Universidade Federal do Pampa
Douglas Montanha Giordano

UML Sketch Recognizer: Um aplicativo para reconhecimento de esboços de diagramas de classe em fotos

Alegrete
2015

Douglas Montanha Giordano

UML Sketch Recognizer: Um aplicativo para reconhecimento de esboços de diagramas de classe em fotos

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Engenharia de Software.

Orientador: João Pablo Silva da Silva

Alegrete

2015

Douglas Montanha Giordano

UML Sketch Recongnizer: Um Aplicativo para Extração de Esboços de Diagramas de Classe de Imagens

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Engenharia de Software.

Trabalho de Conclusão de Curso defendido e aprovado em 02 de dezembro de 2015

Banca examinadora:

João Pablo Silva da Silva
Orientador

Fábio Natanael Kepler
UNIPAMPA

Marcelo Resende Thielo
UNIPAMPA

Este trabalho é dedicado aos meus pais e irmãos pelo exemplo, amor, carinho e apoio.
Também dedico aos meus amigos que nos momentos felizes e tristes me apoiaram.

Agradecimentos

Difícil agradecer todas pessoas que me ajudaram ou me deram apoio durante este trabalho. Mas primeiramente agradeço aos meus pais Ana e Rui que me apoiaram sempre, me dando amor, carinho e também a oportunidade de crescer na vida. Também agradeço aos meus irmãos, Diego e Diogo, que me ensinaram que apesar das dificuldades e medo tudo aquilo com esforço é recompensado.

Agradeço ao meu orientador João Pablo por ter compartilhado seu conhecimento como pessoa e profissional. Pela paciência em mostrar-me o que é certo e errado, também por confiar em mim quando iniciamos um trabalho que tinha de certa forma um risco de ser realizado.

Agradeço aos meus amigos e colegas de faculdade, que me apoiaram durante todos semestres. Agradeço especificamente ao Miguel, Alex e Gean que aguentaram eu falando 24 horas por dia sobre o trabalho de conclusão.

Por fim agradeço a todos professores que me apoiaram nesta longa jornada.

Resumo

A modelagem ágil é uma estratégia de modelagem de software para o desenvolvimento ágil. Existem várias práticas para realizar uma modelagem ágil, como por exemplo a utilização de ferramentas simples. Os quadros brancos são utilizados por muitas equipes que utilizam práticas de modelagem ágil, pois oferecem um ambiente dinâmico e colaborativo de modelagem. O problema ocorre na geração dos artefatos de software. Os esboços feitos em quadros brancos são normalmente armazenados em fotos, ocasionando uma desatualização da documentação e também impossibilidade de gerar códigos equivalentes ao esboço. Com o objetivo de minimizar essa integração entre esboços feitos em quadros e editores UML, este trabalho apresenta um aplicativo de reconhecimento de esboços de diagramas de classe para Android. A aplicação utiliza técnicas de processamento e análise de imagens juntamente com a biblioteca de visão computacional Open CV e o interpretador de caracteres Tesseract. A interpretação da imagem é feita por uma camada de reconhecimento que processa, segmenta, classifica e interpreta a imagem contendo o diagrama. A partir de testes feitos com vários esboços de diagramas, foi possível perceber um alto nível de detecção dos elementos em ambiente controlado, que ficou em torno de 70% e 100%. Já o reconhecimento teve um médio nível de precisão, que ficou em torno de 30% e 90%.

Palavras-chave: UML, reconhecimento de esboço, processamento de imagens, CASE, modelagem ágil, diagrama de classes, modelo conceitual.

Abstract

The Agile Modeling is a software modeling strategy for agile development. There are several practical to perform a responsive modeling, such as the use of simple tools. The whiteboards are used by many teams that use agile modeling practices, they offer a dynamic and collaborative environment modeling. The problem occurs in the generation of software artifacts. The sketches on whiteboards are usually stored in photos, causing a downgrade of the documentation and also inability to generate the outline equivalent codes. In order to minimize this integration between sketches in frames and UML editors, this paper presents a recognition application class diagrams of sketches for Android. The application uses processing techniques and image analysis with computer vision library Open CV and interpreter Tesseract characters. The interpretation of the image is made by a recognition layer processing, segments, classifies and interprets the image containing the diagram. From tests with various sketches of diagrams, it was possible to realize a high level of detection of the elements in a controlled environment, which was around 70% and 100%. Already recognition had an average level of precision that was around 30% and 90%.

keyword:.UML, sketch recognition, image processing, CASE, Agile modeling.

Lista de ilustrações

Lista de tabelas

Lista de siglas

API Application Programming Interface

CASE Computer-Aided Software Engineering

CNN Rede Neural Convolucional

EM Espectro Eletromagnético

GPU Graphics Processing Unit

MVC Model-View-Controller

NDK Native Development Kit

OCR Optical Character Recognition

OHA Open Handset Alliance

OMG Object Management Group

OMT Object Modeling Thecnique

OOSE Object-Oriented Software Engineering

Open CV Open Source Computer Vision Library

SDK Software Development Kit

SVM Support Vector Machine

TDNN Time Delay Neural Network

UML Unified Modeling Language

XMI XML Metadata Interchange

XML Extensible Markup Language

Sumário

1 Introdução

A aplicação de métodos ágeis no decorrer do desenvolvimento de um software é adotado por muitas empresas pois proporciona mais flexibilidade em relação aos métodos tradicionais. Normalmente os métodos ágeis são interativos, evolutivos, adaptativos e incrementais, possuindo uma resposta rápida e flexível a mudanças (LARMAN, 2005).

A modelagem ágil nada mais é que a aplicação padrão da UML seguindo algumas práticas. Frequentemente a utilização da modelagem ágil oferece um alto investimento de tempo, pois enfatiza a utilização da UML como um rascunho diminuindo o tempo de modelagem (LARMAN, 2005). Outra prática comum na modelagem ágil é a criação de diagramas em equipe para resolver problemas do software.

Equipes de desenvolvimento que seguem práticas ágeis de modelagem, frequentemente tem preferência na utilização de quadros brancos ou ferramentas semelhantes. Os quadros brancos oferecem uma simplicidade no momento da modelagem, que se dá pelo fato de ser possível construir um diagrama de forma simples, muitas vezes utilizando apenas um marcador. Além disso os quadros brancos não impõem regras (COSTAGLIOLA; ROSA; FUCCELLA, 2014) na hora de desenhar um modelo, possibilitando o trabalho de vários membros em um único diagrama (HAMMOND; DAVIS, 2006). Como a modelagem ágil enfatiza a utilização de ferramentas simples e que possibilitem a comunicação entre os membros da equipe, quadros brancos ou ferramentas semelhantes são muito utilizadas.

1.1 Motivação

As ferramentas de modelagem de software atualmente não oferecem suporte total a modelagem ágil com UML pois raramente permitem uma modelagem colaborativa e também possuem muitas restrições, que limitam a capacidade de rascunhar diagramas. Visto que uma das principais práticas de modelagem ágil é utilizar ferramentas simples de forma colaborativa, fica inviável a utilização de uma ferramenta Computer-Aided Software Engineering (CASE) que não ofereça essas características de trabalho.

Apesar da facilidade de uso e simplicidade, a modelagem em quadros brancos possui alguns problemas. Os principais problemas são o ato corriqueiro de documentar esboços de diagramas em imagens e a decorrente dificuldade na integração dos artefatos de software (WÜEST; SEYFF; GLINZ, 2013).

A documentação do software por equipes que utilizam quadros brancos é na maioria das vezes feita por meio de imagens. A empresa poderá possuir um repositório local ou online que guarda essas imagens, porém muitas vezes as imagens são deixadas de lado

por não estarem atualizadas. Isso consequentemente leva a uma documentação sem valor para a equipe de desenvolvimento.

Outro problema também muito frequente é a integração dos artefatos de software. Quando um diagrama é feito em um quadro branco, não é possível gerar a codificação do mesmo. Levando muitas vezes a reconstrução do diagrama em uma ferramenta CASE. O processo de modelagem em quadros brancos e a subsequente reconstrução do diagrama gera retrabalho para a equipe, que por sua vez poderia estar trabalhando em outra atividade.

Os problemas de atualização de documentação e retrabalho podem ser minimizados através de uma ferramenta que possibilite a integração do esboço com um editor UML. É importante que os desenvolvedores tenham a integração entre as duas ferramentas pois o quadro branco é dinâmico e colaborativo e o editor UML possibilita a atualização da documentação e também várias outras funções como geração do código fonte.

1.2 Objetivo

Nosso trabalho tem como objetivo desenvolver uma ferramenta para extrair esboços de diagramas de classe. Essa ferramenta vai possibilitar a integração entre esboços feitos em quadros brancos e editores UML minimizando os problemas relatados na motivação. Os objetivos específicos são listados abaixo.

- viabilizar o reconhecimento de diagramas de classe em uma perspectiva conceitual;
- possibilitar a utilização do dispositivo móvel para capturar e reconhecer a imagem;
- gerar arquivos no formato XMI dos diagramas reconhecidos.

1.3 Metodologia

Para iniciar nosso trabalho, primeiramente investigamos o estado da arte das ferramentas de reconhecimento de esboço. Pesquisamos todos os trabalhos relacionados com nosso contexto através de um protocolo de mapeamento, que nos ajudou a organizar tanto a pesquisa como os resultados.

Também estudamos todos os conceitos e técnicas relacionadas com reconhecimento de imagens. Também estudamos os conceitos relacionados a modelagem de software.

Elaboramos uma visão da solução com o conhecimento adquirido. Com a solução pronta estabelecemos uma arquitetura, separando as responsabilidades de cada parte do software em camadas. Projetamos cada uma dessas camadas visando desacoplar as funções do sistema.

Construímos cada funcionalidade do software através de incrementos. No total foram 3 incrementos: detecção e reconhecimento de classes (incremento 1), associações (incremento 2) e multiplicidades (incremento 3). Testamos cada incremento com o objetivo de descobrir o nível de precisão e erro da ferramenta. Realizamos também em uma ambiente controlado alguns experimentos com possíveis usuários a fim de analisar os benefícios e problemas da ferramenta no ambiente de desenvolvimento. Coletamos durante nosso trabalho vários artefatos a fim de registrar no trabalho de conclusão do curso.

1.4 Contribuições

Em nosso trabalho criamos uma ferramenta que contempla o reconhecimento de classes, associações e também multiplicidade. Além de contribuições em termos de funcionalidade aos desenvolvedores que utilizarem o software, apresentamos contribuições a quem for desenvolver softwares de reconhecimento de esboço.

O software UML Sketch Recognizer que criamos deverá contribuir para minimizar o retrabalho de equipes que utilizam quadros brancos como ferramenta de modelagem. Toda modelagem conceitual do sistema que envolva apenas associações, classes e multiplicidades o nosso aplicativo pode reconhecer e gerar um arquivo para o desenvolvedor importar em um editor UML, possibilitando a geração de código ou qualquer outra função desejada.

Para os desenvolvedores de sistemas de processamento de análise de imagens, contribuímos com os métodos selecionados e também módulos utilizados para resolver o problema. Apesar de complexa a aplicação, foram utilizados métodos simples para resolver grande parte dos problemas. Nosso trabalho oferece uma direção para quem está começando a construir aplicativos semelhantes, além já apresentar alguns métodos que podem ser utilizados.

1.5 Organização do Documento

O presente documento está organizado da seguinte forma:

- **Capítulo 2:** abordamos os conceitos relacionados com nosso trabalho na perspectiva de alguns autores.
- **Capítulo 3:** apresentamos as tecnologias que ajudaram na construção da ferramenta.
- **Capítulo 4:** apresentamos o estado da arte das pesquisas relacionadas com nosso objetivo.

- **Capítulo 5:** apresentamos alguns dos artefatos gerados durante o desenvolvimento e também de forma detalhada algumas partes da implementação.
- **Capítulo 6:** apresentamos a verificação e validação do protótipo.
- **Capítulo 7:** apresentamos as conclusões e os potenciais trabalhos futuros.

2 Fundamentação Teórica

Neste capítulo apresentamos os conceitos ligados ao nosso trabalho. Abordamos conceitos relacionados com os sistemas de reconhecimento de esboços e modelagem de software. Na seção 2.1 apresentamos a UML, uma linguagem de modelagem de software. Na seção 2.2 apresentamos o processamento de imagens, responsável pela melhoria ou retirada de características indesejadas em imagens. Na seção 2.3 apresentamos a análise de imagens, que visa a extração e compreensão do conteúdo de uma imagem. Por fim na seção 2.4 apresentamos algumas lições do capítulo.

2.1 Linguagem Unificada de Modelagem

A UML é uma linguagem de modelagem de software padrão que pode ser empregada na visualização, especificação, construção e documentação de artefatos de software. Foi criada por Grady Booch, James Rumbaugh e Ivar Jacobson com objetivo de padronizar a forma de modelar software (BOOCH; RUMBAUGH; JACOBSON, 2000).

A UML foi criada com objetivo de oferecer uma única linguagem de modelagem após o aparecimento de várias linguagens orientadas a objetos (OO). Segundo Booch, Rumbaugh e Jacobson (2000) durante o período de 1989 e 1994 haviam em torno de 50 métodos para análise e projeto OO. Os usuários dos métodos de análise e projeto OO tinham dificuldade de encontrar um método que atendesse a todas suas necessidades.

Depois dessa experiência novos métodos foram criados, alguns com maior destaque como Booch, o Object-Oriented Software Engineering (OOSE) de Jacobson e Object Modeling Thecnique (OMT) de Rumbaugh. Cada um desses métodos possuía pontos forte e pontos fracos. Como os métodos dos três autores estavam indo em uma direção parecida, os mesmos foram motivados a criar uma linguagem unificada.

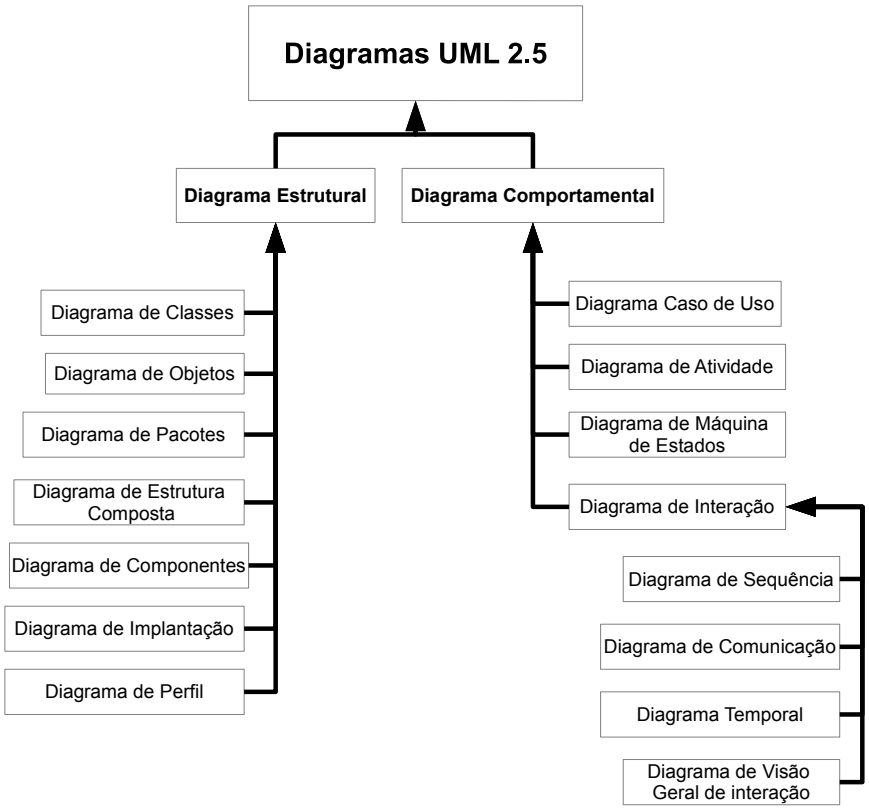
Em 1994 iniciaram os esforços para a criação da UML. Em 1995 foi lançado a versão 0.8 da UML que na época era chamada de método unificado. A documentação da versão 0.9 foi lançada em 1996 após Jacobson se associar a Rational e iniciar a incorporação do método OOSE. Em 1997 a versão 1.1 da UML foi adotada pela Object Management Group (OMG). Atualmente a UML está na versão 2.4.1 e a versão 2.5 está em fase beta.

