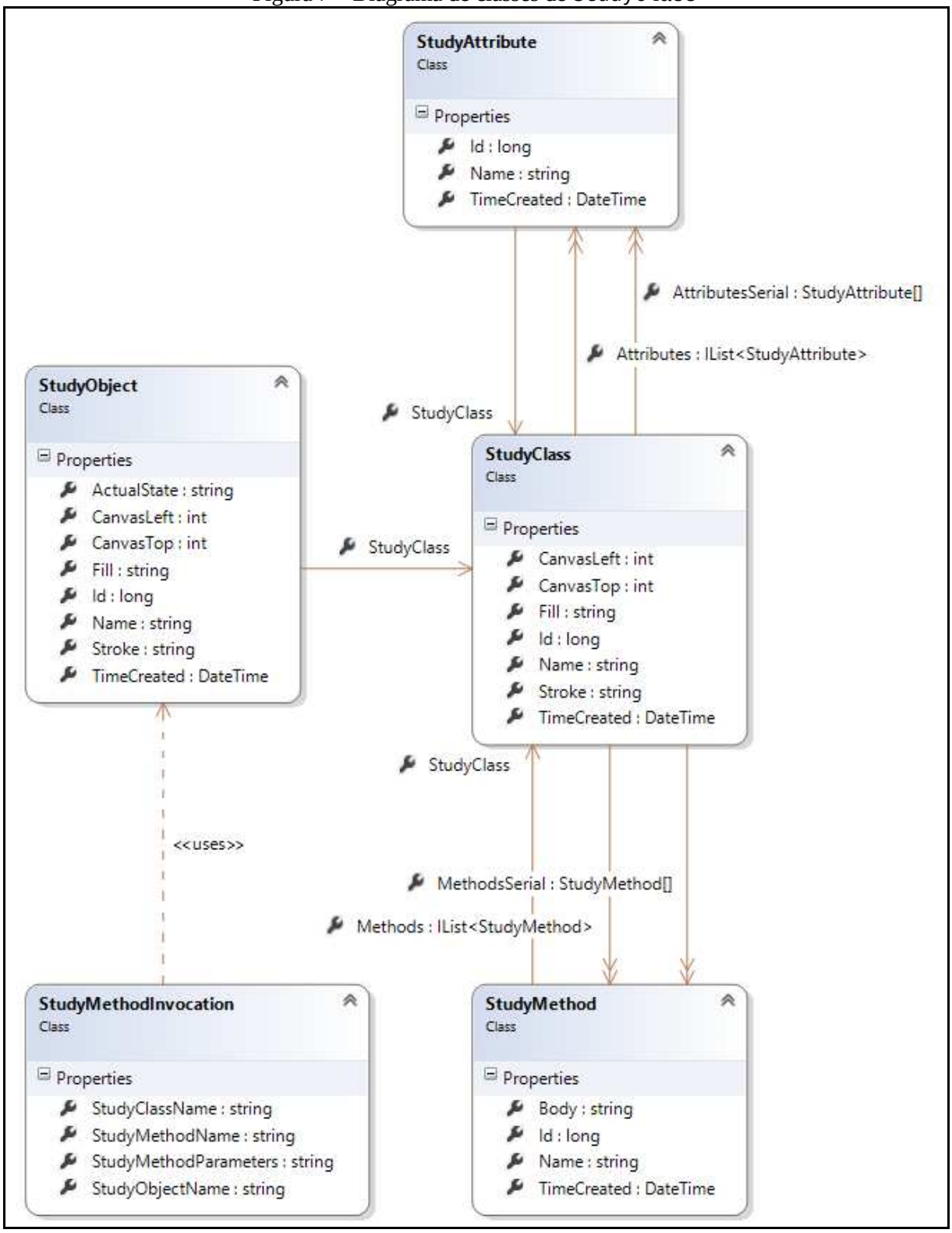
Figura 6 – Diagrama de Caso de Uso



Através do aplicativo para Windows Phone o estudante pode criar classes, métodos, atributos e objetos. No momento em que esses elementos são criados, o aplicativo para Windows Phone passa as informações para o servidor, que irá persistir no banco. O estudante também pode fazer a invocação de um método de um objeto. Os dados dessa invocação, como o método, o objeto alvo, e os parâmetros, são passadas ao servidor que irá gerar o arquivo .cs da classe, compilar essa classe, gerando uma dll. Tendo essa dll gerada, o servidor carrega essa dll, e faz a chamada dinamicamente do método, retornando seu resultado.

A Figura 7 mostra como estão relacionados os dados das classes e objetos criados pelos alunos, assim como os métodos e os atributos. A classe `StudyClass` é o centro da estrutura, que possui uma lista de `StudyAttribute` e `StudyMethod`. A classe `StudyObject` representa um objeto criado pelo aluno e possui um atributo ligando à classe da qual o objeto foi criado (`StudyClass`). A classe `StudyMethodInvocation` representa uma invocação de método pelo aluno.

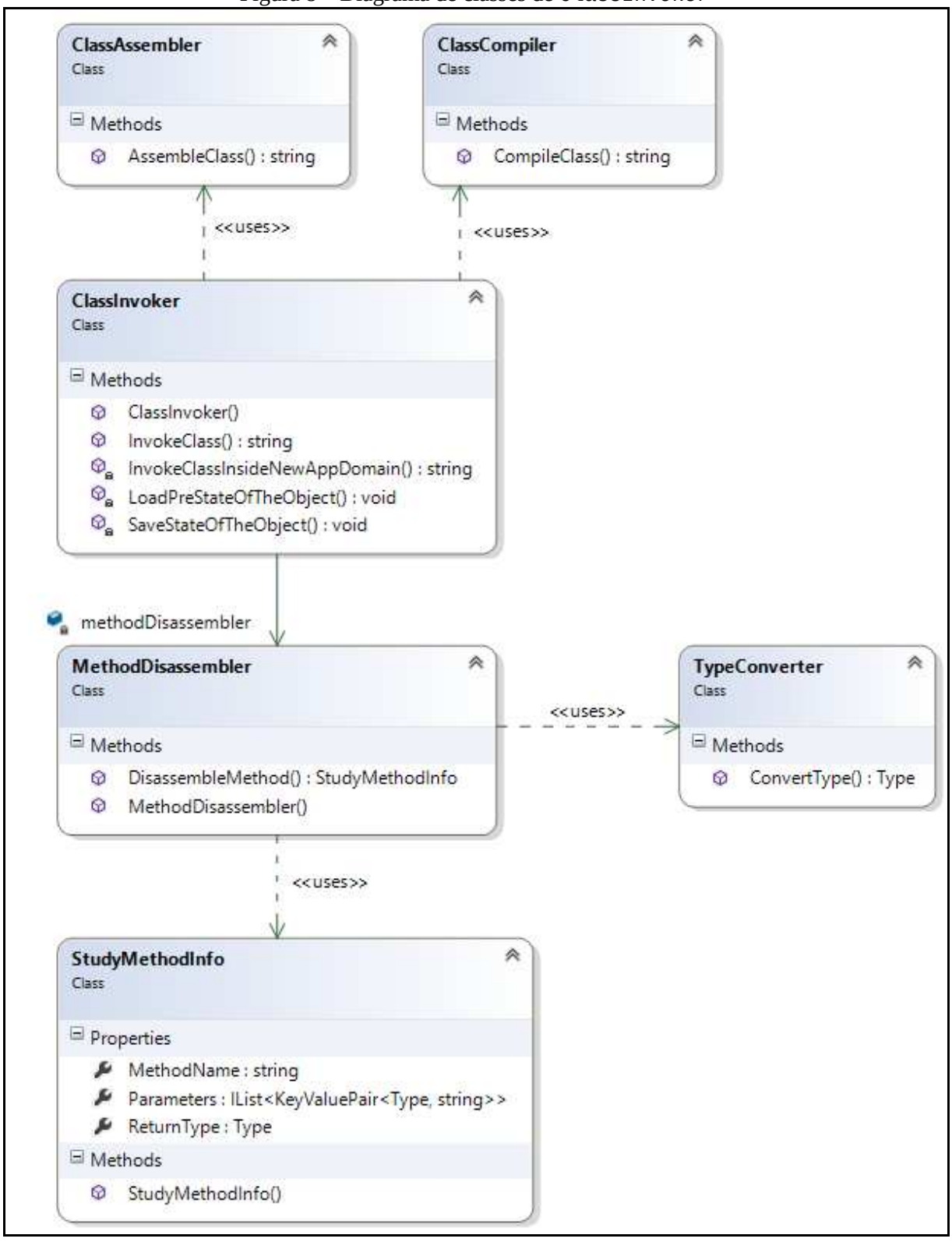
Figura 7 – Diagrama de classes de `StudyClass`



