1
Introdução

1.1.
Contextualização

Os sistemas de software atuais são cada vez maiores em tamanho, em número de usuários e em complexidade estrutural. Estes sistemas funcionam sob uma incontável variedade de condições de operação. Para que sejam dignos de confiança, é necessário que eles provejam seus serviços como esperado por seus usuários mesmo sob condições excepcionais de operação (AZIVIENIS *et al.*, 2004). Condições excepcionais de operação podem emergir durante o fluxo de execução de um sistema de software em decorrência à manifestação de falhas. Falhas podem ser internas aos módulos do sistema, como também podem ser falhas no hardware ou sistema operacional subjacente, dentre outras. É necessário, portanto, que sistemas de software em geral sejam capazes de não apenas identificar a ocorrência destas condições excepcionais, mas também tomar medidas corretivas que os possibilitem prosseguir operando corretamente (ou, ao menos, de maneira aceitável).

Mecanismos de tratamento de exceções (GOODENOUGH, 1975) são atualmente os modelos mais comuns para lidar com a ocorrência de condições excepcionais em sistemas de software. Tais mecanismos têm como objetivo promover maior confiabilidade de software. Eles provêem suporte à detecção da ocorrência de condições excepcionais e à implementação de ações de recuperação. O objetivo dessas ações de recuperação é, depois de detectada a ocorrência de uma exceção, restaurar o sistema para um estado em que possa prover o seu serviço corretamente (ou de maneira aceitável).

Uma das vantagens dos mecanismos de tratamento de exceções é prover uma melhor separação entre a implementação da lógica relacionada ao provimento do serviço oferecido pelo sistema e a lógica responsável por recuperar o sistema para um estado correto, caso uma exceção ocorra durante a sua execução (GOODENOUGH, 1975; MCCLAREN, 1977). Advoga-se que a

PUC-Rio - Certificação Digital Nº 1012632/CA

separação entre comportamento normal e comportamento excepcional de um sistema permite aos desenvolvedores melhor estruturar e modificar as políticas de tratamento de exceções de um sistema (GARCIA *et al.*, 2001; GOODENOUGH, 1975; MCCLAREN, 1977; ROBILLARD e MURPHY, 2003; WIRFS-BROCK, 2006).

1.2. Problemática

Um dos objetivos dos mecanismos de tratamento de exceções é permitir que desenvolvedores de software implementem sistemas mais robustos e tolerantes a falhas. Para tanto, é necessário que os desenvolvedores estruturem ações de recuperação adequadas. Consideram-se aqui ações de recuperação adequadas aquelas capazes de restaurar um sistema de software para um estado correto após a ocorrência de uma exceção. Todavia, estudos recentes dão evidências de que os desenvolvedores não estruturam adequadamente o tratamento de exceções de seus sistemas em cenários reais de desenvolvimento de software (BARBOSA e GARCIA, 2010; BARBOSA e GARCIA, 2011; CABRAL e MARQUES, 2007; COELHO *et al.*, 2008; SHAH *et al.*, 2008a; SHAH *et al.*, 2010; WEIMER e NECULA; 2004). Estes estudos mostram que mesmo aplicações industriais apresentam código de tratamento de exceções de má qualidade. Em especial, percebe-se que as ações de recuperação implementadas por estas aplicações são ineficazes. Em alguns casos, de 40% a 72% das ações de recuperação em aplicações industriais são ações extremamente simplistas em que não há qualquer tentativa de restaurar o estado interno do sistema para um estado correto (CABRAL e MARQUES, 2007). Por exemplo, estas ações tendem a apenas imprimir a pilha de execução associada à exceção, ou até mesmo não fazer nada. Há ainda casos em que falhas recorrentes e problemas de desempenho em sistemas de software estão relacionados à má implementação do tratamento de exceções de um sistema (COELHO *et al.*, 2008; WEIMER e NECULA; 2004). São exemplos de tratadores mal implementados: tratadores declarando tipos de exceção muito genéricos que capturam inadvertidamente exceções indesejadas em locais indevidos (COELHO *et al.*, 2008) e tratadores que não desalocam recursos previamente alocados (WEIMER e NECULA; 2004).