A UML possui três tipos de blocos de construção denominados de itens, relacionamentos e diagramas. Os itens podem ser divididos em estruturais, comportamentais, de agrupamento e notacionais. Os diagramas da UML 2.5 podem ser divididos em duas categorias denominadas de diagramas estruturais e diagramas comportamentais mostrados

na Figura 1 (OMG, 2015).

Cada um dos diagramas podem ter modelos derivados de sua estrutura. Um exemplo de diagrama que possui alguns modelos derivados é o diagrama de classes, que podemos citar como derivado o modelo de domínio. Na construção de um modelo de domínio não é levado em conta questões tecnológicas e sim apenas elementos do negócio que a aplicação faz parte.

Figura 1: Diagramas da UML 2.5 (OMG, 2015).



Nas próximas subseções vamos mostrar os diagramas que compõem cada uma das categorias.

2.1.0.1 Diagramas Estruturais

Os diagramas estruturais tem como objetivo mostrar o sistema de uma perspectiva estática. Abaixo podemos ver uma pequena descrição de cada um deles.

Diagrama de classes: mostra classes, interfaces, colaborações e seus relacionamentos. As classes são representações de um conjunto de objetos. As interfaces são um conjunto de operações que são utilizadas para especificar os serviços que uma classe ou componente vai oferecer. Já as colaborações são um conjunto de classes, interfaces

ou qualquer outro elemento onde todos eles estão relacionados (BOOCH; RUMBAUGH; JACOBSON, 2000).

Diagrama de objetos: mostra as instâncias de cada classe, os valores dos atributos dos objetos e seus relacionamentos (OMG, 2015).

Diagrama de pacotes: além de mostrar os pacotes do sistema, também pode ser utilizado para mostrar alguma abstração ou visão específica de um sistema, para descrever aspectos arquitetônicos, lógicas ou comportamentos do sistema (OMG, 2015).

Diagrama de estrutura composta: mostra a estrutura interna de uma classe ou colaboração (BOOCH; RUMBAUGH; JACOBSON, 2000).

Diagrama de componentes: mostra as partes internas, os conectores e as portas que implementam um componente (BOOCH; RUMBAUGH; JACOBSON, 2000).

Diagrama de implantação: mostra um conjunto de nós relacionados sendo utilizado para visualizar a implantação de uma arquitetura em uma perspectiva estática (BOOCH; RUMBAUGH; JACOBSON, 2000). Também pode ser utilizado para visualizar a arquitetura lógica e física de uma rede (OMG, 2015).

Diagrama de perfil: um perfil possibilita a modificação da UML para diferentes plataformas ou domínios (OMG, 2015), possibilitando criar uma extensão leve da UML.

2.1.0.2 Diagramas Comportamentais

Os diagramas comportamentais tem como objetivo mostrar o sistema em uma perspectiva mais dinâmica. Abaixo podemos ver uma pequena descrição de cada um deles.

Diagrama de caso de uso: mostra um conjunto de ações feitas por atores em um sistema. As ações podem ser feitas em colaboração com um ou mais usuários externos do sistema (OMG, 2015).

Diagrama de atividade: mostra o fluxo de atividades do sistema onde cada atividade pode ter seu fluxo alterado por uma condição. Em um diagrama de atividade também pode ser apresentada as ações ou modificações de um objeto durante o fluxo de atividades (BOOCH; RUMBAUGH; JACOBSON, 2000).

Diagrama de máquina de estado: mostra um conjunto de estado, suas transições, eventos e atividades. São importantes para modelagem do comportamento de uma interface, classe ou colaboração (BOOCH; RUMBAUGH; JACOBSON, 2000).

Diagrama de sequência: mostra uma sequência de mensagens enviadas ou recebidas entre um conjunto de instâncias. Esse diagrama tem ênfase na organização temporal das mensagens (BOOCH; RUMBAUGH; JACOBSON, 2000).

Diagrama de comunicação: mostra um conjunto de instâncias e suas conexões dando ênfase à organização estrutural dos objetos (BOOCH; RUMBAUGH; JACOBSON, 2000).

Diagrama temporal: mostra interações quando um objetivo principal do diagrama trabalha em cima de um determinado tempo. Diagramas temporais se concentram em condições de mudança dentro e entre as linhas de vida de um objeto(OMG, 2015).

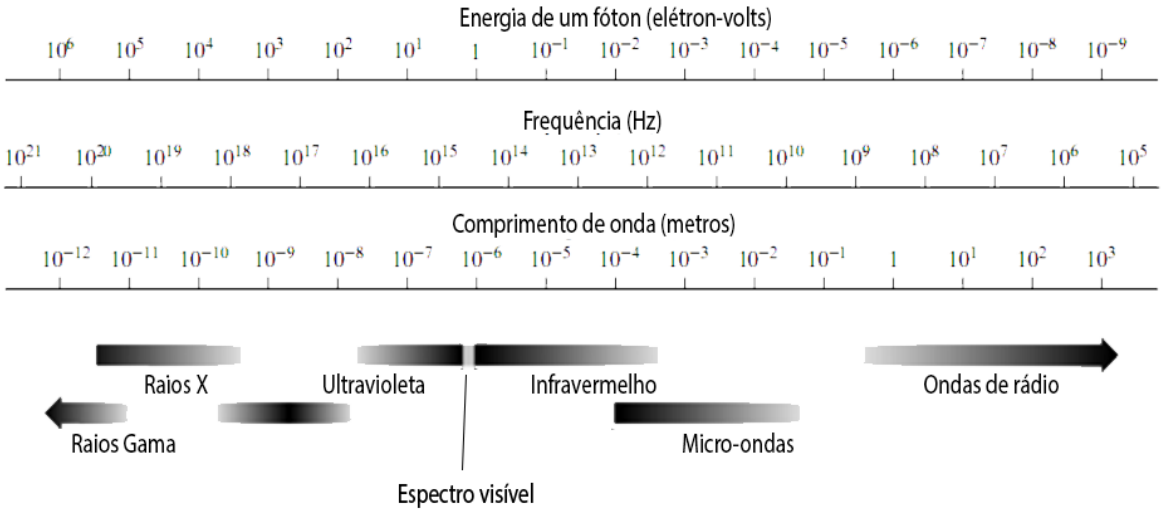
Diagrama de visão geral de interação é uma variante do diagrama de atividades, apresentando uma visão geral do fluxo de controle (OMG, 2015).

2.2 Processamento de Imagens Digitais

O processamento de imagens consiste em um conjunto de técnicas para capturar, representar e transformar imagens com auxílio de computador (PEDRINI; SCHWARTZ, 2008). O interesse nos métodos de processamento digital de imagens vem de duas áreas principais de aplicação: melhora das informações visuais para interpretação humana e processamento de dados de imagens para armazenamento, transmissão e representação, considerado a percepção automática por máquinas (GONZALEZ; WOODS, 2010).

A visão do ser humano é bastante limitada à banda visual do Espectro Eletromagnético (EM) mostrada na Figura 2. Os aparelhos de processamento de imagens cobrem quase todo o espectro EM, possibilitando gerar imagens digitais de fontes incomuns ao olho humano como ultrassom, microscopia eletrônica e imagens geradas por computador (GONZALEZ; WOODS, 2010).

Figura 2: Espectro eletromagnético (GONZALEZ; WOODS, 2010).



Basicamente podemos definir uma imagem digital como uma função bidimensional onde em cada par de coordenadas (x,y) possui um valor representando a densidade,

também chamado nível de cinza. Esse par de coordenada possui vários nomes porém o mais utilizado é Pixel.

As fronteiras entre processamento de imagens, análise de imagens e visão computacional não são totalmente definidas e não há um acordo geral sobre isso. Para tentar dividir ou criar uma fronteira artificial é possível levar em consideração três tipos de processos computacionais: baixo, médio e alto. Processos de nível baixo entram em várias operações onde a entrada é uma imagem e a saída também. Os processos de nível médio levam em conta a extração de características das imagens. Já as de nível alto tentam dar sentido a imagem. O processamento de imagens está em um nível baixo e a análise estaria em um nível médio e alto. A visão computacional está localizada no nível alto também, juntamente com a análise de imagens (GONZALEZ; WOODS, 2010).

Os componentes básicos de um sistema de processamento de imagens de uso geral apresentado por GONZALEZ e WOODS (2010) podem ser dividido conforme mostrado na Figura 3. Os monitores são utilizados na apresentação da imagem. O computador pode ser de uso geral, onde em alguns casos é necessário um computador especializado para aumento de desempenho. O armazenamento em massa é feito por um dispositivo de armazenamento. Esse dispositivo é indispensável para sistemas de processamento de imagens que trabalham com uma grande quantidade de imagens.

O sistema de registro é um dispositivo utilizado para impressão ou apresentação da imagem. O hardware especializado em processamento de imagens normalmente consiste em um digitalizador (conversor de estímulos elétricos em dados digitais) e um hardware para realizar operações que requerem um rápido processamento. O Software de processamento de imagens é um conjunto de módulos especializados para a realização de tarefas especificas. Os sensores são dispositivos responsáveis pela aquisição da imagem. Por último temos a rede de comunicação que faz a transferência de dados entre os componentes.

Segundo GONZALEZ e WOODS (2010) alguns passos são fundamentais no processamento digital de imagens. Na Figura 4 apresentamos alguns métodos utilizados no processamento de imagens que são aplicados com diferentes propósitos e objetivos. Podemos considerar como métodos de processamento de imagem a aquisição da imagem, realce de imagens, restauração de imagens, processamento de imagens coloridas e a compreensão de imagens. Os métodos de processamento morfológico, segmentação, representação e descrição e reconhecimento de objetos (classificação de padrões) são métodos de análise de imagens que apresentamos na seção 2.3. Nas próximas seções vamos apresentar um pouco sobre algumas técnicas consideradas fundamentais e que utilizamos nesse trabalho.

Figura 3: Componentes de um software de processamento de imagens de uso geral (GONZALEZ; WOODS, 2010).

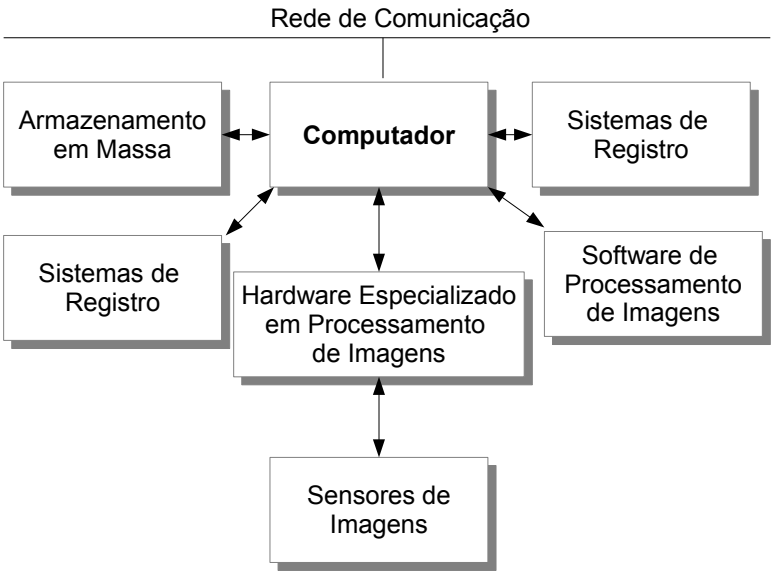
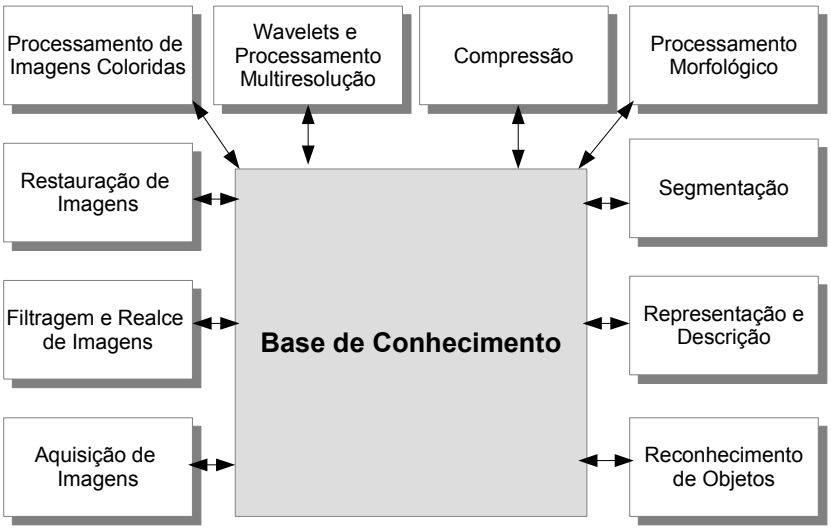


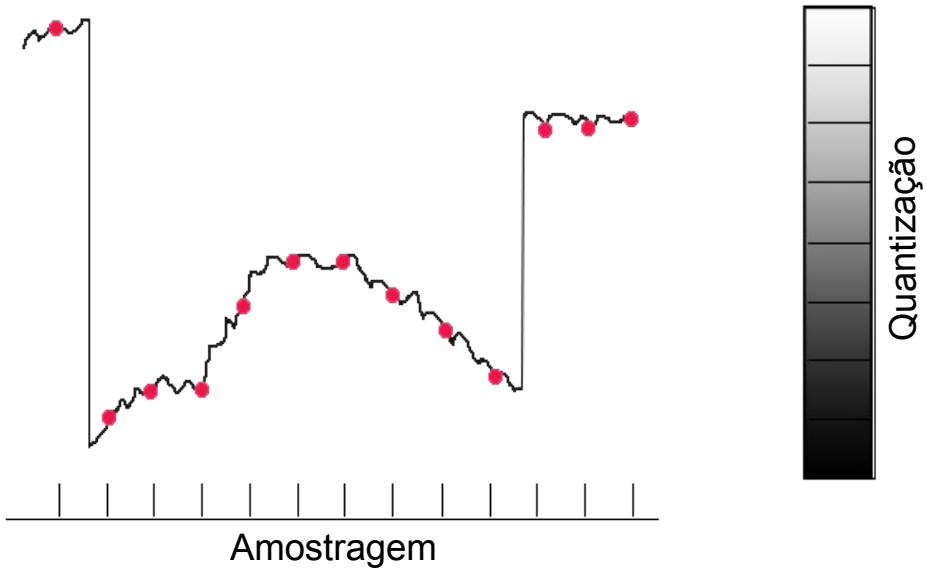
Figura 4: Passos fundamentais para o processamento de imagens.



2.2.1 Aquisição de imagens

A aquisição de uma imagem é basicamente feita através de um processo chamado de amostragem e quantização. A Figura 5 exemplifica melhor o processo, onde os pontos vermelhos são as amostras escolhidas e o quadrado preenchido no lado direito é a intensidade que cada amostra vai ter como valor (GONZALEZ; WOODS, 2010).

Figura 5: Amostragem e quantização (adaptada de GONZALEZ e WOODS (2010)).



