Aluno: Luis Gomes Damasceno Neto
Curso: SI
Matrícula: ******

Resumo do artigo

RAID : Suporte de ferramenta para revisões de código com reconhecimento de refatoração.

Introdução

A revisão de código é uma prática de engenharia de software amplamente utilizada, originada na década de 70, quando era realizada de acordo com regras formais e rígidas. Ao longo dos anos, práticas leves de revisão de código surgiram e ganharam popularidade, a fim de tornar o processo mais ágil. No entanto, a revisão de código leva tempo e pode causar atrasos no lançamento de novas versões. A razão é que é um processo manual que requer experiência na base de código e inspeção cuidadosa das diferenças textuais.

Particularmente, as ferramentas de comparação atuais não detectam automaticamente refatorações na alteração do código inspecionado. Após inferir a refatoração, os revisores também devem comparar o código após e antes da alteração, para revisar possíveis alterações realizadas na função movida. A razão é que as refatorações geralmente são seguidas por pequenas edições no código, principalmente quando comparamos as alterações no nível de commits.

Este artigo descreve os principais recursos e a arquitetura de uma ferramenta para dar suporte a revisões de código com reconhecimento de refatoração. Com o objetivo de aliviar o esforço cognitivo associado às revisões de código, detectando automaticamente as operações de refatoração incluídas nas solicitações pull. Além de oferecer suporte à detecção de refatoração, a ferramenta RAID (Refactoring ciente e Intelligent Diffs) instrumenta perfeitamente as ferramentas de comparação atuais com informações sobre refatorações. E então os revisores podem facilmente inspecionar as alterações realizadas no código de refatoração após a operação. Em relação à sua arquitetura, o RAID é construído sobre ferramentas de refatoração recentes e automáticas.
Este artigo está dividido em três seções principais:

1. Na Seção II, apresentamos os principais recursos do RAID e a interface baseada na Web.

2. Na Seção III, descrevemos a arquitetura interna do RAID, incluindo sua integração com ferramentas de terceiros, como GitHub (Ações e pull requests) e navegadores (Chrome).

3. Na Seção IV, documentamos os resultados e as lições aprendidas em um experimento de campo com oito desenvolvedores profissionais que usaram RAID

durante três meses. Concluímos que o RAID pode de fato reduzir o esforço cognitivo necessário para detectar e revisar refatorações.

Raid in a nutshell

RAID é uma ferramenta para instrumentar diferenças textuais (particularmente, aquelas fornecidas pelo GitHub) com informações sobre refatorações, os revisores de código devem inferir por si mesmos se essa “diferença” representa uma operação de refatoração, que requer uma quantidade de esforço cognitivo. Essencialmente, o RAID pré-processa o diff padrão - fornecido pelo GitHub - de maneira contínua e gera uma visualização de diff aprimorada com informações de refatoração explícitas.

Primeiramente, na parte superior, temos acesso as seguintes informações sobre a operação: nome ,breve descrição, origem e arquivo de destino . Também podemos temos acesso a visão lado a lado do código antes e depois da operação de movimentação. Por fim, neste diff interno, as linhas alteradas no código movido são destacadas, ou seja, os revisores podem revisá-las facilmente. Pelas razões acima mencionadas, afirmamos que o RAID reduz o esforço cognitivo necessário para revisar refatorações.

Todas as janelas fornecidas pelo RAID incluem um botão “Go to source”, que permite ao revisor localizar a fonte da refatoração realizada, reduzindo assim o esforço para encontrar o código no diff padrão fornecido pelo GitHub. Finalmente, o RAID instrumenta a barra de ferramentas do diff padrão com um botão que fornece uma lista com todas as refatorações detectadas em um pull request.

Arquitetura RAID

A arquitetura RAID é composta por 3 componentes :

1. RAID GitHub Action(RGA), responsável por detecção e extração de refatorações.

2. RAID Chrome Extension(RCE) é uma extensão do navegador que instrumenta as páginas de comparação padrão do GitHub com informação de refatorações.

3. RAID server, que fornece uma API REST para armazenar metadados de refatoração e conectar os aplicativos mencionados anteriormente.

RGA : O componente RAID GitHub Action é responsável por analisar o código-fonte enviado pelos desenvolvedores como parte de um pull request. O RGA é implementado em Java e é executado em um container Docker. A principal função do RGA é detectar as refatorações realizadas em um pull request.

RCE: O RAID Chrome Extension (RCE) apresenta os dados de refatoração fornecidos pelo servidor RAID. O RCE é implementado em JavaScript e atualmente suporta o navegador Google Chrome A extensão é chamada automaticamente pelo Chrome depois que um URL de solicitação de pull do GitHub válido é aberto pelo navegador. Essa URL inclui

informações sobre a solicitação pull, que é usada pelo RCE para entrar em contato com o servidor RAID e obter dados sobre possíveis refatorações.