PUC-Rio - Certificação Digital Nº 1012632/CA

O tratamento de exceções é considerado por muitos desenvolvedores como uma tarefa central no processo de desenvolvimento de software (SHAH *et al.*, 2010). Na prática, entretanto, muitos desenvolvedores acabam concentrando a maior parte dos seus esforços na implementação do comportamento normal de um sistema (SHAH *et al.*, 2010). Também é prática comum entre os desenvolvedores postergar a implementação do tratamento de exceções apenas para as versões futuras do sistema (SHAH *et al.*, 2010). Além disso, o código que implementa o comportamento excepcional de um sistema ainda é uma das partes menos compreendidas pelos desenvolvedores (ROBILLARD e MURPHY, 2003). Percebe-se também que desenvolvedores de software em geral têm dificuldade e até relutam em implementar qualquer código de tratamento de exceções (SHAH *et al.*, 2010).

1.3. Limitação do estado da arte e estudos preliminares

A fim de auxiliar equipes de desenvolvimento a lidar com os elementos de mecanismos de tratamento de exceções, algumas ferramentas de apoio têm sido propostas. A maioria destes trabalhos concentra-se na disponibilização de informações sobre a estrutura estática do código de tratamento de exceções. Em especial, são disponibilizadas informações a respeito do fluxo de propagação das exceções. Por fluxo de propagação entende-se (i) o local em que a exceção é lançada, (ii) os locais por onde ela é propagada e o (iii) local em que é tratada, além do tipo da exceção. Tais informações são providas através de ferramentas de análise estática (ROBILLARD e MURPHY, 2003; FU e RYDER, 2007), de medição (GARCIA e CACHO, 2011) ou de visualização (SHAH *et al.*, 2008b). Estas ferramentas seguem um mesmo paradigma: um *back-end* baseado em um mecanismo de análise estática que coleta informações do código fonte e um *front-end* que apresenta estas informações de maneira textual ou visual. Tais ferramentas auxiliam os desenvolvedores a compreender parte dos módulos de um sistema que estão relacionados com a implementação do tratamento de exceções. Também permitem obter uma visão mais global do relacionamento destes módulos com os demais módulos do sistema.

PUC-Rio - Certificação Digital Nº 1012632/CA

Na prática, estas ferramentas não são muito úteis, tendo em vista os problemas relacionados à qualidade do código implementado para tratamento de exceções (Seção 1.2). Tais ferramentas produzem uma vasta quantidade de informações a respeito de fluxos de propagação de exceções. Parte desta informação é inútil ao desenvolvedor, pois são fluxos de propagação que não ocorrem na prática. Além disso, informações a respeito do fluxo de propagação das exceções não é, necessariamente, a informação que o desenvolvedor deseja na prática (Shah *et al.*, 2010). Essas ferramentas falham, sobretudo, ao assumir que o código de tratamento de exceções já está minimamente estruturado e necessita apenas de manutenções periféricas. Estudos anteriores (CABRAL e MARQUES, 2007; COELHO *et al.*, 2008; SHAH *et al.*, 2008a; SHAH *et al.*, 2010; WEIMER e NECULA; 2004) e os estudos preliminares realizados no contexto desta dissertação (Seção 2.3) sugerem que esta suposição é frágil. Em particular, nenhuma destas ferramentas auxilia o desenvolvedor em uma tarefa mais básica, mas não menos importante: implementar código de tratamento de exceções de maneira bem estruturada desde o princípio.

Há também uma lacuna na literatura em termos de trabalhos que ajudam desenvolvedores de software a escrever código de tratamento de exceções. Em especial, há poucos trabalhos que discutem e propõem conjuntos de práticas e padrões de implementação dos elementos de tratamento de exceção. Em (WIRFS-BROCK, 2006), um conjunto de diretrizes básicas para o projeto de tratamento de exceções em sistemas de software orientado a objetos é apresentado. O repositório *Portland Pattern Repository* (PORTLAND PATTERN REPOSITORY, 2012) apresenta um conjunto de descrições de padrões de estruturação para tratamento de exceções. Entretanto, todos esses padrões e diretrizes são descritos em um nível de abstração alto, que muitas vezes não é facilmente mapeado para o nível de implementação, especialmente por desenvolvedores menos experientes.