2.2.2 Filtragem e Realce de Imagens

O realce da imagem é um processo que tem como objetivo melhorar as características da imagem, onde sua aplicação depende totalmente do domínio do problema. As técnicas de realce de imagens possuem um observador para decidirem se o resultado foi o esperado (GONZALEZ; WOODS, 2010). O realce é necessário quando a imagem possui alguma degradação, perda de qualidade, perda de contraste, borramento, distorção, ou condição inadequada de iluminação (PEDRINI; SCHWARTZ, 2008).

2.3 Análise de Imagens Digitais

A análise de imagens é, tipicamente, baseada na forma, textura, níveis de cinza ou nas cores dos objetos presentes em uma imagem. A análise de imagens tem como foco compreender o conteúdo da imagem. Algumas técnicas comuns na análise da imagem são a segmentação da imagem em regiões ou objetos de interesse, descrição dos objetos e reconhecimento ou classificação dos mesmos (PEDRINI; SCHWARTZ, 2008).

Nas próximas seções vamos apresentar etapas que muitas vezes um software de análise de imagem possui para a compreensão da imagem.

2.3.1 Segmentação de Imagens

A segmentação é um processo importante para identificação de objetos. A imagem é subdividida em regiões ou objetos de interesse. A segmentação é uma das tarefas

mais difíceis em um sistema de análise ou processamento de imagens onde a precisão da segmentação determina o sucesso ou o fracasso final de um procedimento (GONZALEZ; WOODS, 2010). A divisão dos objetos e regiões também depende muito do processamento da imagem onde por exemplo imagens com muito ruído podem distorcer a extração dos dados da imagem (PEDRINI; SCHWARTZ, 2008). As técnicas de segmentação buscam detectar descontinuidades e similaridades da imagem. Alguns exemplos de abordagens são a detecção de retas, detecção de bordas, segmentação de regiões e limiarização.

2.3.2 Representação e Descrição

As regiões e objetos gerados pela etapa de segmentação necessitam ser representados e descritos para os próximos passos da análise da imagem. O objeto pode ser representado em termos de suas características externas (bordas) e características internas (pixeis). A descrição depende da representação escolhida e é necessário a extração de características e medidas mínimas não permitindo ambiguidades. Exemplos de representação são código de cadeia e seguidor de fronteira. Um exemplo de descritor é o comprimento de uma fronteira.

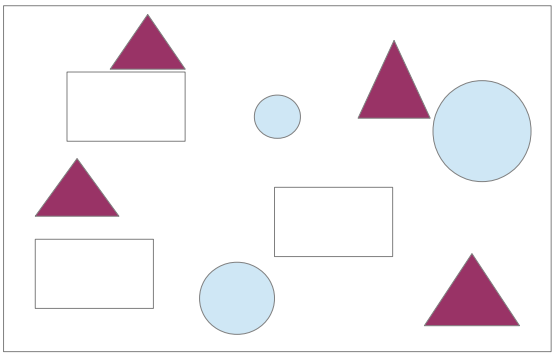
2.3.3 Morfologia Matemática

A morfologia matemática é uma ferramenta para extrair componentes das imagens que são úteis na representação e na descrição da forma de uma região (GONZALEZ; WOODS, 2010). Pode ser utilizada em extração de componentes conexos, busca de padrões específicos na imagem, delimitação do fecho convexo, extração de bordas dos objetos, afinamento de bordas e muitas outras aplicações (PEDRINI; SCHWARTZ, 2008). As formas dos objetos são representadas utilizando teoria dos conjuntos. Com a utilização da morfologia matemática podem ser feitas operações como erosão e dilatação de objetos e abertura e fechamento de objetos além de muitas outras operações.

2.3.4 Classificação de Padrões

A classificação de padrões ou também chamada de reconhecimento de objetos visa mapear amostras extraídas com um conjunto de rótulos possuindo algumas restrições para mapear amostras com características semelhantes (PEDRINI; SCHWARTZ, 2008). Por exemplo a Figura 6 seria classificada contendo triângulos, círculos e quadrados. Apesar de ter círculos de vários tamanhos, os mesmos possuem características semelhantes. Um exemplo de abordagem que pode ser utilizada para classificação de padrões são redes neurais.

Figura 6: Figuras geométricas que devem ser classificadas com relação a sua forma.



2.4 Lições do Capítulo

Durante o capítulo apresentamos conceitos relacionados ao tratamento e interpretação de imagens e também a UML. Os conceitos ajudaram a entender melhor quais são as possibilidades de construir uma solução adequada, e são extremamente importantes para a construção de aplicações relacionadas a modelagem e ao reconhecimento de imagens.

A UML como uma linguagem unificada possui uma grande gama de diagramas. Em sua versão atual temos 14 diagramas diferentes para serem utilizados na modelagem de um software. Cada diagrama pode ser utilizado de várias formas para modelar certa perspectiva do sistema.

O processamento e análise de imagens, apesar de possuir fronteiras que as diferenciam, trabalham juntas para resolver problemas de reconhecimento de imagens. O processamento de imagens especificamente trabalha para melhorar as características de imagens, que em nosso contexto é muito variável e vai depender do ambiente que o desenvolvedor captura a imagem. E a análise de imagens, que oferece recursos para extrair e reconhecer áreas de interesse em uma imagem, que em nosso trabalho são os elementos de um diagrama de classes.

3 Fundamentação Tecnológica

Neste capítulo apresentamos todas tecnologias que ajudaram na construção da ferramenta. Primeiramente na seção 3.1 apresentamos a plataforma que o aplicativo foi construído. Na seção 3.2 apresentamos a biblioteca de visão computacional Open CV, que auxiliou na utilização de técnicas de processamento e análise de imagens. Na seção 3.3 apresentamos o Optical Character Recognition (OCR) Tesseract, que oferece recursos para o reconhecimento de caracteres. Por fim na seção 3.4 apresentamos algumas lições do capítulo.

3.1 Plataforma Android

O Android é uma plataforma de tecnologia *open source* para dispositivos móveis. A plataforma Android foi construída com base no sistema operacional Linux (PEREIRA; SILVA, 2009).

Um grupo denominado de Open Handset Alliance (OHA) foi criado com a intensão de padronizar a plataforma de código aberto e livre para celular. Nesse grupo existem empresas como Motorola, LG, Samsung e muitas outras (LECHETA, 2013). Todas essas empresas são lideradas pela Google na padronização da plataforma Android.

Os aplicativos da plataforma Android são construídos na linguagem Java e também podem ser construídos na linguagem C ou C++. Os aplicativos na linguagem de programação Java utilizam o pacote de desenvolvimento denominado de *Software Development Kit (SDK)*. Já os aplicativos construídos em C ou C++ utilizam o pacote de desenvolvimento denominado de *Native Development Kit (NDK)*. O NDK é utilizado por aplicações que necessitem de um poder de processamento maior do que aplicações normais (ANDROID, 2015a).

Nas próximas seções vamos apresentar algumas informações bases para o desenvolvimento de aplicações Android.

3.1.1 Alguns Recursos Básicos

Cada versão do Android é seguida de vários recursos que fazem parte da construção de aplicativos. Nesta seção vamos nos concentrar na apresentação de alguns dos recursos básicos para aplicativos Android.

As *activities* são classes que fazem o gerenciamento da tela de visualização do usuário. Cada *Activity* controla a criação de uma tela visualizada pelo usuário, podendo

retirar ou colocar componentes além de muitas outras funções. A classe *Activity* pode ser comparada com o controlador do padrão *Model-View-Controller (MVC)* (PEREIRA; SILVA, 2009).

A classe *View* é responsável por desenhar toda tela que o usuário visualiza em uma aplicação Android (PEREIRA; SILVA, 2009). A *View* é uma base para criação de botões, textos, entradas de texto e outros componentes que são visualizados por um usuário (ANDROID, 2015b).

O Android oferece um vocabulário *Extensible Markup Language (XML)* que possibilita criar as telas de uma aplicação em XML. Por meio da estrutura XML é possível adicionar botões, campos de entradas, calendários etc. Esses componentes podem ser adicionados diretamente no código Java. Na maioria das vezes é mais viável deixar apenas o controle das telas no código java, e os componentes diretamente no arquivo XML.

3.2 Open CV Android

Open Source Computer Vision Library (Open CV) é um biblioteca *open source* para criação de aplicativos de visão computacional e também aprendizado de máquina. A biblioteca Open CV tem mais de 2500 algoritmos de visão computacional e aprendizado de máquina.

Com a biblioteca é possível fazer operações como detecção de rostos até filtros simples em uma imagem.

A biblioteca Open CV é construído em C porém possui interface para C, C++, Python, Java e MATLAB. As aplicações podem ser executadas no Windows, Linux, Android e Mac OS. A interface para aplicações Java é bastante simples. Basta efetuar o download da biblioteca, instalar em seu computador e após isso é necessário apenas incluir a *Application Programming Interface (API)* no projeto.

O Open CV é constituído de um grupo de módulos. Cada módulo é responsável por um tipo de aplicação relacionada com visão computacional, desde algoritmos simples até algoritmos complexos. Abaixo apresentamos um dos módulos, descrevendo-os de forma breve.

Core: oferece a estrutura básica para os demais módulos funcionarem. Esse módulo oferece operações básicas com matrizes, estruturas de dados e outras operações (OPENCV, 2015).

imgproc: oferece operações voltadas para o processamento de imagem onde temos como exemplo aplicações de filtros linear e não-linear, operações com histogramas e muitos outros (OPENCV, 2015).

vídeo oferece algoritmos para análise de vídeo como por exemplo rastreamento

de objetos (OPENCV, 2015).

calib3d: oferece recursos para calibrar a câmera e também reconstrução 3D (OPENCV, 2015).

features2d: oferece várias funcionalidades para descrição e categorização de objetos 2D (OPENCV, 2015).

objdetect: oferece vários algoritmos para detecção de objetos com instâncias já definidas como por exemplo detecção de pessoas ou carros (OPENCV, 2015).

highgui: oferece uma interface simples para captura de vídeo e imagem (OPENCV, 2015).

gpu: oferece algoritmos encontrados nos módulos citados, porém acelerados pela *Graphics Processing Unit (GPU)* (OPENCV, 2015).

3.3 Tesseract

Tesseract é um OCR *Open Source*, e atualmente é distribuídos sobre a licença Apache (TESSERACT, 2015). OCRs são interpretadores de caracteres.

Criado por Hewlett-Packard Laboratories Bristol e Hewlett-Packard, Tesseract foi escrito em C e C++, possuindo suporte a várias línguas, entre elas o português (TESSERACT, 2015).

O Tesseract pode ser treinado, para aumentar sua precisão de reconhecimento. Toda ferramenta é bem documentada possuindo também uma versão para ser utilizado no Android.

A versão Android do OCR possui algumas dependências que torna complexa a integração com qualquer aplicativo. Para simplificar a utilização do OCR, utilizamos a extensão denominada “tess-two’ que já possui além do Tesseract a biblioteca de processamento de imagens chamada de Leptonica (TESSTWO, 2015). Esta extensão facilitou bastante a integração com nosso aplicativo.

Para utilizar os serviços de reconhecimento do OCR apenas é necessário passar por parâmetro o caminho do arquivo de treinamento como mostramos no Código 3.1. Para reconhecer os caracteres de uma imagem apenas é necessário passar a imagem por parâmetro e pegar o resultado como mostra no Código 3.2.

Código 3.1: Iniciando OCR.

```
TessBaseAPI baseApi = new TessBaseAPI();
String fileName = Environment.getExternalStorageDirectory().getPath() +
    "/tesseract";
baseApi.init(fileName, "por");
```

Código 3.2: Pegando resutado do reconhecimendo de caracteres de uma imagem.

```
baseApi.setImage(imagem);
String resultado = baseApi.getUTF8Text();
```

Existe também alguns sites online que oferecem serviços de reconhecimento de caracteres utilizando Tesseract. Este sites servem de exemplo para mostrar o poder de reconhecimento do OCR. O primeiro site é <http://www.free-ocr.com/> que oferece opção de reconhecimento por linguagem. O segundo <http://www.i2ocr.com/> também utiliza Tesseract como OCR com a diferença que pode reconhecer imagens online.

3.4 Lições do Capítulo

Como podemos ver durante este capítulo, enfatizamos as principais tecnologias relacionadas ao nosso trabalho. A utilização do Open CV e do OCR Tesseract são indispensáveis para poder construir a ferramenta. O Open CV oferece todos os métodos necessários para construir uma aplicação de processamento ou análise de imagens, diminuindo muito o tempo de desenvolvimento se fosse necessário criar métodos do zero.

O Tesseract é uma ferramenta bastante antiga no mercado de reconhecimento de caracteres, e além disso possibilita o treinamento para melhorar o reconhecimento, possibilitando uma expansão muito maior do software.

4 Trabalhos Relacionados

Neste capitulo relatamos a realização de um mapeamento sistemático da literatura. O mapeamento sistemático tem como objetivo obter uma visão geral sobre determinada área (PETERSEN et al., 2008). Nós utilizamos o mapeamento sistemático para investigar o estado da arte dos trabalhos relacionados aos sistemas de reconhecimento de diagramas ou estruturas semelhantes. Na seção 4.1 é apresentado como foi aplicada o mapeamento sistemático. Na seção 4.2 apresentamos os resultados da aplicação do mapeamento. Por fim na seção 4.3 apresentamos as lições do capítulo.

4.1 Protocolo de Mapeamento

O objetivo do mapeamento foi investigar o estado da arte dos sistema de reconhecimento de esboços de diagramas UML ou semelhantes. As seguintes questões de pesquisas foram levantas:

- Quais técnicas de aquisição de imagem foram utilizadas?
- Quais filtros foram utilizados?
- Quais técnicas de segmentação de imagem foram utilizadas?
- Quais técnicas de reconhecimento de padrões foram utilizadas?

O mecanismo de busca que utilizamos para a pesquisa foi o *Google Scholar* por indexar várias bases de dados contendo artigos científicos. Selecionamos as palavras chaves "*Image Recognition* + UML" e "UML + *Sketch Recognition*" para efetuar a pesquisa.

Selecionamos os seguintes critérios de inclusão:

- Trabalhos dos últimos 5 anos.
- Trabalhos que abordem o desenvolvimento de sistemas de reconhecimento de imagens.

Selecionamos os seguintes critérios de exclusão:

- Trabalhos do tipo position paper, agenda de pesquisa, revisão sistemática, mapeamento sistemático, survey, relatório técnico.
- Trabalhos com menos de 6 páginas.

- Trabalhos repetidos.

Definimos a classificação segundo as questões de pesquisa e de cada artigo selecionado extraímos o objeto, metodologia e resultados.

4.2 Resultado do Mapeamento

Ao executar o protocolo encontramos 307 resultados. Dentro desses resultados foram selecionados 18 artigos que respondiam ao menos uma das questões levantadas no protocolo. Alguns dos artigos eram de domínio especifico como por exemplo UML. Os demais artigos eram sobre um domínio mais geral propondo reconhecimento de esboço para os mais diversos temas. Na Tabela 1 é possível visualizar todos os artigos encontrados e a qual domínio o mesmo pertence.