Na Figura 8 estão representadas as classes `ClassAssembler`, `ClassCompiler` e `ClassInvoker` que fazem os trabalhos respectivamente de montar o arquivo .cs da classe, compilar a classe em forma de dll e carregar a dll e fazer a invocação do método através de

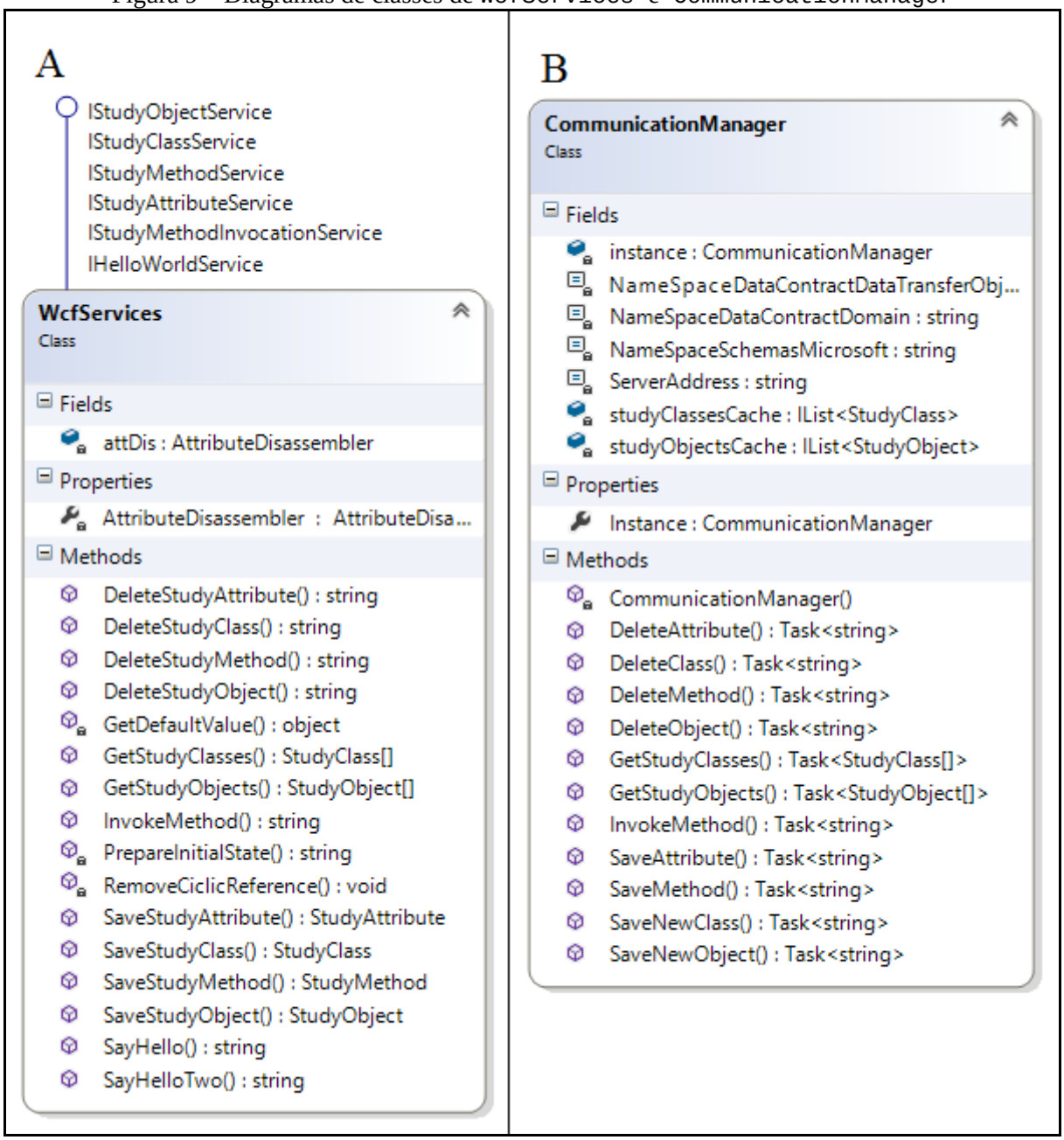
reflexão computacional. As classes `MethodDissassembler` e `StudyMethodInfo` auxiliam esse processo.

Figura 8 – Diagrama de classes de `ClassInvoker`



Na Figura 9.A está representada a classe `WcfServices`, que é a classe por trás do serviço WCF no lado do servidor fazendo o processamento das requisições que o servidor recebe. Na Figura 9.B está representada a classe `CommunicationManager` que é quem faz a conexão do Windows Phone com os serviços WCF.

Figura 9 – Diagramas de classes de `WcfServices` e `CommunicationManager`



3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas e a operacionalidade da implementação.

3.3.1 Técnicas e ferramentas utilizadas

Na implementação foi utilizado o Visual Studio 2013 Communitty com o Windows Phone SDK. A programação da interface de usuário do Windows Phone é feita utilizando XAML. No Quadro 3 apresenta-se o código fonte XAML da página principal do aplicativo para Windows Phone com uma seção da página sendo criada na linha 23 para a tela de classes e outra seção na linha 33 para a tela de objetos. Os *Canvas* (áreas de desenho) que são criados na linha 28 e 38 serão populados em tempo de execução com as classes e os objetos criados pelos estudantes.