Nos estudos preliminares conduzidos no contexto desta dissertação (BARBOSA e GARCIA, 2010; BARBOSA e GARCIA, 2011), avaliou-se a evolução histórica de sistemas de software durante períodos de no mínimo 2 anos. As análises corroboram parcialmente os resultados de estudos recentes (CABRAL e MARQUES, 2007; COELHO *et al.*, 2008; SHAH *et al.*, 2008a; SHAH *et al.*, 2010; WEIMER e NECULA; 2004). Percebe-se que, de fato, é comum encontrar nas primeiras versões de um sistema código de tratamento de exceções de baixa

PUC-Rio - Certificação Digital Nº 1012632/CA

qualidade. Nas versões preliminares dos sistemas analisados, foi bastante comum encontrar: (i) interfaces excepcionais e tratadores definidos com tipos excessivamente abstratos; (ii) tratadores vazios ou ineficazes (como por exemplo, apenas imprimir a pilha de execução associada à exceção); e (iii) tipos de exceções sendo sobrecarregados semanticamente, isto é, um mesmo tipo de exceção sendo levantado em diferentes pontos do sistema, mas representando condições excepcionais distintas. Isto confirmou a expectativa de que menor atenção é dada a tratamento de exceções em versões iniciais de um sistema. E isto ocorreu mesmo em sistemas com atuação de desenvolvedores experientes.

Além disso, observou-se uma série de fenômenos interessantes ao longo do histórico dos sistemas analisados. Percebeu-se que quando as funcionalidades principais dos sistemas atingiam certa maturidade após algumas versões, os desenvolvedores dedicavam maior esforço em tentar melhorar o código de tratamento de exceções. Em geral, os desenvolvedores dos sistemas analisados tentaram apenas definir interfaces excepcionais e tratadores com tipos de exceções mais específicos. Mas é importante ressaltar que pouco (ou quase nenhum) esforço foi alocado em termos de melhorar a qualidade do código dos tratadores. Nem mesmo durante períodos de evolução de 2 a 3 anos foram verificados tais esforços. Durante o processo de evolução de um sistema de software, algumas mudanças e melhorias pretendidas, ou necessárias, no código de tratamento de exceções podem tornar-se custosas ou impeditivas em versões futuras do sistema. Um projeto de tratamento de exceções inadequado pode levar a erosões arquiteturais em projetos de sistemas de software (MACIA et al., 2012). Em casos extremos, sintomas de erosão arquitetural podem levar até mesmo a instabilidades na arquitetura de software de um sistema. Em um dos sistemas analisados (BARBOSA e GARCIA, 2011), verificou-se que o projeto do tratamento de exceções implementado nas primeiras versões violava algumas regras do projeto arquitetural planejado. Para corrigir estas violações arquiteturais, nas versões subseqüentes alguns módulos foram removidos por completo do projeto do sistema.

A implementação do tratamento de exceções é uma tarefa difícil de ser cumprida, mesmo por desenvolvedores mais experientes. O tratamento de exceções é questão central no desenvolvimento de qualquer sistema de software.

PUC-Rio - Certificação Digital Nº 1012632/CA

Portanto, é importante devotar atenção à sua implementação adequada desde as primeiras versões de um sistema, e não apenas em versões posteriores.

1.4. Programando tratamento de exceções: um cenário motivador

Durante o processo de implementação, comumente um desenvolvedor se depara com uma exceção; seja uma exceção lançada por seu próprio código, ou, mais comumente, uma exceção lançada por uma API de terceiros usada em sua aplicação. Quando se deparam com uma exceção, também é comum os desenvolvedores não terem certeza de como deve ser realizado o tratamento desta exceção. Na maioria das vezes, os desenvolvedores estão bem mais focados na implementação da funcionalidade principal e acabam ignorando o tratamento da exceção. Nestes casos, é comum que recorram a algum recurso do ambiente de desenvolvimento integrado para "resolver o problema" rapidamente. A Figura 1 mostra um destes recursos providos pelo ambiente de desenvolvimento Eclipse[1].