Titulo	**Dominio**	**Autor**
Extracting UML Models from Images	UML	Karasneh e Chaudron (2013)
Sketch Recognition via String Kernel	Esboço Geral	Liao e Duan (2012)
A Holonic Multi-Agent System For Sketch, Image and Text Interpretation in The Rock Art Domain	Esboço Geral	Mascardi et al. (2014)
Sketch Recognition using Domain Classification	Esboço Geral	Vashisht, Choudhury e Prasad (2012)
Query-adaptive shape topic mining for hand-drawn sketch recognition	Esboço Geral	Sun et al. (2012)
IMISketch: An interactive method for sketch recognition	Plantas Arquitetônicas	Ghorbel et al. (2014)
Intermodal image-based recognition of planar kinematic mechanisms	Mecanismos Cinemáticos Planares	Eicholtz e Kara (2014)
Local context-based recognition of sketched diagrams	Diagrama	Costagliola, Rosa e Fuccella (2014)
FlexiSketch: A Mobile Sketching Tool for Software Modeling	UML	Wüest, Seyff e Glinz (2013)
Collaborative Creativity From Hand Drawn Sketches to Formal Domain Specific Models and Back Again	UML	Bartelt, Vogel e Warnecke (2013)
First experiments on a new online handwritten flowchart database	Fluxogramas	Awal et al. (2011)
Towards Tool Support For Agile Modeling: Sketching Equals Modeling	UML	Buchmann (2012)
Using data mining for digital ink recognition: Dividing text and shapes in sketched diagrams	Esboço Geral	Blagojevic et al. (2011)
Towards Recovering Archtectural Information from Images of Archtectural Diagrams	Diagramas Arquiteturais	Maggiori et al. (2014)
CogSketch: Sketch Understanding for Cognitive Science Research and for Education	Esboço Geral	Forbus et al. (2011)

CSTutor: A Sketch-Based Tool for Visualizing Data Structures	Estrutura de Dados	Buchanan e Jr (2014)
From engineering diagrams to engineering models: Visual recognition and applications	UML	Fu e Kara (2011)
Understanding, Manipulating and Searching Hand-Drawn Concept Maps	Mapas Conceituais	Jiang et al. (2011)

Tabela 1: Artigos selecionados

Nas próximas seções vamos apresentar como os artigos selecionados respondem as questões de pesquisa.

4.2.1 Quais técnicas de aquisição de imagem foram utilizadas?

Nos artigos pesquisados foram encontradas duas formas de aquisição de imagem. O primeiro tipo consistia na entrada de uma imagem completa contendo um desenho ou esboço e a segunda consistia em um desenho feito diretamente pelo usuário em tempo real no sistema.

Karasneh e Chaudron (2013) em sua ferramenta denominada *Img2uml* tem como entrada imagens de livros feitas por outras ferramentas CASE. Maggiori et al. (2014) também apresentam uma ferramenta com o mesmo tipo de aquisição porem para extrair diagramas arquiteturais. Para plantas arquitetônicas Ghorbel et al. (2014) usam como entrada em sua ferramenta desenhos técnicos feitos a mão para serem reconhecidos. Já para o reconhecimento de mecanismos cinemáticos Eicholtz e Kara (2014) utilizam tanto imagens de livros como desenhos feitos a mão.

Proporcionando uma maior flexibilidade os desenhos feitos diretamente em um dispositivo sensível ao toque são em sua grande maioria reconhecidos momentos depois de seu desenho. Vashisht, Choudhury e Prasad (2012) tem uma preocupação maior com usuário e tentam oferecer uma ferramenta que pode ser utilizada em qualquer domínio. A aquisição da imagem é feita pelo desenho feito pelo usuário na área do sistema que depois é reconhecida e apresentada em uma segunda tela em baixo. No domínio de reconhecimento de fluxograma Awal et al. (2011) utilizam telas sensíveis ao toque para aquisição de imagem. Outra ferramenta com proposta parecida é a *FlexiSketch* criada por Wüest, Seyff e Glinz (2013), tem como objetivo simular um quadro branco e receber esboços de diagramas UML desenhados diretamente na ferramenta.

Buchmann (2012) propõe uma simulação de um quadro branco, oferecendo liberdade ao desenvolvedor. A aquisição da imagem é feito pelo desenho do desenvolvedor na tela do software. Buchmann (2012) implementou sua ferramenta de reconhecimento de

diagramas UML no ambiente de desenvolvimento eclipse. Uma ferramenta de modelagem denominada *Graphical Modeling Framewok* (GMF) possibilitou ao usuário desenhar os esboços na própria tela do ambiente de desenvolvimento. Esse tipo de aquisição de imagens facilita a modelagem de diagramas em telas sensíveis ao toque, sem perder a liberdade de um quadro branco.

4.2.2 Quais filtros foram utilizados?

Poucos filtros foram utilizados nos artigos, pois a maior parte das imagens de entrada foram feitas pelo usuário diretamente no sistema, isso elimina grande parte do ruído ou qualquer outro tipo de distorção na imagem gerada muitas vezes pelo ambiente.

Sun et al. (2012) utilizam vários filtros para a distorção de elementos desenhados pelo usuário, para aumentar o número de variações dos elementos, sem a necessidade da interação do usuário. Porém os filtros utilizados não foram especificados no artigo.

4.2.3 Quais técnicas de segmentação de imagem foram utilizadas?

A segmentação também foi pouco utilizada nos artigos encontrados pois a maior parte dos desenhos eram analisados de forma isolada logo após o usuário desenhar. Apesar disso foram encontradas algumas técnicas para separação do diagrama e do texto.

Mascardi et al. (2014) utilizaram o filtro de *Kalman* para detecção de segmentos distintos em plantas arquitetônicas.

Awal et al. (2011) tem como um dos principais problemas a separação do texto e dos gráficos, pois cada um deles é tratado de forma diferente. Textos são mais complexos que gráficos, para separar os dois foi apresentado uma técnica para medir a complexidade chamada de entropia . Através da entropia o grau de desordem é medido utilizando ângulo de pontos. Isso possibilita verificar a complexidade dos elementos, como o texto possui linhas mais complexas é possível separar a partir desta técnica. Outro método também utilizado para separar o diagrama do texto é a binarização linear absoluta que Maggiori et al. (2014) aplicou juntamente com outras técnicas nos pixeis.

Blagojevic et al. (2011) fizeram análise de algoritmos para separação do texto e das formas baseados na mineração de dados. Foram testados várias técnicas para a separação do texto e das formas. O algorítimo que possuiu maior precisão foi *LogitBoost* e em segundo lugar *LADTree* 1. Em relação ao tempo *Entropy* foi o com melhor desempenho executando em 0.0004 segundos, e o algorítimo Microsoft executando a 0.0159 segundos o reconhecimento de um traço.

Jiang et al. (2011) utilizaram algoritmo de partição de gráficos para separar em vários subgrafos cada mapa conceitual. A partir destes subgrafos é utilizada programação dinâmica para extrair os blocos de conceitos.

4.2.4 Quais técnicas de reconhecimento de padrões foram utilizadas?

Os artigos pesquisados e estudados tratavam de reconhecimento de estruturas de um domínio específico ou geral. Grande parte dos métodos utilizados em domínio específico tinham uma melhor taxa de reconhecimento porém de certa forma se limitavam ao reconhecimento de um domínio específico em relação aos métodos de reconhecimento mais genéricos.

Para o reconhecimento de esboço em um domínio geral Liao e Duan (2012) utilizam uma sequência de caracteres que representam cada primitiva desenhada pelo usuário. São recebidos duas sequências de caracteres e é calculado a distância entre as mesmas utilizando métodos *Kernel*. *Support Vector Machine (SVM)* também é utilizada e treinada com novos exemplos, para possibilitar a classificação das primitivas.

Para o reconhecimento de formas Vashisht, Choudhury e Prasad (2012) utilizaram em sua ferramenta técnicas de classificação de domínio para o reconhecimento de formas. O sistema construído possui uma interface fácil, porém quando muitos esboços são feitos, tem um tempo significativo de espera pelo usuário. Outra questão é que apesar de demorar a reconhecer cada forma, a ferramenta consegue ficar em tempo real com cerca de até 186 formas esboçadas ao mesmo tempo.

A ferramenta de reconhecimento geral construída por Sun et al. (2012) utilizou um banco de dados com mais de um milhão de imagens para auxiliar no reconhecimento dos objetos. A partir dessas imagens foi utilizado um modelo matemático probabilístico. Foi criado um método denominado de *Query-adaptive Shape Topic* (QST) que analisa as imagens semelhantes, então apresenta uma resposta com base da descrição das imagens para o usuário. O padrão utilizado para a descrição das imagens é o MPEG-7, porém os autores criam outro padrão de descrição chamado de Sketch-500.

Forbus et al. (2011) apresentaram uma ferramenta com objetivo de simular uma variedade de tarefas de raciocínio visual e espacial por meio de esboços. Cada esboço feito pelo usuário é denominado glifo. O reconhecimento é feito utilizando uma base de conhecimento alimentada por glifos. Outro método utilizado é o *region connection calculus* (RCC) que tem por objetivo definir a localização espacial do objeto.

A falta de naturalidade das ferramentas CASE e o retrabalho de transferir tais documentos para o computador levou Costagliola, Rosa e Fuccella (2014) a apresentar uma ferramenta de reconhecimento de diagramas. A Metodologia foi dividida 4 em camadas: camada de separação de texto e gráficos, camada do reconhecimento de símbolos, camada de detecção do contexto local e por fim a camada de detecção do diagrama. O reconhecimento é feito em contexto local utilizando aprendizado de máquina.

Também em um domínio de reconhecimento de diagramas Maggiori et al. (2014) utilizaram OpenCV para reconhecimento de diagramas arquiteturais. Para a detecção das

relações foi utilizado Transformada de *Hough* e uma série de outras operações em cima dos pixeis da imagem. Também foi utilizado algoritmos de aprendizado de máquina para verificar a forma no final da linha que representa a relação. Os textos foram reconhecidos utilizando um OCR de código aberto chamado de Tesseract.

Uma ferramenta de reconhecimento de diagramas de classe foi criada por Karasneh e Chaudron (2013). Essa ferramenta foi criada com a linguagem de programação da Microsoft, Visual Basic.NET. Junto com a ferramenta foi utilizado a biblioteca Aforge.NET para o reconhecimento de elementos padrões dos diagramas UML como por exemplo os quadrados. Para o processamento de texto foi utilizado a ferramenta *Microsoft Image (MODI)*.

O sistema denominado *FlexiSketch* criado pelos Wüest, Seyff e Glinz (2013) tem como objetivo integrar ferramentas CASE e esboços feitos por engenheiros. O algoritmo utilizado na ferramenta, foi adaptado de um algoritmo baseado na distância de cadeia de *Levenshtein* que calcula a distância entre duas cadeias. A linguagem de programação utilizada foi Java. Bartelt, Vogel e Warnecke (2013) têm como proposta um ambiente colaborativo utilizando esboços UML que também utiliza o cálculo de distância de *Levenshtein*. Cada elemento recebe uma sequência de números com base na sua intersecção em um quadro. Cada elemento é comparado com uma base de conhecimento e é reconhecido levando em conta a distância *Levenshtein*. Essa base de conhecimento é preenchida por meio do usuário, fazendo o esboço e definindo qual o nome do elemento. Buchmann (2012) apresenta um sistema que tem como proposta simular um quadro branco para o desenvolvedor de software e utiliza também algoritmo baseado na distância *Levenshtein*.

Para o reconhecimento de diagramas feitos por ferramentas ou diagramas feitos a mão Fu e Kara (2011) apresentam uma abordagem para reconhecimento que utiliza Rede Neural Convolucional (CNN) treinada pelo usuário.

Utilizando análise em árvore Ghorbel et al. (2014) criaram três versões de uma ferramenta para reconhecimento de plantas arquitetônicas. A primeira versão explora todas as decisões da arvore contendo o conjunto de primitivas segmentadas. Na segunda versão é criado uma nova árvore que explora as mesmas primitivas. E na terceira versão é explorado a nova árvore criada na segunda versão com a utilização de polígonos das primitivas.

Para o reconhecimento de mecanismos planares cinemáticos Eicholtz e Kara (2014) utilizaram SVM. O objetivo era reconhecer as arestas das figuras, então a SVM era treinada para reconhecer esboços de livros e esboços feitos a mão.

O reconhecimento de fluxogramas online em uma ferramenta criado por Awal et al. (2011) utilizou-se *Time Delay Neural Network (TDNN)* e SVM como classificadores. Para os testes da ferramenta foi disponibilizado um banco de dados online contendo

fluxogramas.

4.3 Lições do Capítulo

Após a execução do protocolo percebemos uma predominância de técnicas de aquisição em tempo real das informações dos usuários. Os usuários desenham diretamente na ferramenta seja com o mouse ou por meio de uma tela sensível ao toque. Um forte argumento entre os artigos é que o mesmo possibilita simular um quadro branco e oferecer um ambiente livre para esboçar.

Como grande parte dos artigos não tinham como aquisição uma imagem obtida por meio de uma câmera, os filtros foram menos comentados nos artigos.

A questão de segmentação também possui poucas informações em relação as demais questões, pois os elementos já vem segmentados a partir do desenho do usuário.

A questão de reconhecimento foi respondida por quase todos os artigos, que utilizavam desde redes neurais para o reconhecimento dos objetos até comparação de cadeias representativas de cada esboço.

5 UML Sketch Recognizer

Neste capítulo apresentamos o aplicativo para Android UML Sketch Recognizer e os artefatos de Engenharia de Software gerados durante o desenvolvimento. Na seção 5.1 abordamos uma visão geral do trabalho, retomando os problemas e apresentando soluções ao qual motivarão a criação da ferramenta. Já na seção 5.2 apresentamos os requisitos encontrados através das *storyboards* e também a visão de solução do software através de diagramas UML. O detalhamento da implementação do protótipo é feito na seção seção 5.3. Por fim na seção seção 5.4 apresentamos as lições do capítulo.

5.1 Visão Geral

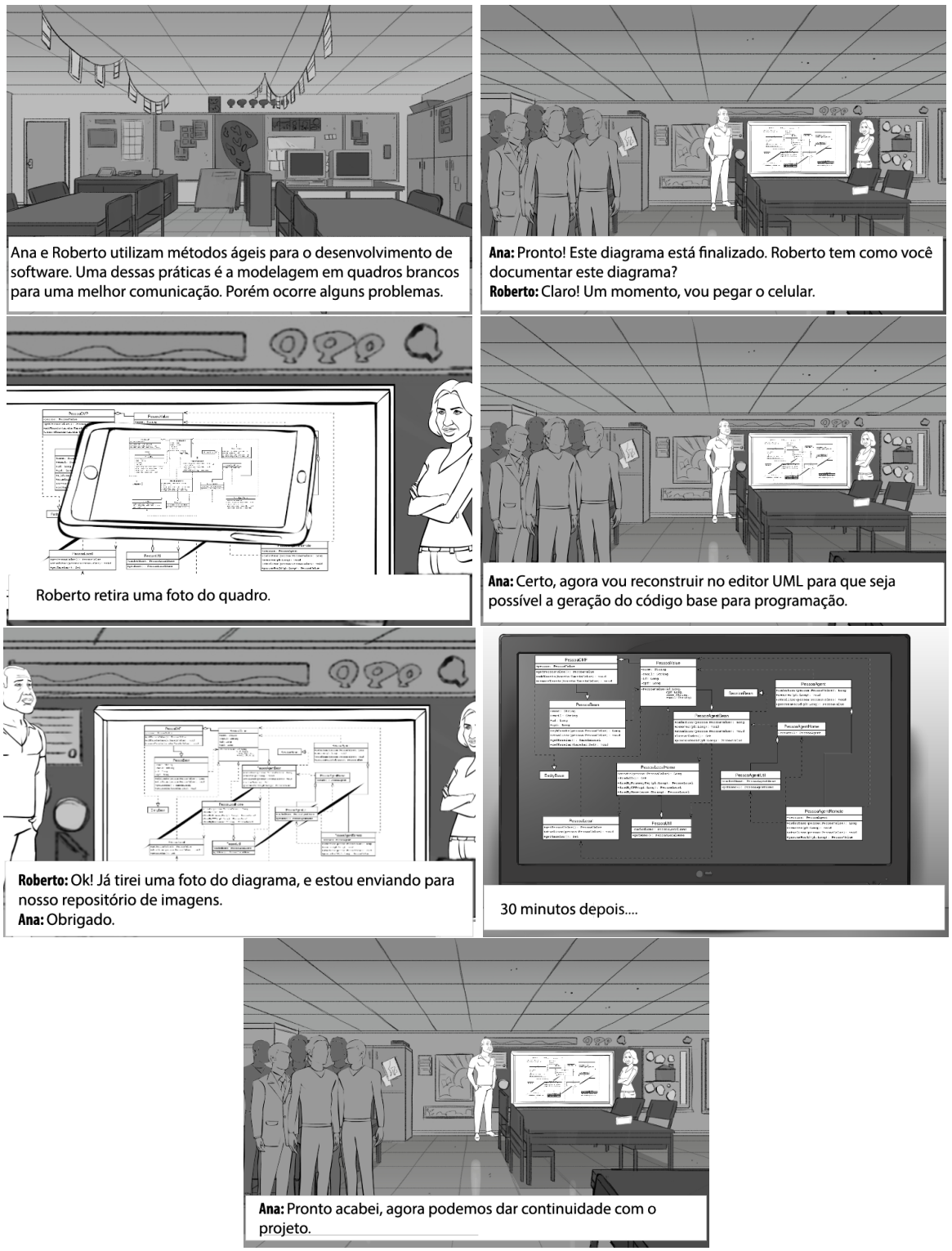
Apresentamos anteriormente na seção de motivação alguns dos problemas que motivaram a criação da ferramenta. A partir de uma *storyboard*[1] construímos um cenário apresentando um dos problemas da modelagem em quadros brancos. Na Figura 7 podemos notar a demora que leva a reconstrução de um diagrama na ferramenta CASE. Mesmo que o diagrama não seja muito grande, existe um retrabalho na construção do diagrama. Analisando os problemas descritos, é possível propor um protótipo que consiga minimizar o retrabalho. Uma ferramenta que possa extrair diagramas UML a partir de uma imagem resolveria o retrabalho de integrar o esboço de um diagrama com uma ferramenta CASE, consequentemente minimizando o trabalho das equipes.