Quadro 3 – Código fonte XAML – MainPage

```
01  <Page
02      x:Class="OOProgrammingLabClient.MainPage"
03      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
04      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
05      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
06  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
07      mc:Ignorable="d"
08      Background="{ThemeResource ApplicationPageBackgroundThemeBrush}"
09  Loaded="Page_Loaded">
10
11      <Grid>
12          <StackPanel Grid.Row="0" Margin="10,0,0,0">
13              <TextBlock Text="OO Programming Lab"
14                         Style="{ThemeResource TitleTextBlockStyle}"
15                         Margin="0,12,0,0"/>
16              <TextBlock Text="Sala #1" Margin="0,0,0,26"
17                         Style="{ThemeResource HeaderTextBlockStyle}"
18      CharacterSpacing="{ThemeResource PivotHeaderItemCharacterSpacing}"
19                         Height="77"/>
20          </StackPanel>
21
22          <Hub Margin="0,150,0,0">
23              <HubSection Name="ClassesHubSection" Header="Classes">
24                  <DataTemplate>
25                      <StackPanel Margin="0,-32,0,0">
26                          <Button FontSize="14" Name="NewClassButton"
27                       Click="NewClassButton_Click">Nova Classe</Button>
28                          <Canvas Name="ClassCanvas">
29                          </Canvas>
30                      </StackPanel>
31                  </DataTemplate>
32              </HubSection>
33              <HubSection Name="ObjectsHubSection"  Header="Objetos">
34                  <DataTemplate>
35                      <StackPanel Margin="0,-32,0,0">
36                          <Button FontSize="14" Name="NewObjectButton"
37                      Click="NewObjectButton_Click">Novo Objeto</Button>
38                          <Canvas Name="ObjectCanvas">
39                          </Canvas>
40                      </StackPanel>
41                  </DataTemplate>
42              </HubSection>
43          </Hub>
44      </Grid>
45  </Page>
```

A comunicação do Windows Phone com o servidor foi feita via HTTP Post, uma vez que o *framework* do Windows Phone não dá suporte a trabalhar diretamente com o WCF. O

código fonte de comunicação com o serviço WCF pode ser observado no Quadro 4. A imagem mostra a conexão com o método para salvar uma nova classe onde nas linhas 9 a 12 a posição da classe no Canvas é gerada aleatoriamente. Nas linhas 15 a 24 a classe é serializada para ser passada como parâmetro. Nas linhas 31 a 36 é gerada a *stream* de requisição para o servidor. Por fim na linha 48, é obtida a *stream* de resposta do servidor com o resultado da solicitação.

Quadro 4 – Código fonte para conexão com o servidor – Salvar Nova Classe

```
01  public async Task<string> SaveNewClass(StudyClass studyClass)
02  {
03      const string service = "StudyClass";
04      const string method = "SaveStudyClass";
05      Uri requestUri = new Uri(string.Format("{0}/{1}/{2}",
06                                       ServerAddress, service, method));
07      string responseFromServer = "Sem resposta!";
08
09      var random = new Random(DateTime.Now.Day * 1000000 +
10          DateTime.Now.Hour * 100000 + DateTime.Now.Minute * 10000 +
11          DateTime.Now.Millisecond);
12      studyClass.CanvasTop = random.Next(10, 370);
13      studyClass.CanvasLeft = random.Next(20, 260);
14
15      XmlSerializer xmlSerializer = new
16                       XmlSerializer(typeof(StudyClass),
17                      "http://schemas.datacontract.org/2004/07/
18                       OOProgrammingLabServer.Interface.Domain");
19      StringWriter stringWriter = new StringWriter();
20      XmlWriter xmlWriter = XmlWriter.Create(stringWriter, new
21                            XmlWriterSettings { OmitXmlDeclaration =
22                            true });
23      xmlSerializer.Serialize(xmlWriter, studyClass);
24      var xml = stringWriter.ToString();
25
26      HttpWebRequest httpWebRequest = WebRequest.Create(requestUri) as
27                                                          HttpWebRequest;
28      httpWebRequest.ContentType = "application/xml; charset=utf-8";
29      httpWebRequest.Method = "POST";
30
31      using (Stream requestStream = await
32                           httpWebRequest.GetRequestStreamAsync()) {
33          byte[] xmlAsBytes = Encoding.UTF8.GetBytes(xml);
34          await requestStream.WriteAsync(xmlAsBytes, 0,
35                                                  xmlAsBytes.Length);
36      }
37
38      try {
39          XmlSerializer xmlDeserializer = new
40                            XmlSerializer(typeof(StudyClass),
41                            "http://schemas.datacontract.org/2004/07/
42                            OOProgrammingLabServer.Interface.Domain");
43          WebResponse webResponse = await
44                                    httpWebRequest.GetResponseAsync();
45          StudyClass newStudyClass;
46
47          using (var reader =
48                 new StreamReader(webResponse.GetResponseStream())) {
49              newStudyClass = xmlDeserializer.Deserialize(reader) as
50                                                        StudyClass;
51              if (newStudyClass != null && newStudyClass.Id > 0) {
52                  responseFromServer = "Salvo com sucesso!";
53              }
54              else {
55                  responseFromServer = "Erro ao obter a resposta!";
```

```
56              }
57          }
58      }
59      catch (WebException webException) {
60          responseFromServer = string.Format("Erro ao comunicar com o
61                                servidor. {0}", webException.Message);
62      }
63      return responseFromServer;
64  }
```

O servidor foi programado utilizando a tecnologia WCF da Microsoft. Como o *framework* do Windows Phone não dá suporte para trabalhar diretamente com o WCF, o serviço do lado do servidor foi programado utilizando arquitetura RESTful para poder ser facilmente utilizado através de HTTP Post. No Quadro 5 está representado o código fonte para levantar o serviço WCF.

Quadro 5 – Código fonte do serviço WCF

```
01  public class ServiceHostManager
02  {
03      private ServiceHost serviceHost;
04
05      public void HostService(Uri baseUri, Action progressBarUpdate)
06      {
07          if (serviceHost != null)
08              CloseService();
09
10          serviceHost = new ServiceHost(typeof(WcfServices), baseUri);
11          progressBarUpdate();
12
13          var helloWorldServiceEndpoint =
14              serviceHost.AddServiceEndpoint(typeof(IHelloWorldService),
15                                    new WebHttpBinding(), "HelloWorld");
16          helloWorldServiceEndpoint.Behaviors.Add(new
17                                                      WebHttpBehavior());
18          helloWorldServiceEndpoint.Name =
19                                    "WebHttpBinding_IHelloWorldService";
20          helloWorldServiceEndpoint.Contract.ConfigurationName =
21                           "OOProgrammingLabServer.IHelloWorldService";
22          progressBarUpdate();
23
24          var studyClassServiceEndpoint =
25              serviceHost.AddServiceEndpoint(typeof(IStudyClassService),
26                                    new WebHttpBinding(), "StudyClass");
27          studyClassServiceEndpoint.Behaviors.Add(new
28                                                      WebHttpBehavior());
29          studyClassServiceEndpoint.Name =
30                                    "WebHttpBinding_IStudyClassService";
31          studyClassServiceEndpoint.Contract.ConfigurationName =
32                           "OOProgrammingLabServer.IStudyClassService";
33          progressBarUpdate();
34
35          var studyObjectServiceEndpoint =
36             serviceHost.AddServiceEndpoint(typeof(IStudyObjectService),
37                                   new WebHttpBinding(), "StudyObject");
38          studyObjectServiceEndpoint.Behaviors.Add(new
39                                                      WebHttpBehavior());
40          studyObjectServiceEndpoint.Name =
41                                   "WebHttpBinding_IStudyObjectService";
42          studyObjectServiceEndpoint.Contract.ConfigurationName =
43                          "OOProgrammingLabServer.IStudyObjectService";
44          progressBarUpdate();
45
46          var studyMethodServiceEndpoint =
47             serviceHost.AddServiceEndpoint(typeof(IStudyMethodService),
48                                   new WebHttpBinding(), "StudyMethod");
```

```
49          studyMethodServiceEndpoint.Behaviors.Add(new
50                                                      WebHttpBehavior());
51          studyMethodServiceEndpoint.Name =
52                                      "WebHttpBinding_IStudyMethodService";
53          studyMethodServiceEndpoint.Contract.ConfigurationName =
54                              "OOProgrammingLabServer.IStudyMethodService";
55          progressBarUpdate();
56
57          serviceHost.Description.Behaviors.Add(new
58                     ServiceMetadataBehavior { HttpGetEnabled = true });
59          serviceHost.Open();
60          progressBarUpdate();
61      }
62
63      public void CloseService()
64      {
65          serviceHost.Close();
66      }
67  }
```

A implementação dos serviços no servidor se faz por meio da implementação das interfaces dos próprios serviços. O Quadro 6 mostra o código fonte do serviço para salvar uma nova classe.