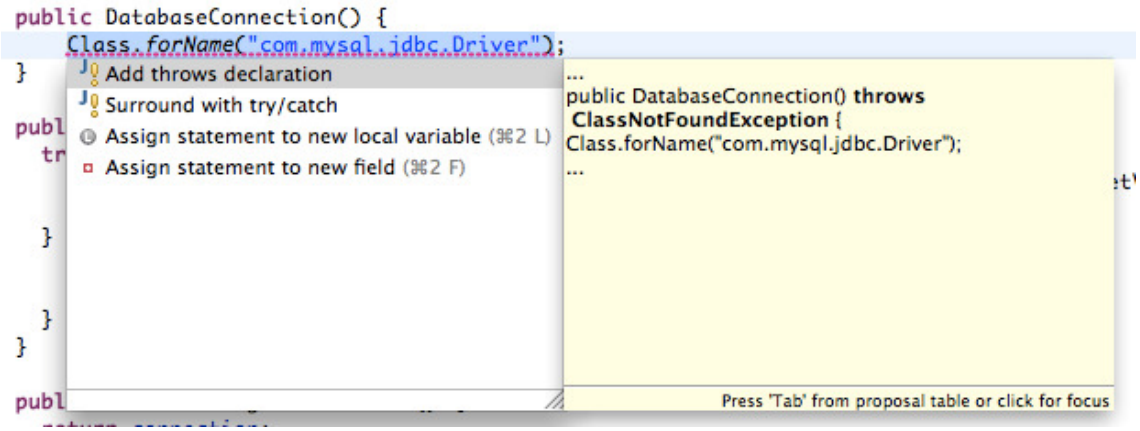

Figura 1 Sugestão para tratamento de exceções

Na Figura 1, é mostrado que o ambiente de desenvolvimento Eclipse ao detectar que há uma exceção não tratada no código em desenvolvimento exibe a seus usuários um alerta de erro. Além disso, o Eclipse também oferece recomendações para remediar a situação de erro. Como mostra a Figura 1, o ambiente Eclipse apresenta a seus usuários a possibilidade de indicar na assinatura do método sendo implementado (método *DatabaseConnection* da Figura 1) a ocorrência desta exceção, ou cercar o método que está lançando a exceção

[1] http://www.eclipse.org/

PUC-Rio - Certificação Digital Nº 1012632/CA

(método *Class.forName* da Figura 1) com um bloco tratador. Fica a critério do usuário qual opção será escolhida. A Figura 2 mostra o código gerado pelo Eclipse caso a segunda opção seja escolhida pelo usuário.

```
public DatabaseConnection() {
    try
  {
      Class.forName("com.mysql.jdbc.Driver");
  }
  catch (ClassNotFoundException e)
  {
      // TODO Auto-generated catch block
      e.printStackTrace();
  }
}
```

Figura 2 Código de tratamento de exceções gerado automaticamente pelo Eclipse

Embora a recomendação de código seja um recurso comumente usado por desenvolvedores, é notório que mesmo os ambientes de desenvolvimento mais populares, como o Eclipse, ainda não oferecem suporte satisfatório à recomendação de código de tratamento de exceções. A Figura 2 mostra que o esqueleto de código gerado pelo Eclipse apenas imprime no console do sistema a pilha de execução associada à exceção capturada. O tratador gerado é de má qualidade, indo contra as boas práticas de tratamento de exceções (MCCUNE, 2006). Além disso, é ineficaz, visto que nem implementa uma medida corretiva que restaure o estado interno do sistema para um estado correto, nem reporta ao módulo cliente a ocorrência da exceção.

O problema mencionado acima leva a uma série de outras consequências negativas a curto, médio e longo prazo. Por exemplo, um desenvolvedor mais inexperiente pode não ser capaz de julgar a qualidade do tratador sugerido e, por falta de conhecimento (ou por confiar em seu ambiente de desenvolvimento), acatar a sugestão como adequada. Este desenvolvedor inexperiente pode "aprender" com esta sugestão do Eclipse, considerando equivocadamente tal tratador como adequado e passar a repetir o código em outros cenários. Outros desenvolvedores podem utilizar a recomendação de código como um recurso para imediatamente silenciar o erro gerado pelo ambiente de desenvolvimento, e só posteriormente reestruturar o código gerado, de modo que o tratamento de exceções seja realizado adequadamente. No entanto, pode ocorrer destes desenvolvedores esquecerem, negligenciarem, ou até mesmo serem impedidos por

PUC-Rio - Certificação Digital Nº 1012632/CA

escassez de tempo, de retornar ao ponto em que o código foi gerado a fim de melhorar a sua qualidade. As recomendações de código de tratamento de exceções (como no caso da Figura 2) realizadas por ambientes de desenvolvimento populares são geralmente bastante simplistas. Isto talvez explique porque alguns estudos anteriores indicam que a qualidade do código de tratamento de exceções é ruim e porque muitos desenvolvedores estão usando os mecanismos de tratamento de exceções apenas como uma ferramenta de auxílio a tarefas de diagnóstico e auxílio a manutenção de falhas (SHAH et al, 2010).