Na Figura 8 apresentamos um cenário onde uma equipe de desenvolvimento possui uma ferramenta que possibilita a extração do diagrama de uma imagem. Podemos notar um contraste dos cenários principalmente no pós modelagem, onde os desenvolvedores já definiram a solução. No primeiro caso apresentado na Figura 7 os desenvolvedores tem retirar uma foto do esboço do diagrama e além disso tem o retrabalho de reconstruir o diagrama em um editor UML para poder gerar codificação ou mesmo para poder atualiza-lo posteriormente. Já na Figura 8 não existe esse trabalho de reconstruir todo diagrama, apenas é necessário capturar a imagem com um aplicativo que vai gerar um arquivo para ser importado em um editor UML, sem necessidades de retrabalho.

A Figura 9 mostra a estrutura da solução. A Figura 9 (a) representa o diagrama que o desenvolvedor necessita editar e na (b) o dispositivo móvel Android que vai fazer a captura da imagem através do aplicativo. Este dispositivo vai detectar em tempo real os elementos do diagrama e também através de uma solicitação do usuário reconhece-los, sem necessidade de nenhum servidor. Assim que reconhecido o diagrama o dispositivo vai

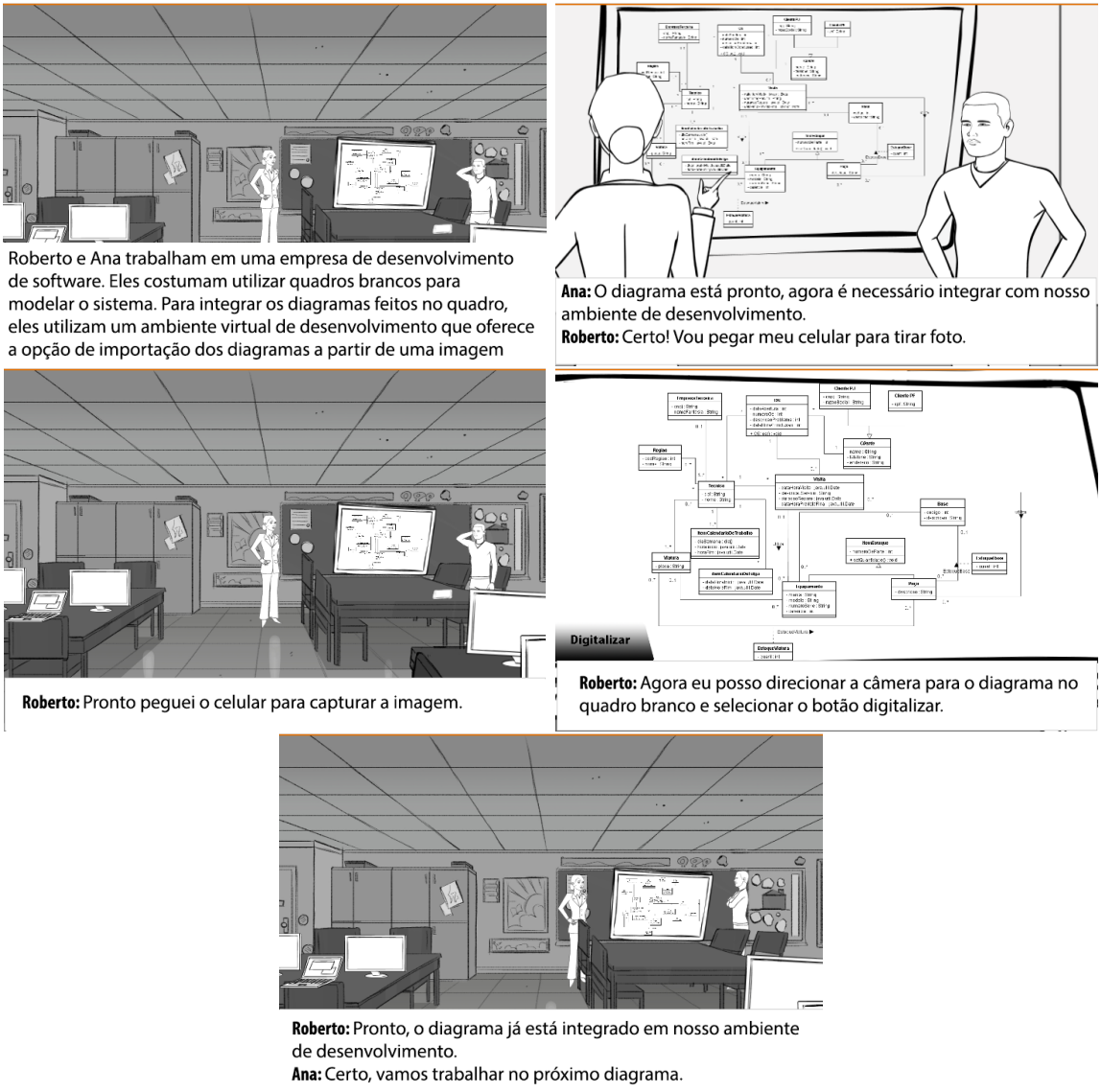
[1] Storyboard é uma sequência de desenhos que representam a interação do usuário com determinado sistema em seu ambiente de trabalho.

Figura 7: Cenário: equipe de desenvolvimento que utiliza quadros brancos para modelagem.



gerar um arquivo XMI como mostrado na Figura 9 (c). Este arquivo pode ser importado em uma ferramenta com suporte a especificação da UML em XMI. Com o diagrama importado como mostra na Figura 9 (d) o usuário pode editar o diagrama ou efetuar qualquer outra funcionalidade que o editor UML ofereça.

Figura 8: Cenário: equipe de desenvolvimento que utiliza quadros brancos para modelagem porém possui uma ferramenta para extração do diagrama.

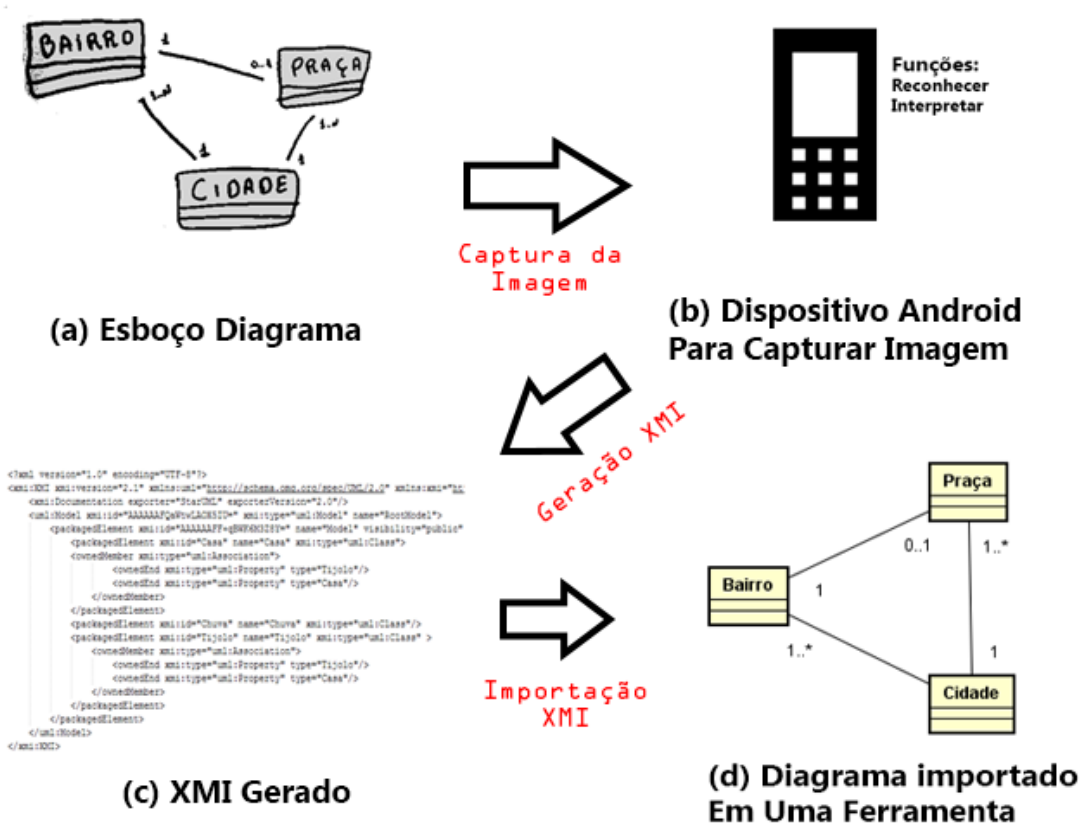


5.2 Análise e Projeto

A partir da *storyboard* encontramos requisitos necessários ao sistema. Organizamos os requisitos através de uma lista apresentada abaixo.

- Detecção e reconhecimento de classes em um diagrama de classes.
- Detecção e reconhecimento de associações em um diagrama de classes.
- Detecção e reconhecimento de multiplicidades em um diagrama de classes.
- Reconhecimento de caracteres do diagrama de classes.
- Geração do diagrama de classes no formato XMI seguindo a especificação UML.

Figura 9: Estrutura solução.



- Utilização de um dispositivo Android para capturar e reconhecer o diagrama de classes.

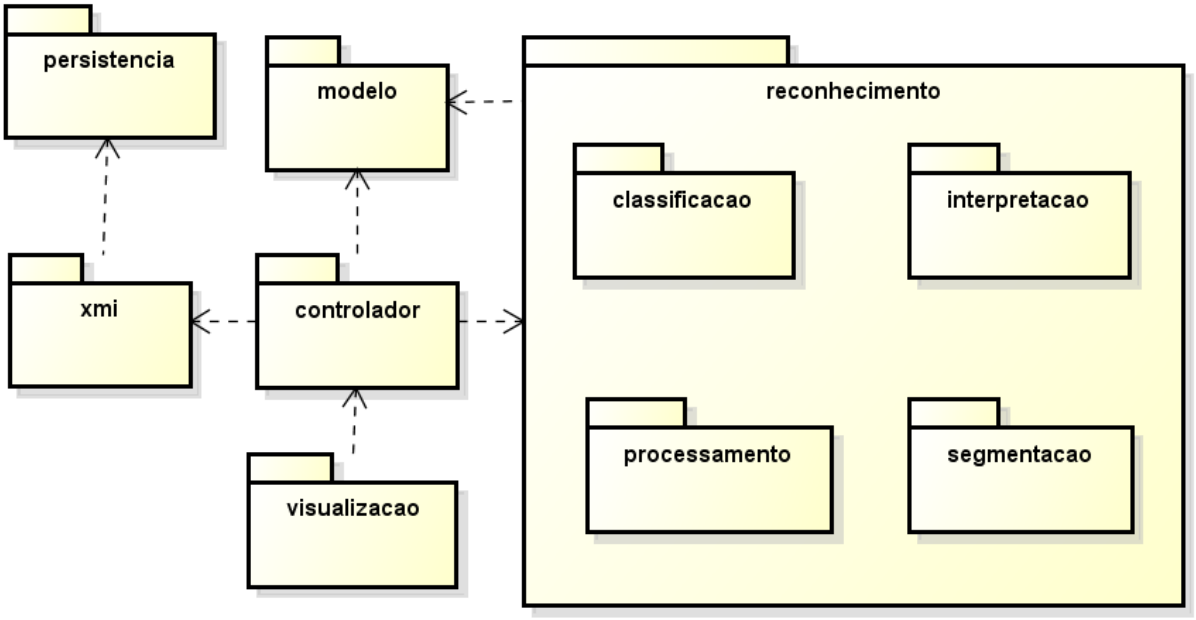
A arquitetura do nosso software possui as seguintes camadas: reconhecimento, persistência, XMI, modelo, controlador e visualização. Cada uma dessas camadas possui responsabilidades distintas. Na Figura 10 podemos ver todas as camadas do sistema.

Na camada de reconhecimento, separamos as responsabilidades em classificação, interpretação, processamento e segmentação. Na camada de processamento, realizamos operações objetivando a melhoria das características da imagem, e também outras operações simples. Na camada de segmentação, extraímos os elementos da imagem. Cada elemento tem suas características como tamanho, largura e altura extraídas da imagem. Na camada de classificação cada elemento é classificado em classe, relação ou multiplicidade. Por fim a ultima camada, denominada de interpretação os elementos são interpretados.

A camada de modelo possui toda as estrutura de classes ligadas ao nosso domínio, auxiliando na comunicação entre as camadas. A camada denominada xmi é semelhante a camada modelo, porém nela são representados a estrutura de um diagrama em relação a versão do XMI.

A camada controlador controla as etapas de reconhecimento, relacionando-se com grande parte das camadas do sistema. A camada de visualização apresenta as estruturas relacionadas com a interface gráfica do sistema, que no caso do Android se encontram normalmente Activities e Views utilizando os serviços de reconhecimento por meio de chamadas feitas na camada controlador.

Figura 10: Arquitetura do Software.



Na Figura 11 apresentamos o diagrama de classes, que mostra as relações entre o controlador e o pacote de reconhecimento do sistema. Podemos notar que nenhuma classe do pacote reconhecimento está conectada. Neste contexto quem utiliza e controla o fluxo de execução dos métodos é a classe controlador. Nenhuma dependência foi criada entre as diferentes funções das classes.

5.2.1 Processo de Reconhecimento

Para conseguir reconhecer os elementos, a nossa ferramenta necessita passar por uma série de fases. Através de um diagrama de atividades na Figura 12 mostramos cada passo necessário para reconhecer o diagrama.

5.3 Detalhes da Implementação

Nesta seção vamos apresentar alguns detalhes da implementação da ferramenta, discutindo e mostrando os métodos utilizados para poder detectar e reconhecer os elementos do esboço UML.