Quadro 6 – Código fonte de implementação dos serviços – Salvar nova classe

```
01      public StudyClass SaveStudyClass(StudyClass studyClass)
02      {
03          using (var session = NHibernateHelper.OpenSession())
04          using (var transaction = session.BeginTransaction())
05          {
06              session.SaveOrUpdate(studyClass);
07              transaction.Commit();
08          }
09
10          return studyClass;
11      }
12      public StudyClass[] GetStudyClasses()
13      {
14          IList<StudyClass> studyClasses;
15          using (var session = NHibernateHelper.OpenSession())
16          using (session.BeginTransaction())
17          {
18              studyClasses = session.QueryOver<StudyClass>().List();
19          }
20
21          foreach (var studyClass in studyClasses)
22          {
23              RemoveCiclicReference(studyClass);
24          }
25
26          return studyClasses.ToArray();
27      }
```

No servidor, os dados são armazenados em um banco de dados do Microsoft SQL Server 2014. Para salvar, editar e acessar esses dados foi utilizado o framework NHibernate por possuir facilitadores para trabalhar com a sessão, transação e fazer os *updates* no banco. Por exemplo, ao editar uma classe e chamar o NHibernate para persistir esse dado no banco, ele já automaticamente monta o *update* apenas para as colunas alteradas. A configuração pode ser vista no Quadro 7, onde é mostrado o código fonte no qual as configurações para conexão com o banco são carregadas. Para o funcionamento do NHibernate também é necessário fazer

a configuração do mapeamento das tabelas. No Quadro 8 está representado o código fonte de mapeamento da tabela `STUDY_OBJECTS`. O Quadro 9 apresenta a configuração de conexão ao banco que é feita via XML.

Quadro 7 – Código para configuração do NHibernate

```
01 public class NHibernateHelper
02 {
03     private static ISessionFactory sessionFactory;
04 
05     private static ISessionFactory SessionFactory
06     {
07         get
08         {
09             if (sessionFactory != null)
10                 return sessionFactory;
11 
12             var configuration = new Configuration();
13             configuration.Configure("NHibernate.xml");
14 
15             var mapping = GetMappings();
16             configuration.AddDeserializedMapping(mapping, null);
17 
18             sessionFactory = configuration.BuildSessionFactory();
19 
20             return sessionFactory;
21         }
22     }
23 
24     private static HbmMapping GetMappings()
25     {
26         var mapper = new ModelMapper();
27 
28         mapper.AddMappings(Assembly.GetAssembly(typeof(StudyObjectMap))
29               .GetExportedTypes());
30         var mapping =
31                   mapper.CompileMappingForAllExplicitlyAddedEntities();
32 
33         return mapping;
34     }
35 
36     public static ISession OpenSession()
37     {
38         return SessionFactory.OpenSession();
39     }
40 }
```

Quadro 8 – Código para mapeamento da tabela `STUDY_OBJECTS`

```
01  public class StudyObjectMap : ClassMapping<StudyObject>
02  {
03      public StudyObjectMap()
04      {
05          Table("STUDY_OBJECTS");
06          Id(x => x.Id, map => map.Generator(Generators.Native));
07          Property(x => x.Name, prop => { prop.Column("NAME");
08                                          prop.NotNullable(true); });
09          Property(x => x.Fill, prop => { prop.Column("FILL");
10                                          prop.NotNullable(true); });
11          Property(x => x.Stroke, prop => { prop.Column("STROKE");
12                                            prop.NotNullable(true); });
13          Property(x => x.ActualState, prop => {
14                                            prop.Column("ACTUAL_STATE");
15                                            prop.NotNullable(true); });
16          Property(x => x.CanvasTop, prop => {
17                                             prop.Column("CANVAS_TOP");
18                                             prop.NotNullable(true); });
19          Property(x => x.CanvasLeft, prop => {
20                                             prop.Column("CANVAS_LEFT");
21                                             prop.NotNullable(true); });
22          Property(x => x.TimeCreated, prop => {
23                                            prop.Column("TIME_CREATED");
24                                            prop.NotNullable(true); });
25          ManyToOne(x => x.StudyClass, prop =>
26          {
27              prop.Column("STUDY_CLASS_ID");
28              prop.NotNullable(true);
29              prop.Lazy(LazyRelation.NoLazy);
30          });
31      }
32  }
```

Quadro 9 – XML de configuração do NHibernate

```
01  <?xml version="1.0" encoding="utf-8" ?>
02  <hibernate-configuration  xmlns="urn:nhibernate-configuration-2.2" >
03    <session-factory name="NHibernate">
04      <property name="connection.driver_class">
05        Nhibernate.Driver.SqlClientDriver
06      </property>
07      <property name="connection.connection_string">
08        Server=.\SQLEXPRESS;Database=OOProgrammingLab;
09        User ID=sa;Password=sa
10      </property>
11      <property name="dialect">
12        NHibernate.Dialect.MsSql2008Dialect
13      </property>
14      <property name="show_sql">true</property>
15    </session-factory>
16  </hibernate-configuration>
```

Durante o processo de desenvolvimento foi utilizada a técnica de TDD, sempre desenvolvendo código de teste antes do código de implementação. O *framework* de testes utilizado foi o disponibilizado pela própria Microsoft através do Visual Studio, chamado Microsoft.VisualStudio.TestTools. No Quadro 10 está representado um método de teste criado para guiar o desenvolvimento da classe `ClassInvoker`. Após escrito esse teste foi desenvolvido o código da funcionalidade, demonstrado no Quadro 11. Tendo o teste passado, então foram feitas as refatorações, para melhorar o código e remover possíveis duplicações código, sem quebrar o teste.