Servidor RAID: O servidor RAID atua como uma ponte entre o RCA e componentes RCE, ou seja, recebe as informações de refatoração após cada execução do RGA e envia as refatorações para o RCE quando solicitado. Para projetos de código aberto o RAID fornece uma API pública, e para projetos privados, o servidor fornece configurações personalizadas para controlar o acesso do usuário.

Experimento de Campo

1. Metodologia

Para avaliar o RAID, foi realizado um experimento de campo. Mais especificamente, obtiveram permissão para incluir a ferramenta no fluxo de trabalho de desenvolvimento de uma empresa de tecnologia de médio porte que desenvolve software para produtos musicais.

Antes de iniciar o experimento, os participantes receberam instruções e treinamento em nossa ferramenta (~30 minutos), seguidos por uma semana de aquecimento e para se acostumar com o RAID.

Durante o experimento, foram criadas 325 solicitações pull. Após o experimento, enviamos uma breve pesquisa aos participantes perguntando suas percepções sobre o RAID. Neste levantamento, nós fez três perguntas:

• Quais são os principais benefícios do RAID?

• Quais são as dificuldades que você enfrentou ao usar o RAID?

• Algum comentário ou sugestão adicional?

2. Questões de pesquisa

Usamos os dados coletados no experimento de campo para responder quatro perguntas de pesquisa:

RQ1: Qual é a sobrecarga de tempo de execução introduzida pelo RAID?

Começamos com essa pergunta, pois é importante verificar se o RAID introduz (ou não) um atraso no processo de revisão de código(é importante que as informações fornecidas pelo RAID sejam disponibilizadas aos revisores de código o mais rápido possível).

RQ2: Como os desenvolvedores usaram o RAID durante a revisão do código?

Por exemplo, nos três meses de uso do RAID, foi relatado e analisado os dados de uso coletados, incluindo refatorações mais frequentes com eventos de usuário, tempo de

revisão e eventos de interface do usuário realizados por usuários na interface baseada na web do RAID.

RQ3: Quanto esforço cognitivo é reduzido com RAID?

Embora este seja o objetivo principal do RAID, não é trivial estimar a quantidade de esforço cognitivo que é economizado quando o RAID é usado para suportar tarefas de revisão de código, particularmente no caso de um experimento realizado no contexto de uma empresa de software real.

RQ4: Como os desenvolvedores perceberam o RAID?

Nesta última questão de pesquisa, relatamos as percepções dos participantes sobre nossa ferramenta, comentadas por eles na pesquisa pós-experimento.

3. Resultados da pesquisa

RQ1: Qual é a sobrecarga de tempo de execução introduzida pelo RAID?

Normalmente, as informações fornecidas pelo RAID estão disponíveis para revisores de código em menos de 2 minutos após o envio da solicitação pull. Portanto, em termos práticos, o RAID não atrasa o início das revisões de código.

RQ2: Como os desenvolvedores usaram o RAID durante a revisão do código?

Em termos relativos, os revisores confiaram no RAID para obter informações principalmente para Move Struct, Move and Rename Function e Move Function; (2) os revisores usaram os dois botões “R”— disponíveis no lado esquerdo e direito do diff original—para solicitar informações de refatoração fornecidas pelo RAID; (3) na mediana, os revisores gastaram de 2,5 segundos (Função Renomear) a 12,4 segundos (Função Inline) para interpretar as informações de refatoração fornecidas pelo RAID.

RQ3: Quanto esforço cognitivo é reduzido com RAID?

71,8% das refatorações da Função Mover são entre arquivos, quando o RAID supera claramente as diferenças com reconhecimento de não refatoração; (2) No caso de movimentos intra-arquivo, o código é movido 41 linhas após ou abaixo de sua posição original (resultados medianos); (3) Diff Code Churn reduz de 14,5 linhas para apenas 2 linhas (resultados médios), quando os movimentos são revisados usando RAID.

Os métodos são extraídos 25 linhas antes ou depois do método fonte; (2) Diff Code Churn diminui de 113 linhas para 55 linhas ao usar RAID para revisar refatorações de função de extração.

RQ4: Como os desenvolvedores perceberam o RAID?

Benefícios do RAID. Todos os participantes mencionaram que o RAID representa uma melhoria em relação ao diff tradicional do GitHub. Como exemplos, recebemos as seguintes respostas.

O RAID destaca grandes migrações de código, facilitando a revisão de código quando o trecho acabou de ser movido sem grandes alterações. O RAID também permite uma revisão de código mais eficiente com um diferencial realista, que mostra as mudanças reais que ocorreram no código, a ferramenta torna o processo de reconhecimento de refatoração instantâneo, e mostra um diff mais direcionado e limpo, facilitando e acelerando o entendimento do código.

Foi recebido cinco respostas apontando limitações no RAID. Por exemplo, um revisor relatou que o RAID não identificou refatorações em casos específicos. Eu particularmente experimentei alguns bugs em que o RAID não detectou possíveis refatorações.