Figura 11: Diagrama de classes apresentando a relação entre o controlador e os serviços de reconhecimento.

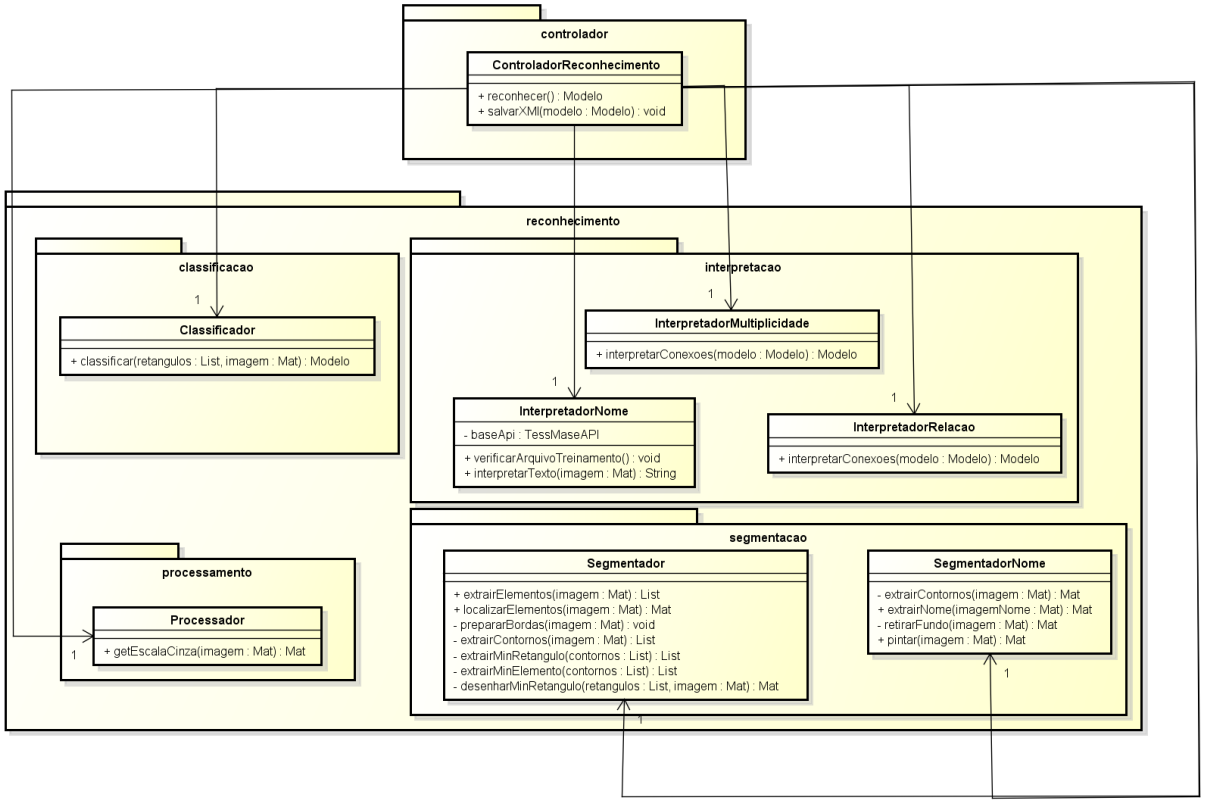
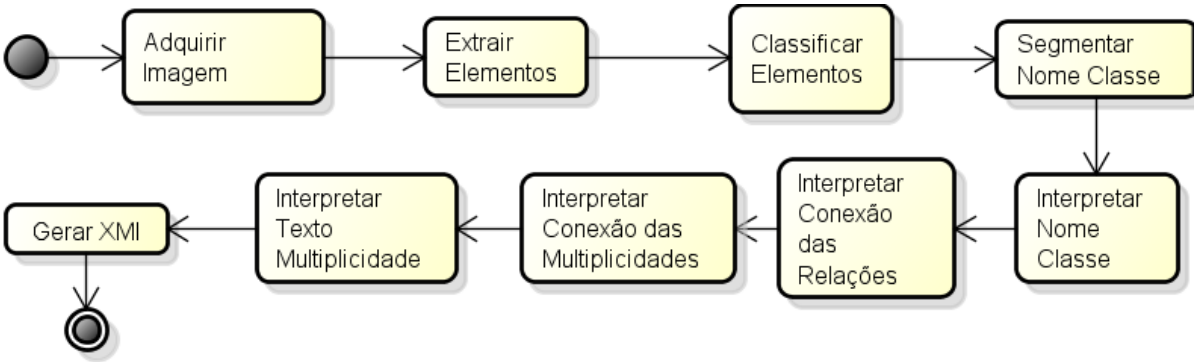


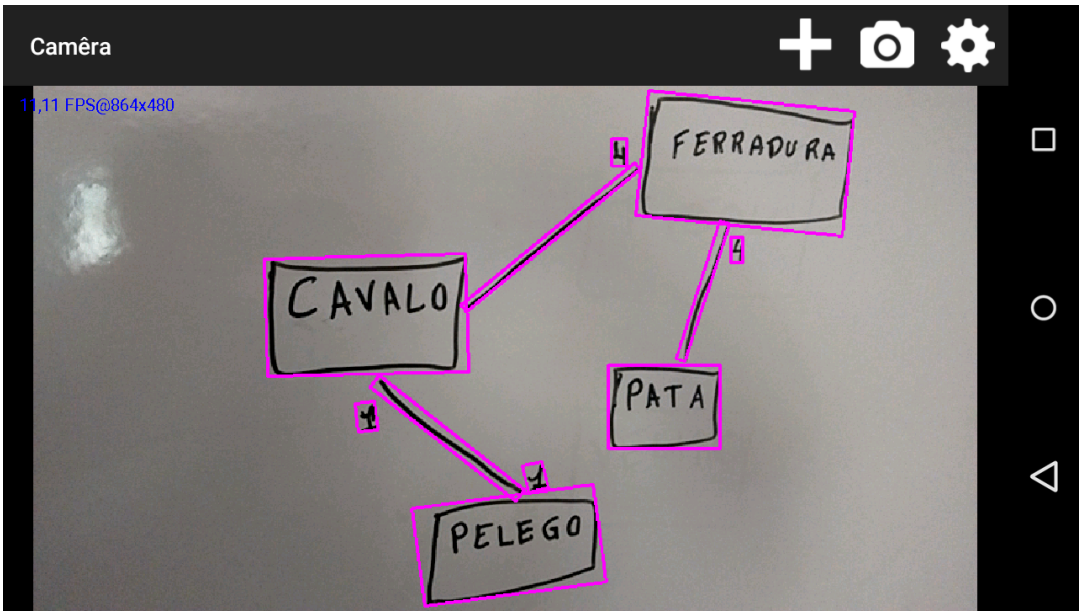
Figura 12: Processo de reconhecimento.



5.3.1 Detecção dos Elementos

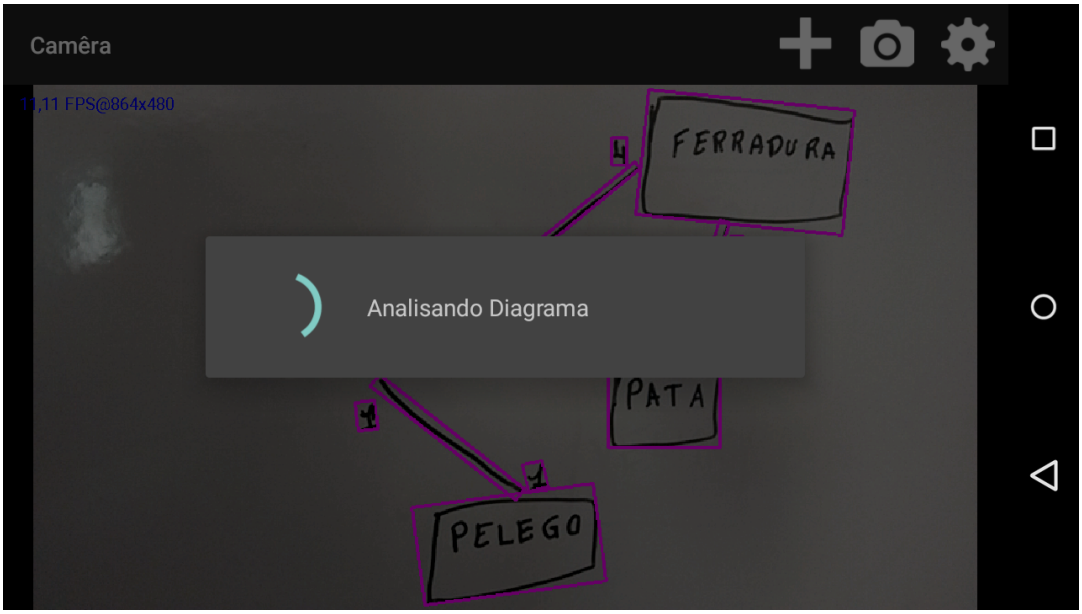
A detecção dos elementos é feita em duas etapas. A primeira etapa ocorre em tempo real quando o usuário abre o aplicativo. Neste momento nossa ferramenta processa cada imagem por segundo e localiza os elementos como é mostrado na Figura 13. O Segundo passo é o momento de extração dos elementos, onde além de serem detectados os mesmos são classificados e interpretados. Os elementos das classes são extraídos através de uma série de processos bastante simples. Primeiramente capturamos a imagem por meio de um toque na tela no dispositivo (*smartphone, tablet*). Este toque na tela

Figura 13: Detecção dos elementos em tempo real.



inicia o processo de reconhecimento, pausando a detecção em tempo real e iniciando o reconhecimento completo da imagem como mostrado na Figura 14.

Figura 14: Iniciando reconhecimento dos elementos.



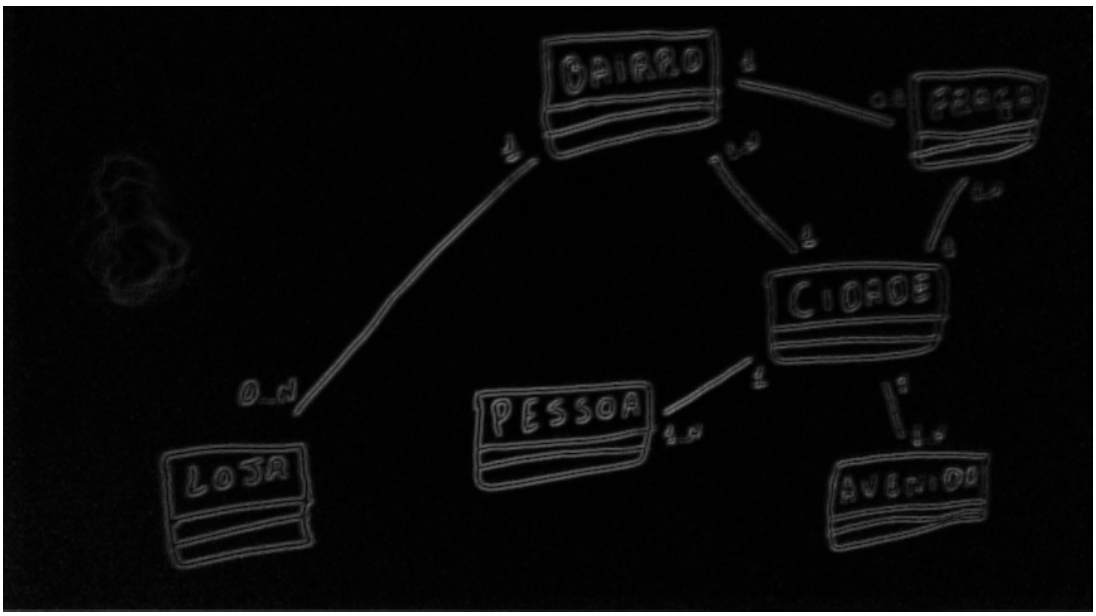
A imagem capturada é processada a fim de transforma-la para uma escala de cinza e facilitar a extração de características. Com a imagem em escala de cinza, utilizamos um operador de morfologia matemática denominado de *Morphological Gradient* que é basicamente a diferença entre a erosão e dilatação da imagem. No Código 5.1 apresentamos a técnica utilizando Open CV e na Figura 15 apresentamos o resultado da operação. Esse método foi utilizado para poder fechar pequenas aberturas nas classes, retirar sombras e

também pequenos reflexos.

Código 5.1: Codificação aplicação da morfologia matemática.

```
int operation = 4;//MORPH_GRADIENT
int sizeKernel = 3;
Mat element = getStructuringElement(0, new Size(sizeKernel, sizeKernel), new
    Point(0, 0));
morphologyEx(imagem, imagem, operation, element, new Point(0, 0), 1);
```

Figura 15: Resultado da aplicação da morfologia matemática.



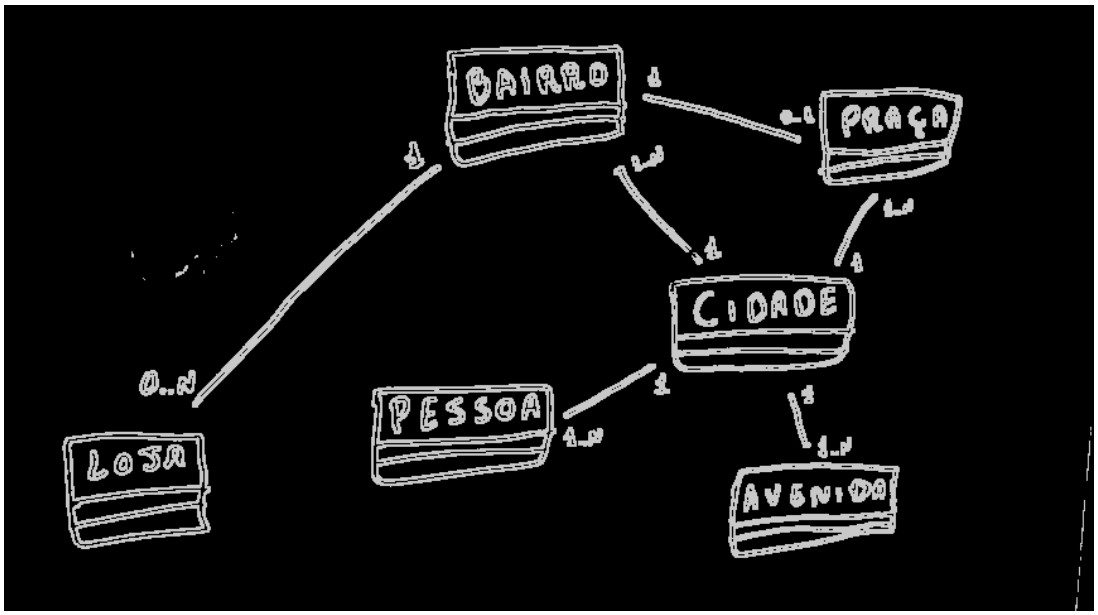
Com as bordas da imagem tratadas, utilizamos uma técnica para segmentação de bordas denominada de Limiarização (*Threshold*). Os parâmetros adicionados no método da Código 5.2 são respectivamente a imagem a ser segmentada, a imagem destino, o limiar, o valor máximo a ser processador e o tipo de limiarização. Executamos vários testes empíricos para definir qual o melhor valor para limiar e o melhor tipo de limiarização. O melhor resultado encontrado foi utilizando o limiar 0 e o tipo de limiarização *OTSU*.

Código 5.2: Codificação do método de segmentação *Threshold*.

```
Imgproc.threshold(imagem, imagem, 0, 200, Imgproc.THRESH_OTSU);
```

Com a imagem segmentada, procuramos todos os contornos utilizando o método "findContours" do Open CV. Extraímos apenas os contornos externos primeiramente, para saber onde está cada elemento. Nesse momento ainda não é necessário pegar outras informações, como as letras da classe.