Quadro 10 – Código de teste da classe `ClassInvoker`

```
01  [TestMethod]
02  public void
03         DeveChamarUmMetodoDaClasseCriadaCarregandoOEstadoAtualDoObjeto()
04  {
05      var studyMethodInvocation = new StudyMethodInvocationBuilder()
06          .ConfigureAddValueMethod()
07          .Build();
08
09      var studyObject = new StudyObjectBuilder()
10          .WithStudyClass(new StudyClassBuilder().Build())
11          .WithActualState("<MinhaClasse>
12                              <minhaString>Oi</minhaString>
13                              <meuInteiro>10</meuInteiro>
14                            </MinhaClasse>")
15          .Build();
16
17      var classLocation = classAssembler.AssembleClass(studyObject,
18                                                  studyMethodInvocation);
19      var libraryLocation =
20                           classCompiler.CompileClass(classLocation);
21      var result = classInvoker.InvokeClass(libraryLocation,
22                              studyMethodInvocation, studyObject);
23
24      Assert.AreEqual("15", result);
25      Assert.AreEqual("<MinhaClasse>
26                          <minhaString>Oi</minhaString>
27                          <meuInteiro>15</meuInteiro>
28                       </MinhaClasse>",
29                       studyObject.ActualState);
30  }
```

Quadro 11 – Código da classe `ClassInvoker`

```
01  public class ClassInvoker
02  {
03      private readonly MethodDisassembler methodDisassembler;
04
05      public ClassInvoker(MethodDisassembler methodDisassembler)
06      {
07          this.methodDisassembler = methodDisassembler;
08      }
09
10      public string InvokeClass(string libraryLocation,
11                           StudyMethodInvocation studyMethodInvocation,
12                           StudyObject studyObject)
13      {
14          Assembly assembly = Assembly.LoadFrom(libraryLocation);
15          Type loadedType = assembly.GetType(
16          string.Format("OOProgrammingLabServer.CompiledClasses.{0}",
17                                  studyMethodInvocation.StudyClassName));
18          dynamic createdObject = Activator.CreateInstance(loadedType)
19                                                              as dynamic;
20
21          StudyMethodInfo studyMethodInfo = methodDisassembler
22             .DisassembleMethod(studyMethodInvocation.StudyMethodName);
23          MethodInfo methodInfo = loadedType.GetMethod(
24                                            studyMethodInfo.MethodName);
25          string[] parameterValues = studyMethodInvocation
26                                .StudyMethodParameters.Split(new []{'|'},
27                                  StringSplitOptions.RemoveEmptyEntries);
28
29          if (studyMethodInfo.Parameters.Count !=
30                                                 parameterValues.Count())
31              throw new InvalidOperationException("O número de
32                        parâmetros informados não está de acordo com o
33                        número de parâmetros esperado");
34
35          IList<object> parametersAsObjects = new List<object>();
36
```

```
37          for (int ind = 0; ind < studyMethodInfo.Parameters.Count;
38                                                                  ind++)
39          {
40              Type parameterType = studyMethodInfo.Parameters[ind].Key;
41              object paramenterValue = Convert.ChangeType(
42                                  parameterValues[ind], parameterType);
43
44              parametersAsObjects.Add(paramenterValue);
45          }
46
47          dynamic result = methodInfo.Invoke(createdObject,
48                                  parametersAsObjects.ToArray())
49                              ?? "Método executado sem retorno.";
50
51          return result.ToString();
52      }
53  }
```

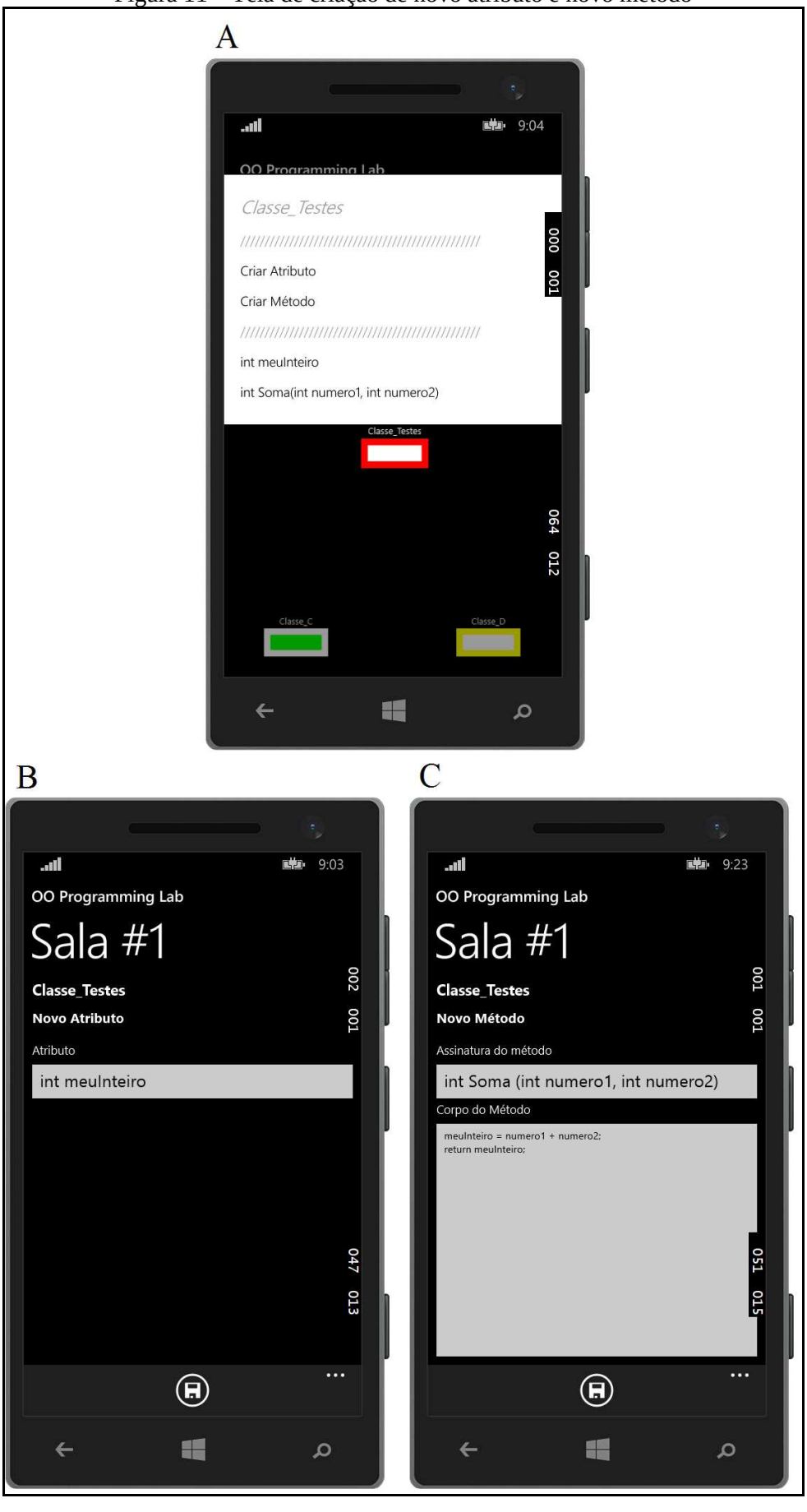
3.3.2 Operacionalidade da implementação

Ao iniciar o aplicativo no Windows Phone (*OO Programming Lab*), ele conecta com o servidor e apresenta as classes já criadas, conforme a Figura 10.A. No topo da tela há um botão que dá a opção de criar uma nova classe que leva à tela de criação de nova classe representada na Figura 10.B. As classes são representadas através de retângulos, apenas para visualização. Elas não podem ser ligadas como ocorre em um diagrama de classes. No entando através delas, pode-se criar atributos e métodos.

Figura 10 – Tela de classes

Pressionando sobre a classe, surge um menu com as opções de criar um novo método, um novo atributo, ou editar os métodos e atributos já existentes, conforme mostra a Figura 11.A. A Figura 11.B apresenta a tela de criação de um novo atributo e a Figura 11.C apresenta a tela de criação de um novo método.