Dois revisores relataram a limitação do suporte ao navegador. Atualmente, o RAID suporta apenas o Google Chrome. Como trabalho futuro, pretendemos estender a ferramenta para outros navegadores, como Safari e Mozilla Firefox.

Ameaças à validade

Validade externa. Em nosso experimento de campo, avaliamos quatro projetos de linguagem Go de uma empresa de médio porte. Os participantes usaram o RAID durante três meses. Portanto, por um lado, nos esforçamos para realizar um experimento real, com sistemas do mundo real e desenvolvedores profissionais. Por outro lado, reconhecemos que nossos resultados podem mudar se considerarmos mais projetos, mais desenvolvedores, um time box maior e outras linguagens de programação. Devido à sua importância, a literatura apresenta diversos estudos explorando problemas relacionados à refatoração. Por exemplo, existem estudos sobre motivações para realizar operações de refatoração , , desafios e benefícios , o impacto na qualidade do software, segurança , e também em ferramentas para auxiliar os desenvolvedores durante as tarefas de refatoração . Além disso, vários estudos focaram na identificação de operações de refatoração por meio da análise de sistemas de controle de versão .

Validade do construto. Primeiro, o módulo de detecção de refatoração conta com RefDiff , que tem precisão de 92% e recall de 80%, no caso de projetos Go [10]. Dessa forma, as visualizações RAID podem perder refatorações (falsos negativos) ou podem detectar refatorações incorretamente (falsos positivos). De fato, a presença de falsos negativos foi mencionada por um dos participantes do experimento. No entanto, destacamos que a precisão de RefDiff é compatível com o estado da arte em ferramentas de detecção de refatoração [9], [11]. Além disso, para mitigar o impacto dessa ameaça nos resultados dos experimentos de campo, o primeiro autor deste artigo avaliou manualmente cada uma das refatorações utilizadas para calcular as métricas de proxy apresentadas e discutidas ao responder ao RQ3. No total, encontramos 10 falsos positivos (13,2%) entre jogadas e extratos.

Validade interna. Uma primeira possível ameaça interna à validade pode ser encontrada no RQ1, onde o tamanho das solicitações pull (ou seja, o número de linhas modificadas) pode

influenciar o tempo de execução. Em nossa terceira questão de pesquisa, propusemos duas proxies para estimar a quantidade de esforço cognitivo reduzido pelo RAID: Diff Code Churn (DCC) e refatorações de distância de movimento. Por um lado, afirmam que tais proxies têm uma correlação positiva com o esforço cognitivo exigido nas revisões de código. Por outro lado, não nos permitiram fornecer uma medida clara da redução do esforço. Portanto, mais estudos - como os controlados - podem contribuir para esclarecer o esforço economizado com o RAID. O tempo de inspeção do flutuador janela também é uma ameaça à validade, pois um desenvolvedor pode ter deixado a janela aberta além do tempo usado na revisão.

Trabalho Relacionado

Devido à sua importância, a literatura apresenta diversos estudos explorando problemas relacionados à refatoração. Por exemplo, existem estudos sobre motivações para realizar operações de refatoração, desafios e benefícios , o impacto na qualidade do software, segurança e também em ferramentas para auxiliar os desenvolvedores nas tarefas de refatoração. Além disso, varios estudos focaram na identificação e operações de refatoração por meio da análise de sistemas de controle de versão. Por exemplo, recentemente, Tsantalis e os outros, apresentaram o Refactoring Miner que identifica 40 tipos distintos de refatoração em Java.

Silva e os outros, propuseram uma ferramenta multilíngue para detectar atividades de refatoração. A última versão identifica 13 tipos distintos de refatoração e suporta três linguagens de programação (ou seja, Java, JavaScript e C).

As diferenças textuais também apresentam problemas ao mesclar alterações nos sistemas de controle de versão. Como, Bo e os outros, que propuseram um algoritmo com reconhecimento de refatoração baseado em grafos para detectar e resolver conflitos relacionados à refatoração.

Os benefícios das revisões de código com reconhecimento de refatoração também foram explorados na literatura.

Por exemplo, Ge e os outros, apresentaram um estudo formativo com 35 desenvolvedores para investigar a motivação e os desafios de revisar refatorações durante a revisão de código. Os autores também apresentaram uma ferramenta com reconhecimento de refatoração chamada ReviewFactor, que fornece uma separação de alterações de refatoração e não refatoração para fins de revisão de código.

O RAID é uma ferramenta com reconhecimento de refatoração que instrumenta o GitHub diff com informações de refatoração. Também foi realizado um experimento de campo com oito desenvolvedores profissionais durante três meses e concluímos que o RAID pode reduzir o esforço cognitivo necessário para revisar refatorações ao usar diferenças textuais. Além disso, nosso estudo relata uma redução no número de linhas necessárias para a revisão de tais operações.