Figura 16: Resultado segmentação com *Threshold*.



5.3.2 Reconhecimento de Classe

O reconhecimento de classe passa por duas etapas denominadas de classificação e de interpretação de nome. Na primeira etapa vários elementos são extraídos da imagem. Um elemento pode ser considerado uma classe quando o mesmo possuir o tamanho da área maior que 1000 pixeis e também possuir um nome.

Na segunda etapa o nome da classe é interpretado. Primeiramente temos que pegar a imagem contendo apenas a classe. A partir dessa imagem utilizamos o método de limiarização para segmentar as bordas da imagem. Como temos a necessidade de fazer o reconhecimento apenas do nome da classe e não de toda estrutura, tivemos que separar as bordas da classe do nome. Através da diferença de matrizes conseguimos separar o nome das bordas. Pegamos as bordas externas da classe e depois todas as bordas e subtraímos os valores, sobrando apenas o nome da classe.

Com o nome da classe já tratado, levamos a imagem para o reconhecimento através do OCR chamado de Tesseract. Na versão atual da ferramenta, o OCR está treinado apenas para reconhecer letras de forma. Então para conseguir um bom nível de precisão na hora do reconhecimento é necessário desenhar letras separadas das bordas, espaçadas e levemente quadriculadas.

5.3.3 Reconhecimento de Relação

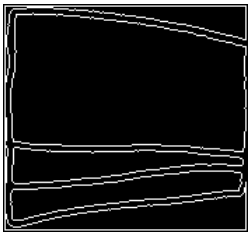
Atualmente nosso software suporta o reconhecimento de associações. As associações simples são diferenciadas dos demais elementos pela diferença entre sua largura e altura. Caso a altura for cinco vezes maior que a largura ou vice versa, classificamos o

Figura 17: Passos que a imagem passou até ser extraído o nome.

(a) Imagem Original

(b) Imagem Limiarizada

(c) Contornos Externos



(d) Todos Contornos

(e) Resultado Subtração Imagem (c) e (d) e preênchimento

elemento como uma relação.

Assim que encontrada as relações do diagrama, devemos verificar quais classes estão relacionadas. As relações não devem encostar nas classes por que existe uma limitação no aplicativo (não foi encontrada nenhuma técnica viável para separação das linhas no momento), apenas devem estar perto. Através de um cálculo de proximidade, utilizando as extremidades da relação, é possível verificar quais classes estão conectadas.

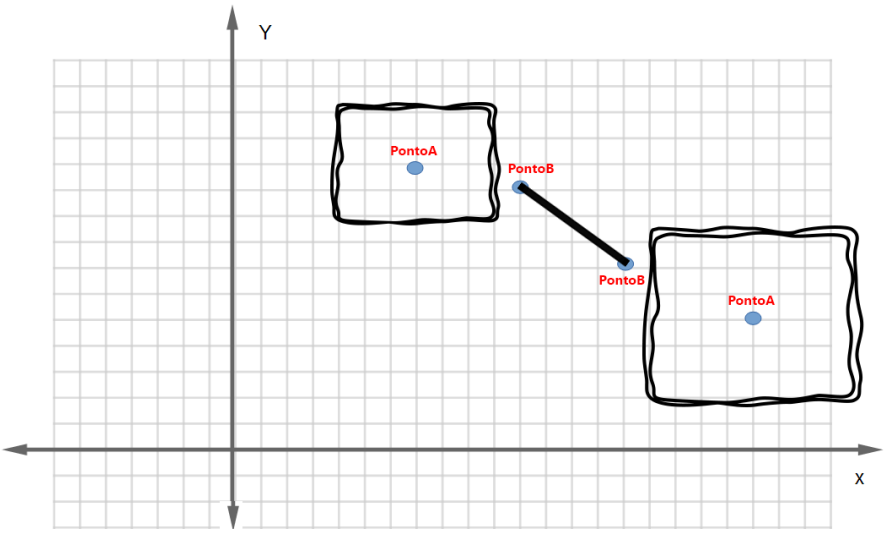
O cálculo para verificar a proximidade entre cada classe e relação é bastante simples. Utilizando uma fórmula derivada do Teorema de Pitágoras, mostrada na Equação 5.1, medimos a distância entre o centro da classe e a extremidade da associação.

$$\sqrt{distancia = (pBX - pAX)^2 + (pBY - pAY)^2} \tag{5.1}$$

Em nosso contexto não é aconselhado utilizar valores fixos para verificar se a classe está conectada ou não, pois o tamanho do desenho da classe e também a resolução do dispositivo vai ser bastante variável. A melhor solução é utilizar a média de tamanho da classe, considerando altura e largura para definir a distância permitida de conexão entre os elementos. O cálculo da distância aceitável é mostrado na Equação 5.2 e estamos considerando um valor um pouco maior que a média comum para contornar a limitação do software que não consegue trabalhar com associações ligadas diretamente a classe. A Figura 18 exemplifica a situação.

$$distanciaAceitavel = altura + largura/1.8 \tag{5.2}$$

Figura 18: Proximidade das relações e classes.



5.3.4 Reconhecimento de Multiplicidade

Cada elemento com área maior que 100 pixeis e menor que 1000 pixeis é definido como multiplicidade. As multiplicidades são ligeiramente menores que as classes, isto foi suficiente para diferencia-las.

O cálculo de proximidade é o mesmo utilizado nas relações, com a diferença que nesse contexto as multiplicidades ficam mais próximas das relações. Então apenas verificamos se cada multiplicidade está próxima da sua respectiva extremidade da associação. Com a proximidade verificada, reconhecemos os caracteres com o OCR;

5.3.5 Geração do XMI

Estamos utilizando a estrutura do XMI versão 2.1. Trabalhamos com a versão 2.4 inicialmente, porém não encontramos nenhuma ferramenta que ofereça suporte a essa versão. Para versão 2.1 do XMI encontramos a ferramenta Star UML. Este editor UML importa qualquer arquivo no formato XMI, possuindo também opções de exportação.

No Código 5.3 apresentamos um exemplo de estrutura XMI seguindo a especificação 2.1 da UML e que pode ser importado pela ferramenta Star UML.

Código 5.3: Exemplo de um XMI gerado pela nossa ferramenta UML Sketch Recognizer.

```
<xmi:XMI xmlns:xmi="http://schema.omg.org/spec/XMI/2.1"
    xmlns:uml="http://schema.omg.org/spec/UML/2.0">
   <uml:Model name="Diagrama" xmi:type="uml:Model">
      <packagedElement name="diagrama" xmi:type="uml:Model">
         <packagedElement name="PELEGa" xmi:type="uml:Class" xmi:id="PELEGa">
            <ownedMember xmi:type="uml:Association">
```

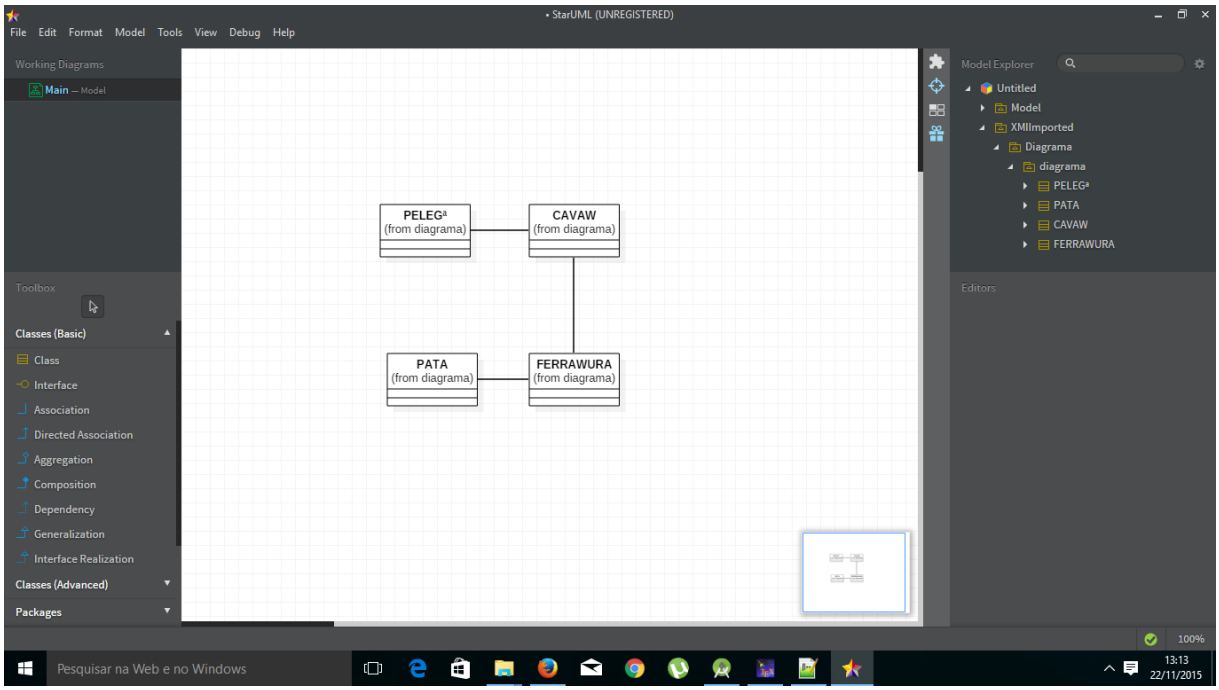
```
            <ownedEnd type="CAVAW" xmi:type="uml:Property"/>
            <ownedEnd type="PELEGa" xmi:type="uml:Property"/>
            </ownedMember>
         </packagedElement>
         <packagedElement name="PATA" xmi:type="uml:Class" xmi:id="PATA">
            <ownedMember xmi:type="uml:Association">
            <ownedEnd type="FERRAWURA" xmi:type="uml:Property"/>
            <ownedEnd type="PATA" xmi:type="uml:Property"/>
            </ownedMember>
         </packagedElement>
         <packagedElement name="CAVAW" xmi:type="uml:Class" xmi:id="CAVAW">
            <ownedMember xmi:type="uml:Association">
               <ownedEnd type="CAVAW" xmi:type="uml:Property"/>
               <ownedEnd type="PELEGa" xmi:type="uml:Property"/>
            </ownedMember>
            <ownedMember xmi:type="uml:Association">
               <ownedEnd type="FERRAWURA" xmi:type="uml:Property"/>
               <ownedEnd type="CAVAW" xmi:type="uml:Property"/>
            </ownedMember>
         </packagedElement>
         <packagedElement name="FERRAWURA" xmi:type="uml:Class"
             xmi:id="FERRAWURA">
            <ownedMember xmi:type="uml:Association">
               <ownedEnd type="FERRAWURA" xmi:type="uml:Property"/>
               <ownedEnd type="PATA" xmi:type="uml:Property"/>
            </ownedMember>
            <ownedMember xmi:type="uml:Association">
               <ownedEnd type="FERRAWURA" xmi:type="uml:Property"/>
               <ownedEnd type="CAVAW" xmi:type="uml:Property"/>
            </ownedMember>
         </packagedElement>
      </packagedElement>
   </uml:Model>
</xmi:XMI>
```

Atualmente temos um problema ao importar qualquer XMI para a ferramenta Star UML (versão 5.0) pois não está conseguindo importar as multiplicidades. Esse problema está ocorrendo tanto em arquivos gerados pelo editor UML como pela nossa ferramenta. Estamos procurando outras ferramentas com suporte a importação do XMI porém ainda não encontramos.

Na Figura 19 apresentamos o Código 5.3 XMI importador na ferramenta Star

UML.

Figura 19: XMI importado na ferramenta Star UML.



5.4 Lições do Capítulo

Durante este capítulo apresentamos como foi construído o software, desde sua análise até sua implementação. Conseguimos construir um protótipo funcional, através de três grandes incrementos, que oferecem suporte ao reconhecimento de classes, associações e multiplicidades.

Para resolver os problemas durante o desenvolvimento testamos várias técnicas de processamento e análise de imagens, e conseguimos ter sucesso na utilização das mais simples e também mais relevantes ao contexto. Como as técnicas utilizadas foram bastante simples, conseguimos obter um software relativamente rápido, que pode ser utilizado diretamente em um *smartphone* ou *tablet*.

O protótipo que construímos foi denominado de UML Sketch Recognizer e já pode ser utilizado por desenvolvedores. A versão beta esta disponível no endereço: <www.umlsketch.hol.es>.

6 Verificação e Validação

Após a implementação do software, realizamos alguns testes e também uma validação com possíveis usuários. Na seção 6.1 apresentamos quais testes foram realizados para verificar o funcionamento do produto. Na seção 6.2 apresentamos um teste feito com alguns alunos usando a ferramenta UML Sketch Recognizer. Por fim na seção 6.3 apresentamos as lições do capítulo.

6.1 Verificação da Solução

Para realizarmos os testes, capturamos várias imagens com seus respectivos XMI. Comparamos cada classe com o seu respectivo arquivo XMI. Com os valores de reconhecimento fizemos uma média geral conseguindo como resultado a % de acerto na detecção e reconhecimento.

Realizamos os testes com 10 diagramas de classe, utilizando o celular Moto G2, com a resolução minima de 864 x 480. Os diagramas testados estão inseridos em um ambiente controlado que deveriam seguir as seguintes regras:

- As associações não devem encostar nas classes.
- As multiplicidades não devem encostar nas relações.
- As classes devem estar fechadas (aberturas minimas são retiradas por meio da morfologia matemática, porém aberturas maiores não).
- O nome da classe deve ser escrito em letra de forma o mais legível possível.
- O nome da classe não devem encostar nas bordas da classe.

Na Tabela 2 é possível visualizar os resultados dos testes. No momento fizemos testes apenas com ambientes controlados, sem considerar sombras, reflexos ou alguma distorção do ambiente.

Tabela 2: Resultado dos testes

Elemento	**% Detecção**	**% Reconhecimento**
Classes	100%	80%
Associações	100%	90%
Multiplicidades	70%	30%

A alta porcentagem de detecção nas classes e associações acontece pois estamos trabalhando em um ambiente controlado, e com um *feedback* constante para o usuário do que está sendo detectado ou não. O usuário tem uma resposta constante diretamente no *smartphone* do que está sendo detectado ou não, oferecendo a possibilidade de tentar capturar a imagem da melhor maneira possível. As multiplicidades tem uma taxa de detecção um pouco mais baixa pois em alguns momentos a informação pode ficar muito pequena, ocasionando que o software desconsidere essa informação.

O reconhecimento de classes está bastante alto para o nosso contexto. Primeiro porque estamos trabalhando com caracteres totalmente variáveis. Cada pessoa tem um tipo de letra, uma forma mais arredondada, quadriculada ou outra variação qualquer. Estas variações diminuem a taxa de reconhecimento, porém conseguimos ótimos resultados para letras legíveis e de forma. O OCR Tesseract possui um dicionário interno, então caso a palavra seja parcialmente reconhecida, ele consegue inferir na maioria das vezes qual é a palavra correta.

O reconhecimento das associações também está bastante alto. Utilizamos cálculo baseado no Teorema de Pitágoras, o que aumentou bastante a taxa de reconhecimento das conexões entre classes.

As multiplicidades tem uma taxa de reconhecimento menor, pois normalmente contêm um ou dois caracteres, não possibilitando ao OCR utilizar seu dicionário interno. Nesse caso o caractere tem que estar ao máximo parecido com a respectiva multiplicidade.