Figura 11 – Tela de criação de novo atributo e novo método

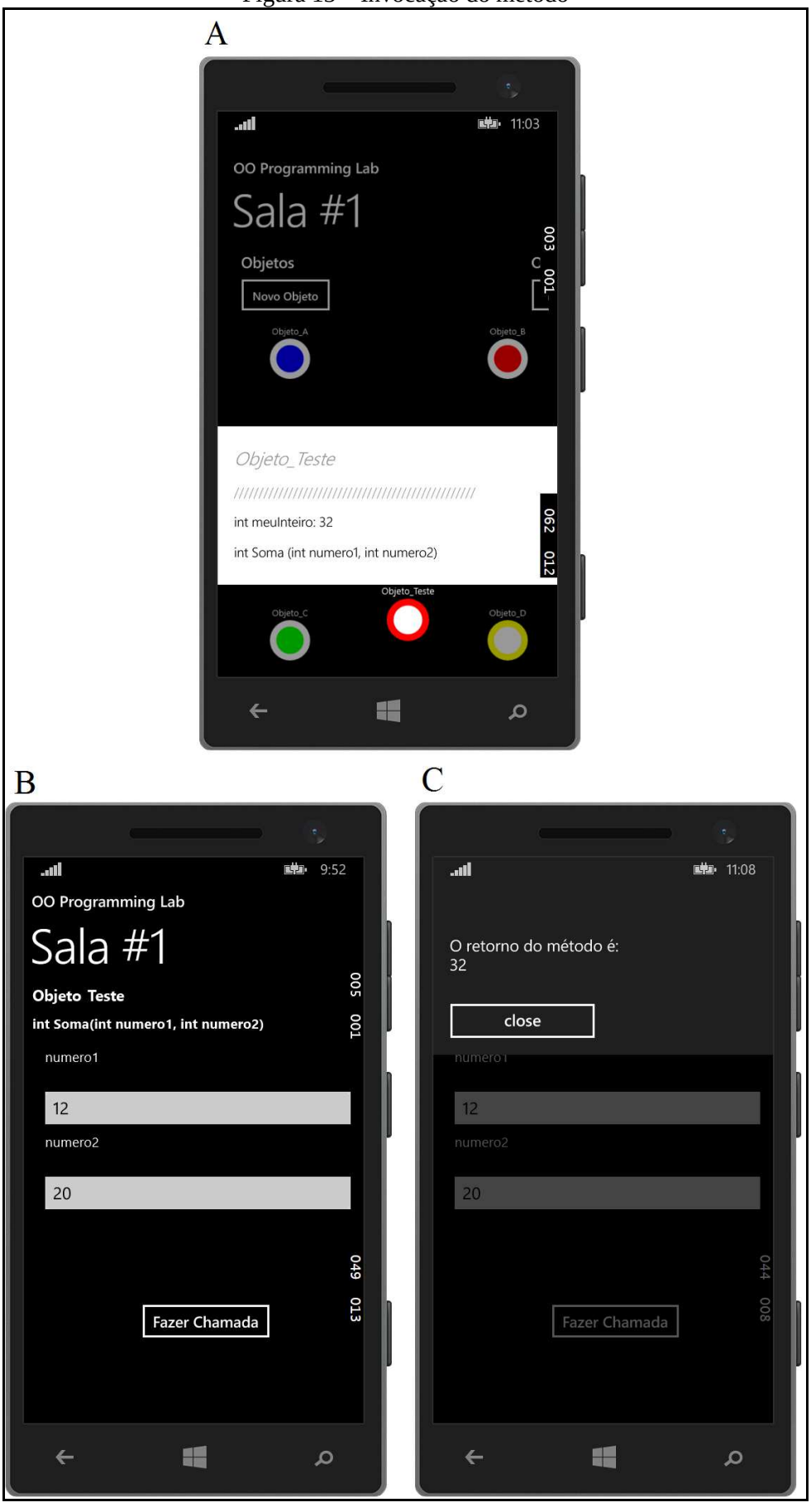


Uma vez a classe criada, é possível ir para a tela de objetos e criar novos objetos com base nessas classes. Na Figura 12.A está apresentada a tela de objetos e a Figura 12.B representa a tela de criação de objetos. Os objetos criados terão o mesmo esquema de cores da classe para facilitar a associação pelo aluno.

Figura 12 – Tela de objetos

A partir do objeto criado é possível pressionar sobre o mesmo para visualizar o valor atual de seus atributos, ou invocar os seus métodos, conforme mostra a Figura 13.A. Ao tocar a linha com o nome do método é aberta a tela de invocação de método, como mostra a Figura 13.B. Quando o servidor retorna o resultado da invocação do método, o retorno é apresentado ao usuário conforme Figura 13.C.

Figura 13 – Invocação do método



3.4 RESULTADOS E DISCUSSÕES

Como resultado conseguiu-se obter uma ferramenta que pode ser utilizada por alunos para cooperativamente criarem e editarem classes e objetos. A ferramenta não alcançou o objetivo de poder observar a execução do código, identificando os métodos chamados e seus respectivos objetos. Apesar disso, a ferramenta permite que sejam criados e editados métodos que podem ser executados, passando os parâmetros desejados, e atributos que podem ter seu valor inspecionado no objeto. Outro requisito que não foi alcançado foi a funcionalidade de cadastro e *login* de usuários (alunos e professores), nem de salas de estudo. A ferramenta apresenta uma sala única sem distinção de usuários com relação a permissões para criação ou exclusão de classes, objetos, métodos e atributos. Durante a execução, todos os dados são armazenados em banco Microsoft SQL Server 2014 no servidor e carregados toda vez que a aplicação é inicializada. Permitindo assim, que um professor possa analisar os objetos e classes criadas pelos alunos.

O uso de TDD ao longo do desenvolvimento da ferramenta foi de boa ajuda, para manter o código sempre funcionando. Algumas vezes era alterada alguma parte do sistema que acabava fazendo com que outra parte parasse de funcionar. Executando os testes com certa frequência era fácil identificar quando isso acontecia e corrigir. Também auxiliou bastante no desenvolvimento de pequenas partes do sistema, permitindo testar durante o desenvolvimento daquela funcionalidade sem ter que executar todo o sistema. O *framework* de testes disponibilizado pela Microsoft, chamado Microsoft.VisualStudio.TestTools, no entanto deixa um pouco a desejar no quesito assertivas de teste. O *framework* da Microsoft disponibiliza poucos métodos de comparação entre o valor esperado e o obtido durante um teste.

Uma das dificuldades encontradas foi fazer a conexão do Windows Phone com o servidor WCF. Pelo *framework* do Windows Phone não dar suporte a uma conexão direta com o WCF, foi necessário utilizar HTTP Posts para fazer a comunicação. Foi difícil fazer a serialização correta dos parâmetros que eram passados, porque qualquer erro, como uma falta de *namespace* ou até mesmo os atributos estarem serializados em uma ordem diferente, já fazia com que o servidor não conseguisse interpretar a requisição. Essa dificuldade não existe quando um aplicativo *desktop* conecta com o WCF, pois, nesse caso é possível utilizar as ferramentas de WCF disponíveis para fazer conexão, abstraindo toda a parte de comunicação e serialização dos parâmetros.