No estudo de Shah et al. (2010), os autores realizaram entrevistas com desenvolvedores. O objetivo era identificar quais práticas os desenvolvedores mais experientes adotam a fim de adquirir um maior conhecimento a respeito de tratamento de exceções. Um dos resultados do estudo apontou para uma situação em que os desenvolvedores dão maior atenção ao tratamento de exceções (SHAH et al, 2010, p. 3):

> A única exceção ao comportamento (dos desenvolvedores) de ignorar o tratamento de exceções ocorre em cenários em que o código no qual eles estavam trabalhando já continha algum código de tratamento de exceções previamente implementado. Nestes cenários, os (desenvolvedores) participantes confirmaram que tentam imitar o código já existente. Portanto, em geral, os participantes evitam o tratamento de exceção, exceto quando alguma estrutura de suporte já está disponível.

Além disso, os desenvolvedores experientes também ressaltaram a importância de estar constantemente revisando e observando código de outros desenvolvedores. Um dos participantes afirmou (SHAH et al, 2010, p. 7):

> "Eu sempre olho o código de softwares desenvolvido por outras pessoas e a comunidade de software aberto é ótima para isso, para encontrar softwares que realizam tratamento de exceções de maneira que nunca fiz, ou nunca vi."

No contexto desta dissertação, o ambiente de desenvolvimento ideal seria aquele capaz de recomendar trechos de código de tratamento de exceções que pudessem ser reutilizados integralmente e sem necessidade de alteração por parte do desenvolvedor. Sabe-se que este é, na prática, um ambiente de desenvolvimento idealizado e provavelmente infactível. No entanto, um ambiente de desenvolvimento um pouco mais refinado, capaz de recomendar trechos de código mais próximos ao contexto de desenvolvimento do programador já seria de grande valia, como apontado por alguns desenvolvedores do estudo de Shah et al. (2010).

PUC-Rio - Certificação Digital Nº 1012632/CA

1.5. Objetivos e visão geral da solução

O objetivo inicial deste trabalho foi analisar e compreender a qualidade do código de tratamento de exceções produzido atualmente por desenvolvedores de software. Para tanto, foram realizadas inspeções manuais do código de tratamento de exceções de aplicações reais utilizadas pela indústria. As inspeções compreenderam versões distintas de um mesmo sistema em períodos de evolução variando de 2 a 3 anos. Ao todo, 24 versões distintas de 3 sistemas diferentes foram inspecionadas. Ao término desta inspeção, alguns fenômenos interessantes foram observados. Em especial, observou-se que nas versões preliminares dos sistemas comumente implementaram-se tratadores ineficazes. Em alguns casos, cerca de 50% dos tratadores eram vazios. Observou-se ainda que, durante os períodos de evolução avaliados, poucos esforços foram alocados na melhoria da qualidade dos tratadores. Na prática, verificaram-se apenas tentativas no sentido de tornar mais específicas as exceções que são declaradas pelos tratadores. Pouco, ou nenhum, esforço em termos de tornar as ações de recuperação mais apropriadas foi verificado. A conclusão destas análises mostra que desenvolvedores em geral têm dificuldade não apenas em implementar tratadores adequados para exceções, mas também de melhorar durante a evolução do sistema os tratadores de má qualidade previamente implementados. Desta forma, desenvolvedores deveriam ser apoiados por ferramentas que ajudassem a identificar ações adequadas de tratamento de exceções no desenvolvimento, preferencialmente, e ao longo da evolução de sistemas.

Neste contexto, o segundo objetivo deste trabalho foi propor, implementar e avaliar um sistema de recomendação capaz de auxiliar desenvolvedores a implementar o comportamento excepcional de seus sistemas através do provimento de exemplos implementando código de tratamento de exceções. Não é objetivo do sistema de recomendação proposto prover exemplos de código para serem reusados *ipsis litteris*. O objetivo do sistema de recomendação é prover exemplos concretos de código implementando tratamento de exceções de modo a auxiliar desenvolvedores no processo de aprendizado de como tratar suas exceções mais adequadamente. Os usuários do sistema de recomendação proposto

PUC-Rio - Certificação Digital Nº 1012632/CA

poderão se beneficiar de seus serviços tanto em tarefas de implementação de novos tratadores, quanto em tarefas de melhoria de tratadores previamente implementados.