6.2 Validação da Solução

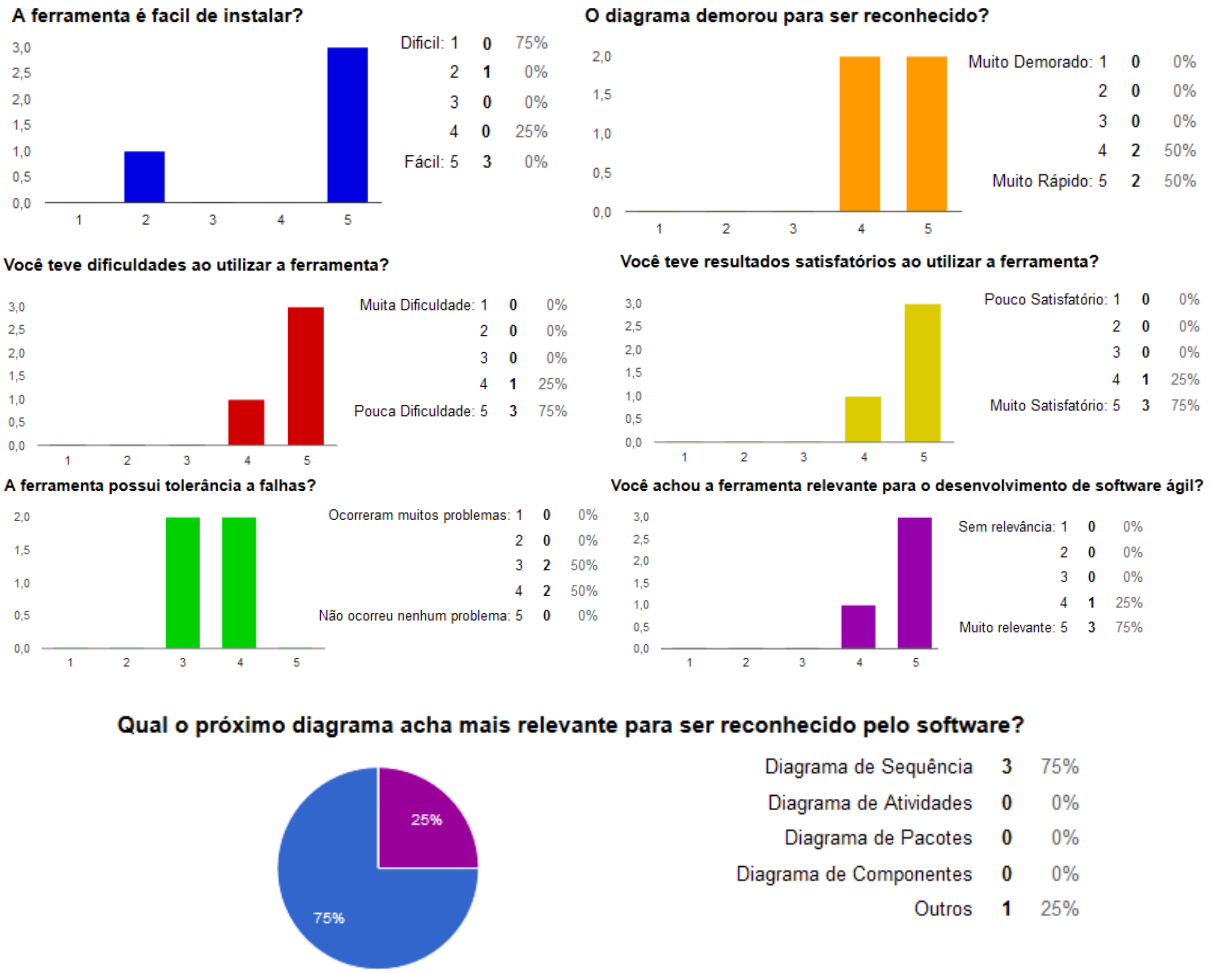
A validação foi feita por um grupo de alunos de Engenharia de Software. Um horário foi marcado para realizar o teste de aceitação.

Conseguimos testar o software com 4 alunos de Engenharia de Software. Cada aluno instalou o software em seu dispositivo, e desenhou alguns diagramas. Assim que os alunos terminaram de utilizar a ferramenta, repassamos um formulário para ser preenchido, contendo algumas perguntas sobre a facilidade de instalação, facilidade de uso, tolerância a falhas e relevância da ferramenta. Também foram adicionados dois campos para informar qual o próximo diagrama a ser reconhecido e considerações.

Na Figura 20 apresentamos as respostas feitas pelos alunos e também qual o próximo diagrama mais relevante a ser reconhecido pela ferramenta.

Os alunos mostraram um grande interesse na ferramenta. Como estamos validando em um ambiente controlado, alguns problemas iniciais ocorreram como os alunos conectarem as associações nas classes. Alguns tiveram alguma dificuldade na instalação pois é necessário baixar o Open CV inicialmente antes da utilização do aplicativo.

Figura 20: Repostas dos alunos que participaram da validação.



Durante a validação encontramos alguns problemas com a conexão das relações, o que já foi corrigido na versão atual do software.

6.3 Lições do Capítulo

A verificação e a validação foram essenciais para o protótipo. Na verificação encontramos problemas na interpretação de conexões entre as classes. Inicialmente conseguimos um percentual em torno de 30% no reconhecimento por causa deste problema. Porém encontramos um método melhor para analisar a distância entre os pontos conseguindo um percentual em torno de 90% no reconhecimento das associações.

A validação foi interessante e gratificante, apesar de testarmos com apenas 4 alunos com disponibilidade de tempo, todos conseguiram instalar o aplicativo sem muitos problemas. Algumas dificuldades iniciais na utilização, pois ainda temos o ambiente controlado de não poder encostar um elemento no outro. Apesar destes pequenos problemas tivemos um aceitação grande dos alunos que validaram e também pessoas que tiveram

um pequeno contato com o aplicativo, fora do horário de validação. A ferramenta sem dúvida causa um impacto positivo em quem a utiliza.

Futuramente podem ser feitos testes considerando ambientes diversos com pouca iluminação, muita sombra, muito reflexo, letras não muito legíveis. Este testes podem ajudar a melhorar as técnicas de detecção e reconhecimento do software, para os mais variáveis ambientes.

7 Conclusões

O principal objetivo do nosso trabalho foi criação de uma ferramenta de reconhecimento de esboços de diagramas UML de fotos. Esta ferramenta objetiva minimizar o trabalho de equipes que normalmente necessitam reconstruir os diagramas feitos em quadros brancos em uma ferramenta CASE apenas para geração do código ou para atualização da documentação.

A construção de uma ferramenta de reconhecimento de esboço passou por vários desafios. O primeiro desafio foi trabalhar com um ambiente controlado de desenvolvimento, que foi apresentado durante os testes. O segundo desafio foi trabalhar com a grande variação do esboço, que depende de como a pessoa desenha o diagrama e de como está o ambiente no momento da captura da imagem. Apesar dessas variações conseguimos resolver problemas como relações e classes com muitas curvas, reflexos pequenos e aberturas minimas em classes. Outras variações como reflexos fortes e sombras muito escuras ainda devem ser tratadas. Também estamos controlando em nosso ambiente a conexão entre elementos, que ainda devem ficar separados para o software reconhecer corretamente o diagrama.

Outro grande desafio foi o primeiro contato ao utilizarmos as técnicas de processamento e análise de imagens. Até as técnicas mais simples possuem um grau complexo de aprendizagem. Além disso cada técnica possui parâmetros que devem ser testados de várias formas, considerando variações de ambiente, luz, sombra, resolução da câmera etc.

Apesar dos desafios apresentados foi possível construir uma ferramenta funcional. Apesar de se limitar ao reconhecimento de certos elementos da UML, a ferramenta ficou bastante interessante e com precisões de detecção e reconhecimento aceitáveis dentro do ambiente controlado.

Os usuários que tiveram contato com a ferramenta tiveram uma boa impressão, pois apesar do ambiente controlado ela automatiza um processo que muitas vezes é demorado e causa retrabalho. Outro ponto positivo é não ter necessidade de um servidor. Apenas com um *smartphone* ou *tablet* é possível reconhecer um diagrama e enviar direto para a nuvem, sem gerar retrabalhos.

Para trabalhos futuros pretendemos adicionar o reconhecimento dos demais elementos de um diagrama de classes, e também adicionar o reconhecimento de outros diagramas. Além disso pretendemos aumentar a taxa de detecção e reconhecimento dos elementos existentes. Atualmente falta reconhecer atributos, métodos e também o tipo de relação em cada diagrama de classe.

Referências

ANDROID. *Android NDK*. 2015. <http://developer.android.com/tools/sdk/ndk/index.html>. Accessed: 2015-06-01. Citado na página 35.

ANDROID. *Android Studio Overview*. 2015. <https://developer.android.com>. Accessed: 2015-06-01. Citado na página 36.

AWAL, A.-M. et al. First experiments on a new online handwritten flowchart database. In: INTERNATIONAL SOCIETY FOR OPTICS AND PHOTONICS. *IS&T/SPIE Electronic Imaging*. [S.l.], 2011. p. 78740A–78740A. Citado 4 vezes nas páginas 40, 41, 42 e 44.

BARTELT, C.; VOGEL, M.; WARNECKE, T. Collaborative creativity: From hand drawn sketches to formal domain specific models and back again. In: *MoRoCo@ ECSCW*. [S.l.: s.n.], 2013. p. 25–32. Citado 2 vezes nas páginas 40 e 44.

BLAGOJEVIC, R. et al. Using data mining for digital ink recognition: Dividing text and shapes in sketched diagrams. *Computers & Graphics*, Elsevier, v. 35, n. 5, p. 976–991, 2011. Citado 2 vezes nas páginas 40 e 42.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. *UML-GUIA DO USUARIO: TRADUÇAO DA SEGUNDA EDIÇAO*. [S.l.]: Elsevier Brasil, 2000. Citado 3 vezes nas páginas 25, 27 e 28.

BUCHANAN, S.; JR, J. J. L. Cstutor: A sketch-based tool for visualizing data structures. *ACM Transactions on Computing Education (TOCE)*, ACM, v. 14, n. 1, p. 3, 2014. Citado na página 41.

BUCHMANN, T. Towards tool support for agile modeling: sketching equals modeling. In: ACM. *Proceedings of the 2012 Extreme Modeling Workshop*. [S.l.], 2012. p. 9–14. Citado 3 vezes nas páginas 40, 41 e 44.

COSTAGLIOLA, G.; ROSA, M. D.; FUCCELLA, V. Local context-based recognition of sketched diagrams. *Journal of Visual Languages & Computing*, Elsevier, v. 25, n. 6, p. 955–962, 2014. Citado 3 vezes nas páginas 21, 40 e 43.

EICHOLTZ, M.; KARA, L. B. Intermodal image-based recognition of planar kinematic mechanisms. *Journal of Visual Languages & Computing*, Elsevier, 2014. Citado 3 vezes nas páginas 40, 41 e 44.

FORBUS, K. et al. Cogsketch: Sketch understanding for cognitive science research and for education. *Topics in Cognitive Science*, Wiley Online Library, v. 3, n. 4, p. 648–666, 2011. Citado 2 vezes nas páginas 40 e 43.

FU, L.; KARA, L. B. From engineering diagrams to engineering models: Visual recognition and applications. *Computer-Aided Design*, Elsevier, v. 43, n. 3, p. 278–292, 2011. Citado 2 vezes nas páginas 41 e 44.

GHORBEL, A. et al. Imisketch: an interactive method for sketch recognition. *Pattern Recognition Letters*, Elsevier, v. 35, p. 78–90, 2014. Citado 3 vezes nas páginas 40, 41 e 44.

GONZALEZ, R. C.; WOODS, R. E. *Processamento Digital de Imagens. Traduçao: Cristina Yamagami e Leonardo Piamonte*. [S.l.]: Sao Paulo: Pearson Prentice Hall, 2010. Citado 6 vezes nas páginas 13, 28, 29, 30, 31 e 32.

HAMMOND, T.; DAVIS, R. Tahuti: A geometrical sketch recognition system for uml class diagrams. In: ACM. *ACM SIGGRAPH 2006 Courses*. [S.l.], 2006. p. 25. Citado na página 21.

JIANG, Y. et al. Understanding, manipulating and searching hand-drawn concept maps. *ACM Transactions on Intelligent Systems and Technology (TIST)*, ACM, v. 3, n. 1, p. 11, 2011. Citado 2 vezes nas páginas 41 e 42.

KARASNEH, B.; CHAUDRON, M. R. Extracting uml models from images. In: IEEE. *Computer Science and Information Technology (CSIT), 2013 5th International Conference on*. [S.l.], 2013. p. 169–178. Citado 3 vezes nas páginas 40, 41 e 44.

LARMAN, C. *Utilizando UML e padrões: uma introdução à análise e ao projeto orientados a objetos*. [S.l.]: Bookman, 2005. Citado na página 21.

LECHETA, R. R. *Google Android-3ª Edição: Aprenda a criar aplicações para dispositivos móveis com o Android SDK*. [S.l.]: Novatec Editora, 2013. Citado na página 35.

LIAO, S.; DUAN, M. Sketch recognition via string kernel. In: IEEE. *Natural Computation (ICNC), 2012 Eighth International Conference on*. [S.l.], 2012. p. 101–105. Citado 2 vezes nas páginas 40 e 43.

MAGGIORI, E. et al. Towards recovering architectural information from images of architectural diagrams. In: *XLIII Jornadas Argentinas de Informática e Investigación Operativa (43JAIIO)-XV Simposio Argentino de Ingeniería de Software (Buenos Aires, 2014)*. [S.l.: s.n.], 2014. Citado 4 vezes nas páginas 40, 41, 42 e 43.

MASCARDI, V. et al. A holonic multi-agent system for sketch, image and text interpretation in the rock art domain. *International Journal of Innovative Computing, Information and Control*, 2014. Citado 2 vezes nas páginas 40 e 42.

OMG. *OMG Unified Modeling Language*. 2015. Citado 4 vezes nas páginas 13, 26, 27 e 28.

OPENCV. *Introduction: Open CV*. 2015. <http://docs.opencv.org/modules/core/doc/intro.html>. Accessed: 2015-06-15. Citado 2 vezes nas páginas 36 e 37.

PEDRINI, H.; SCHWARTZ, W. R. *Análise de imagens digitais: princípios, algoritmos e aplicações*. [S.l.]: Thomson Learning, 2008. Citado 3 vezes nas páginas 28, 31 e 32.

PEREIRA, L. C. O.; SILVA, M. L. D. *Android para desenvolvedores*. [S.l.]: Brasport, 2009. Citado 2 vezes nas páginas 35 e 36.

PETERSEN, K. et al. Systematic mapping studies in software engineering. In: SN. *12th International Conference on Evaluation and Assessment in Software Engineering*. [S.l.], 2008. v. 17, n. 1. Citado na página 39.

SUN, Z. et al. Query-adaptive shape topic mining for hand-drawn sketch recognition. In: ACM. *Proceedings of the 20th ACM international conference on Multimedia*. [S.l.], 2012. p. 519–528. Citado 3 vezes nas páginas 40, 42 e 43.

TESSERACT. *Introduction: Tesseract*. 2015. <https://github.com/tesseract-ocr/tesseract>. Accessed: 2015-11-11. Citado na página 37.

TESSTWO. *tess-two*. 2015. <https://github.com/rmtheis/tess-two>. Accessed: 2015-11-11. Citado na página 37.

VASHISHT, V.; CHOUDHURY, T.; PRASAD, T. Sketch recognition using domain classification. *arXiv preprint arXiv:1211.2742*, 2012. Citado 3 vezes nas páginas 40, 41 e 43.

WÜEST, D.; SEYFF, N.; GLINZ, M. Flexisketch: A mobile sketching tool for software modeling. In: *Mobile Computing, Applications, and Services*. [S.l.]: Springer, 2013. p. 225–244. Citado 4 vezes nas páginas 21, 40, 41 e 44.