Outra dificuldade foi em utilizar o Xamarin para portar o aplicativo para Android. A primeira dificuldade foi o fato de que nem todo o código é convertido para o Android, conforme foi imaginado inicialmente. O que ocorre é que pode ser reaproveitado todo o código-fonte C#, desde que não tenha funções específicas relacionadas ao sistema operacional Windows, como pegar informações do calendário, por exemplo. Além disso, é necessário que toda a interface de usuário seja reescrita para o Android, ou seja, da interface de usuário do Windows Phone, nada é aproveitado. Para superar esta limitação, é preciso utilizar o Xamarin.Forms, que é um *toolkit* de interface ao usuário que permite criar essas interfaces de maneira que possam ser compartilhadas entre Windows Phone, Android e iOS. Mas quando se identificou isto, toda a interface ao usuário já havia sido criada através do Visal Studio, específica para o Windows Phone. Além dessas dificuldades, esbarrou-se com um erro ao tentar compilar o projeto que estava sendo convertido para o Android, o qual apenas informava "*Error: The operation could not be completed*" e não foi possível passar desse ponto.

A utilização do banco de dados Microsoft SQL Server foi acertada. Apesar do volume pequeno de dados nesta aplicação, mostrou-se um banco de dados confiável e não teve nenhum problema de inconsistência nos dados ou instabilidade durante o desenvolvimento.

Como foi utilizado o TDD para desenvolvimento da ferramenta, que auxilia no *design* da estrutura do código, o código-fonte foi escrito antes de serem gerados os diagramas de classes. Inicialmente pretendia-se utilizar o Visual Studio para gerar todo os diagramas necessários. No entanto verificou-se que apenas a versão Ultimate possuía a opção de diagramas de caso de uso. Para esse fim então utilizou-se a ferramenta Visual Paradigm, com a qual foi possível gerar os diagramas de caso de uso atendendo as necessidades desse trabalho.

Após concluída a implemtnação da ferramenta, foi realizada uma validação da mesma com uma turma de alunos. Nessa validação os alunos fizeram a utilização da ferramenta, criaram classes, atributos, métodos e objetos, e fizeram a chamada dos métodos. Ao final da validação foi pedido que os alunos respondessem um questionário de avaliação da ferramenta.

A ferramenta foi avaliada nos quesitos:

a) Funcionalidade: Avaliada, em uma nota de 1 a 5, a capacidade do software de fornecer funções as quais satisfazem as necessidades, obtendo nota média de 3,6;
b) Confiabilidade: Avaliada, em uma nota de 1 a 5, a capacidade do software de manter seu nível de performance sem apresentar quantidade excessiva de erros, obtendo nota média de 3,6;

c) Usabilidade: Avaliada, em uma nota de 1 a 5, a capacidade que o software possui com relação ao entendimento, aprendizagem e satisfação do usuário, obtendo nota média de 4.0;
d) Eficiência: Avaliada, em uma nota de 1 a 5, a capacidade do software de proporcionar o nível de desempenho exigido com relação ao tempo de resposta, obtendo uma nota média de 4,6.

Para que possam ser comparados os trabalhos correlatos com os resultados alcançados nessa ferramenta, demonstram-se as principais funcionalidades das ferramentas no Quadro 12. Pode-se destacar como maior diferencial entre o *OO Programming Lab* e as demais ferramentas, o fato de o *OO Programming Lab* estar disponível para o ambiente mobile. Outra característica que podemos destacar é o fato de a ferramenta trabalhar com C# ao invés de Java, como as demais.

Quadro 12 – Quadro comparativo

Funcionalidade/Programa	**BlueJ**	**Jeliot**	**Metodologia e Ferramenta para Ensino da Orientação a Objetos**	**OO Programming Lab**
Principal funcionalidade	Demonstrar as classes e objetos de forma concreta	Fazer uma animação do código fonte	Auxiliar o professor a acompanhar a resolução de exercícios	Demonstrar as classes e objetos de forma concreta
Benefícios	Interatividade, visualização e simplicidade	Não requer anotações ou modificações no código para fazer a animação	Permite melhor acompanhamento do aprendizado do aluno	Interatividade, simplicidade e estar disponível em plataformas móveis
Tecnologias usadas	Java	Java	Java, HTML	C#, WCF
Ambiente	*Desktop*	*Desktop*	*Web*	*Mobile*
Permite criar e visualizar todas as classes criadas	Sim	Não	Não	Sim
Permite criar e visualizar todos os objetos criados	Sim	Não	Não	Sim
Permite visualizar o estado do objeto e fazer chamada de seus métodos	Sim	Não	Não	Sim
Apresenta as classes em formato semelhante a um diagrama de classes	Sim	Não	Não	Não

4 CONCLUSÕES

No decorrer deste trabalho, foram alcançados os objetivos de desenvolver uma ferramenta capaz de auxiliar os usuários no aprendizado dos conceitos de Orientação a Objeto. Essa ferramenta, batizada de *OO Programming Lab*, permite a criação de classes com seus atributos e métodos além de permitir que os métodos sejam chamados passando seus parâmetros.

Durante o desenvolvimento, quando foi trabalhado com o WCF no lado do servidor, teria sido melhor utilizar os métodos recebendo apenas parâmetros do tipo *string*. Recebendo parâmetros de tipos complexos fez com que o WCF gerenciasse a serialização, o que dificultou muita a utilização desses serviços, pois era necessário fazer com que o cliente serializasse exatamente como o WCF esperava que os parâmetros fossem passados. Se tivesse sido programado para o WCF receber apenas parâmetros *string*, seria possível ter total controle sobre a serialização, tanto do lado do cliente, quando do lado do servidor.

Com a ferramenta Xamarin, acreditava-se que poderia ser facilmente convertida a aplicação de Windows Phone para Android, o que não se provou verdade. A interface de usuário teria que ser totalmente reescrita para o Android. Dessa forma a intenção de portar o aplicativo para Android foi deixada como sugestão para trabalhos futuros.

Por meio da técnica de TDD foi possível manter o código com uma boa coesão e desacoplado. Tendo isso em mente, é possível afirmar que o TDD cumpriu a sua premissa de melhorar o ciclo de desenvolvimento de uma ferramenta e é recomendável a sua utilização. O *framework* de teste da Microsoft, no entanto, era muito limitado na questão de possibilidades de comparação entre o valor esperado e o obtido. Por tal motivo talvez teria sido melhor utilizar um *framework* de terceiros, como o NUnit, por exemplo, que é um *framework* de testes unitários de código aberto.

Optou-se por fazer essa ferramenta para Windows Phone por ser uma plataforma não tão explorada quanto o Android e o iOS, e, além disso, estar crescendo em participação no mercado. As tecnologias utilizadas para o desenvolvimento para Windows Phone, como o Windows Phone SDK, e o simulador são muito boas, apesar das funcionalidades no nível de programação do Windows Phone serem mais limitadas em comparação com as funcionalidades disponíveis para programação para *desktop*. Um exemplo disso é o fato de o Windows Phone não dar suporte direto a WCF. A programação para Windows Phone é razoavelmente recente, em comparação com Android e iOS, e tem crescido muito. Por isso muitas funcionalidades estão sendo constantemente criadas, e substituídas, o que dificultou

um pouco nos momentos de pesquisa. Pois, em certo número de ocasiões, encontrava-se material referente a funções que já não existiam mais, ou não funcionavam mais daquela forma. Mas é uma plataforma muito boa, que está crescendo e se solidificando no mercado, tanto entre consumidores, quanto programadores.