Para realizar o sistema de recomendação proposto estratégias heurísticas foram propostas e implementadas para buscar exemplos a serem recomendados ao desenvolvedor que solicitou as recomendações. As estratégias heurísticas baseiam suas buscas em um conjunto de informações estruturais extraídas do código em que o desenvolvedor está trabalhando. Com isso, as heurísticas tentam buscar por exemplos que compartilhem similaridades estruturais com o código sob implementação. Desta forma, tenta-se maximizar a chance de realizar recomendações que sejam relevantes ao desenvolvedor no contexto do código em que ele está trabalhando. Definiram-se ainda pesos associados a cada uma das heurísticas de modo a permitir que os usuários tenham a possibilidade de refinar as recomendações realizadas pelas heurísticas através do ajuste destes pesos.

O sistema de recomendação proposto foi implementado em Java. Sua arquitetura base é composta por três módulos: (i) Módulo Extrator de Informações, o qual extrai de projetos de software os fragmentos de código armazenados no repositório de exemplos e as suas respectivas informações estruturais; (ii) Módulo Gerenciador de Repositório, o qual cria e manipula o repositório de exemplos; e o (iii) Módulo Recomendador, o qual busca por candidatos no repositório de exemplos e os ranqueia de acordo com a sua relevância em relação ao código em que o desenvolvedor está trabalhando.

Por fim, uma série de avaliações foram realizadas. Para a realização das avaliações, foi construído um repositório de exemplos de código implementando tratamento de exceções. Este repositório foi construído com fragmentos de código extraídos de projetos de software de código aberto disponíveis em repositórios públicos disponíveis na Internet. As avaliações foram divididas em dois grupos: avaliações preliminares, realizadas ainda durante o processo de desenvolvimento do sistema de recomendação, e avaliações de relevância, realizadas após concluída a implementação do sistema de recomendação. Além disso, comparações com trabalhos relacionados também foram realizadas.

A primeira avaliação preliminar avaliou se as informações estruturais usadas pelas heurísticas são capazes de identificar candidatos no repositório que compartilhe similaridades estruturais com o código em que o desenvolvedor está

PUC-Rio - Certificação Digital Nº 1012632/CA

trabalhando. A segunda avaliação preliminar avaliou se os ajustes dos pesos permite identificar com maior precisão no repositório candidatos compartilhando similaridades estruturais com o código em que o desenvolvedor está trabalhando.

A primeira avaliação de relevância avaliou se as heurísticas do sistema de recomendação para código de tratamento de exceções são capazes de recomendar exemplos de código contendo informações relevantes em tarefas de implementação de tratamento de exceções. A segunda avaliação avaliou qual a influência do tamanho e da proveniência das informações estruturais usadas pelas heurísticas sobre a relevância das informações recomendadas.

1.6. Estrutura da dissertação

O restante deste documento está estruturado da seguinte maneira. O Capítulo 2 apresenta um breve referencial teórico a respeito dos temas relacionados ao domínio desta dissertação. Além disso, no Capítulo 2 ainda apresentam-se os resultados principais dos estudos exploratórios preliminares realizados no contexto desta dissertação. No Capítulo 3 é apresentado o sistema de recomendação para código de tratamento de exceções proposto nesta dissertação. Nesse capítulo, são apresentados e discutidos todos os conceitos e mecanismos que compõem o sistema de recomendação proposto. No Capítulo 4 mostra-se como os conceitos e mecanismos do sistema de recomendação para código de tratamento de exceções do Capítulo 3 foram implementados em Java. Mostra-se a arquitetura de software do sistema implementado, bem como alguns detalhes de implementação dos módulos dessa arquitetura. No Capítulo 5 discutem-se as avaliações realizadas para medir a precisão e a relevância dos exemplos de código recomendados pelo sistema de recomendação proposto. Por fim, o Capítulo 6 conclui esta dissertação, apresentando as principais contribuições realizadas e discutindo algumas possiblidades e direcionamentos de pesquisa futura.

PUC-Rio - Certificação Digital Nº 1012632/CA