4.1 EXTENSÕES

Como sugestões para trabalhos futuros pode-se relacionar:

a) portar a ferramenta para Android e para iOS;
b) concluir o cadastro de salas de aula, professores e alunos;
c) representar as classes em formato que lembre um diagrama de classes;
d) permitir que sejam passados parâmetros de tipos complexos na chamada dos métodos, ao invés de apenas tipos simples;
e) adicionar funcionalidades na ferramenta para permitir utilização de herança e porlimorfimos.

REFERÊNCIAS

BECK, Kent; CUNNINGHAM, Ward. **A laboratory for teaching object-oriented thinking**. New Orleans, 1989. Disponível em: <http://c2.com/doc/oopsla89/paper.html>. Acesso em: 23 jan. 2015.

BECK, Kent. **Test driven development**: by example. Indianapolis: Addison-Wesley, 2002.

BEN-ARI, Mordechai; LEVY, Ronit; URONEN, Pekka. **An extended experiment with Jeliot 2000**. Israel, 2000. Disponível em: <http://stwww.weizmann.ac.il/g-cs/articles/jel.pdf>. Acesso em: 12 mar. 2015.

BURNS, Kyle. **Beginning Windows 8 application development**: XAML edition. New York: Springer Science, 2012.

CAMINHA, Kaléu. **Diretrizes para o ensino de orientação a objetos com Java.** Florianópolis, 2012a. Disponível em: <http://www.slideshare.net/kaleu/diretrizes-para-o-ensino-de-orientao-a-objetos-com-java>. Acesso em: 22 jan. 2015.

CAMINHA, Kaléu. **Ensino de programação orientada a objetos com jogos 2D**. Florianópolis, 2012b. Disponível em: <https://kaleu.files.wordpress.com/2012/10/relatc3b3rio-1-metodologia-ensino-poo-baseada-em-jogos.pdf>. Acesso em: 22 jan. 2015.

CARDOSO, Carlos H. R.. **PraticOO**: método para o aprendizado de orientação a objetos por estudantes de engenharia. Santa Rita do Sapucaí, 2002. Disponível em: <http://www.inatel.br/biblioteca/component/ docman/doc_download/3668-praticoo-metodo-para-o-aprendizado-de-orientacao-a-objetos-por-estudantes-de-engenharia>. Acesso em: 22 jan. 2015.

CARR, Richard. **Invoking methods using reflection**. Yorkshire, 2012a. Disponível em: <http://www.blackwasp.co.uk/ReflectionInvokeMethods.aspx>. Acesso em: 08 jun. 2015.

CARR, Richard. **Reflecting method information**. Yorkshire, 2012b. Disponível em: <http://www.blackwasp.co.uk/ReflectingMethods.aspx>. Acesso em: 08 jun. 2015.

CLARK, Dan. **Introdução à programação orientada a objetos com Visual Basic .Net**. Rio de Janeiro: Editora Ciência Moderna, 2003.

COPLIEN, James O.. **Teaching OO:** putting the object back into OOD. Illinois, 2003. Disponível em: <http://www.artima.com/weblogs/viewpost.jsp?thread=6771>. Acesso em: 22 jan. 2015.

C-SHARP TUTORIALS. **Reflection introduction**: The complete C# tutorial. 2015. Disponível em: <http://csharp.net-tutorials.com/reflection/the-right-type/>. Acesso em: 20 fev. 2015.

FORMAN, Ira; FORMAN, Nate. **Java reflection in action**. Greenwich: Manning Publications Co., 2005.

FREEMAN, Adam; **Metro revealed:** Building Windows 8 apps with XAML and C#. Nova Iorque: Springer Science, 2012.

FREEMAN, Steve; PRYCE, Nat. **Growing object-oriented software, guided by tests**. Boston: Addison-Wesley, 2010.

GALHARDO, Mariane F. **Metodologia e ferramenta para ensino da programação orientada a objetos.** Sorocaba, 2004. Disponível em: <https://www.academia.edu/2977606/METODOLOGIA_E_FERRAMENTA_PARA_ENSINO_DA_PROGRAMA%C3%87%C3%83O_ORIENTADA_A_OBJETOS>. Acesso em: 22 jan. 2015.

GALHARDO, Mariane F.; ZAINA, Luciana A. M.. **Simulação para ensino de conceitos da orientação a objetos.** Blumenau, 2004. Disponível em: <http://www.inf.furb.br/seminco/2004/artigos/110-vf.pdf>. Acesso em: 22 jan. 2015.

GUTTAG, John; LISKOV, Barbara. **Program development in java**: abstraction, specification, and object-oriented design. 9 Ed. Indianapolis: Addison-Wesley, 2006.

JELIOT 3. **User guide**. Finlândia, 2012. Disponível em: <http://cs.joensuu.fi/jeliot/files/userguide.pdf>. Acesso em: 22 jan. 2015.

KOLLING, Michael. **The problem of teaching object-oriented programming:** part 1: languages. Australia, 2011. Disponível em: <https://kar.kent.ac.uk/21879/2/the_problem_of_teaching_ object-oriented_kolling_1.pdf>. Acesso em: 22 jan. 2015.

LÖWY, Juval. **Programming WCF services**. 3 Ed. California: O'Reilly Media, 2010.

MCLAUGHLIN, Brett; POLLICE, Gary; WEST, David. **Use a cabeça**: análise e projeto orientado a objeto. Rio de Janeiro: Alta Books, 2007.

MENDES, Douglas R.. **Programação Java com ênfase em orientação a objetos**. São Paulo: Novatec, 2009.

MSDN. **Reflection (C# and Visual Basic).** 2013a. Disponível em: <https://msdn.microsoft.com/en-us/library/ms173183.aspx>. Acesso em: 20 fev. 2015.

MSDN. **Reflection in the .NET Framework.** 2013b. Disponível em: <https://msdn.microsoft.com/en-us/library/f7ykdhsy.aspx>. Acesso em: 20 fev. 2015.

MSDN. **What is Windows Communication Foundation.** 2015a. Disponível em: <https://msdn.microsoft.com/en-us/library/ms731082.aspx>. Acesso em: 26 mai. 2015.

MSDN. **What is XAML.** 2015b. Disponível em: <https://msdn.microsoft.com/en-us/library/cc295302.aspx>. Acesso em: 26 mai. 2015.

NAPS et al. **Exploring the role of visualization and engagement in computer science education**. Montana, 2002. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.15.2493&rep=rep1&type=pdf>. Acesso em: 13 mar. 2015.

PATHAK, Nishith. **Pro WCF 4**: practical Microsoft SOA implementation. 2 Ed. New York: Springer Science, 2011.

STATISTA. **Smartphone operating systems - global market share 2011-2015.** Nova Iorque, 2015. Disponível em: <http://www.statista.com/statistics/277048/global-market-share-forecast-of-smartphone-operating-systems/>. Acesso em: 18 fev. 2015.

WAZLAWICK, Raul S. **Análise e projeto de sistemas de informação orientados a objetos.** Rio de Janeiro: Elsevier Editora, 2004.

XAMARIN. **Getting started with Android.** San Francisco, 2015. Disponível em: < http://developer.xamarin.com/guides/android/getting_started/>. Acesso em: 14 jun. 2015.