PROTOCOLO DE ANÁLISE MANUAL PARA DEFINIÇÃO DE *GROUND TRUTH*

Para definição do *ground truth* deve ser realizada uma checagem dupla:

- Para cada cenários uma dupla é alocada para realizar a análise manual seguindo as instruções abaixo.
- A análise manual deve ser feita de forma individual pelos colaboradores alocados no cenário.
- A análise individual dos dois colaboradores comparada
    - Se convergirem, o resultado deve ser adotado com *ground truth* do cenário;
    - Se divergir, os colaboradores devem se reunir e discutir o cenário juntos buscando chegar a um consenso.
        - Se não for possível chegar a um consenso, o cenário deve ser apresentado a outro colaborador que tomará a decisão.

INSTRUÇÕES PARA REALIZAÇÃO DA ANÁLISE MANUAL PARA DEFINIÇÃO DE *GROUND TRUTH*

O conjunto de dados consiste em um número de assuntos ou casos a serem analisados. Cada assunto é um par formado por um cenário de *merge* (tupla com *base*, *left*, *right* e *merge commits*) e uma declaração de método ou campo (sim, não focamos apenas em métodos) que foi alterado independentemente por dois *branches* (ou dois desenvolvedores). Um cenário que contém apenas uma única declaração alterada por duas ramificações leva a um único assunto (e, consequentemente, a uma única linha na planilha do conjunto de dados). Um cenário s que contém n declarações alteradas por duas ramificações leva a n assuntos: (s,*declaration* 1), …, (s,*declaration* n).

Para cada assunto(s,d) no conjunto de dados, a análise manual deve seguir estas etapas:

1. Verifique se você tem a versão mais recente do conjunto de dados (https://github.com/spgroup/mergedataset) em sua máquina.
2. Use o kdiff (http://kdiff3.sourceforge.net) para entender as mudanças feitas na base nos ramos esquerdo e direito.
    a. Basta carregar os *commits base*, esquerdo e direito de s no kdiff;
    b. Procure a declaração d alterada em ambas as ramificações;
    c. Concentre a análise nesta declaração;
    d. Consideramos o ramo esquerdo como a sequência (caminho) de *commits* do *commit* esquerdo até a base; da mesma forma para o ramo direito.
3. Faça uma ou mais capturas de tela que revelem as alterações na declaração sendo o foco da etapa anterior e como elas são integradas ao *commit* de mesclagem.

a. Adicione as capturas de tela a esta apresentação (https://docs.google.com/presentation/d/1_IhJulqfsj4OF8pGrTEEtbVwhtP9SFXldDDPtzYYyhg/edit?usp=sharing);
b. Considerando a ordem em que o cenário de mesclagem aparece na planilha do conjunto de dados, adicione um *slide* com o *hash* de confirmação de mesclagem;
c. Logo em seguida, adicione um ou mais *slides* com as capturas de tela do diff entre esquerda-base-direita (esta ordem é importante);
d. Depois disso, adicione uma captura de tela do *commit* de mesclagem com as anotações "// *left*" e "// *right*" no final das linhas alteradas pelas ramificações esquerda e direita, respectivamente;
    i. Se uma linha for alterada pela esquerda e pela direita, adicione "// *left and right*";
e. O foco das capturas de tela deve ser revelar as alterações na declaração, mas se a declaração for curta o suficiente, as capturas de tela devem mostrar toda a declaração
f. Se as alterações à esquerda e à direita levarem a um conflito de mesclagem
    i. Adicione uma nota ("Conflito de mesclagem") ao *slide*;
    ii. Se os arquivos no *commit* de mesclagem ainda tiverem os marcadores de conflito (ou seja, os conflitos não foram resolvidos), tente criar um novo *commit* que remova os marcadores e incorpore as alterações da esquerda e da direita de maneira lógica;
        1. Se isso não for possível, o assunto deve ser descartado.
        2. Caso contrário, continue com os arquivos mesclados no novo *commit* de mesclagem que você criou para corrigir os conflitos e adicione sua captura de tela à apresentação como no item "b" (a apresentação terá capturas de tela para a mesclagem original e fixa)
g. Se as alterações nas ramificações esquerda e direita não são incorporadas no *commit* de mesclagem (algumas das alterações podem ter sido removidas ou adaptadas pelo integrador por vários motivos: resolver um conflito de mesclagem, resolver um conflito de compilação, teste ou produção detectado, remover redundância, etc.), ou são incorporados, mas o *commit* de mesclagem tem alterações adicionais feitas pelo integrador (ou seja, algumas alterações não vêm dos ramos esquerdo e direito)
    i. Adicione uma observação ("Mesclar alterado pelo integrador") ao slide;
    ii. Reproduza a mesclagem (`git checkout left-hash; git merge right-hash`) e verifique se o resultado é diferente do que está armazenado no repositório original;
    iii. A análise manual deve prosseguir considerando ambos os *commits* de *merge* (o original no repositório e o repetido), com *screenshots* anotados para ambos (use "// *integrator*" se uma linha foi alterada pelo

integrador; se uma linha foi alterada pelo esquerda, direita e o integrador, adicione os três nomes ao comentário);

1. A análise do original nos ajudará a entender se há interferência localmente observável causada pelas mudanças feitas pela esquerda, direita e pelo integrador (possivelmente uma interferência que não foi causada pelas mudanças na esquerda e direita, mas pelas mudanças feitas pelo integrador);
2. A análise do repetido nos ajudará a entender se há interferência observável localmente causada pelas alterações feitas pela esquerda e pela direita (certamente uma interferência original causada pelas alterações da esquerda e da direita);

4. Resuma as alterações feitas no ramo esquerdo conforme observado no kdiff
    a. O foco está no que o kdiff mostra, não em todas as alterações feitas por *commits* na ramificação esquerda;
    b. Portanto, se um *commit* inicial no *branch* adiciona, por exemplo, uma atribuição à variável x (digamos x=1), e um *commit* posterior remove isso, não veremos isso na Etapa 1 acima; o resumo também não deve mencionar essa mudança, pois o foco está no resultado;
    c. O resumo deve ser expresso em termos de:
        i. As ações realizadas: adicionados, removidos, alterados;
        ii. As mudanças sintáticas feitas nos elementos do programa: classe, campo, construtor, declarações de método, blocos de inicialização, instruções, expressões, argumentos e parâmetros, condições `if` e `while`, `then branch`, `else branch`, corpo do método, corpo do loop, anotações, comentários;
        iii. Como nos estilos a seguir: "*Left* remove a referência ao campo `maxRetries`, na expressão *string* que é retornada pelo método `toString`"; "*Left* altera a condição de uma instrução *if* no método `outerHtmlHead`. A ramificação then do if chama o método `preserveWhitespace`, que agora só é chamado se o nó do parâmetro não for nulo e for uma instância de *Element*"
    d. Para decidir o que deve constar no resumo, pense em um exemplo "*toy*" que reproduza a essência das diferenças no assunto original. O resumo deve então descrever tal essência, utilizando o vocabulário do item acima.
5. Resuma as alterações feitas no ramo direito conforme observado no kdiff (semelhante ao item anterior);
6. Agora, analisando o *commit* do *merge*, verifique se a execução das alterações à esquerda pode substituir uma atribuição (OA) da execução das alterações à direita, ou vice-versa.
    a. Consideramos que pode haver tal fluxo quando as alterações (adições e modificações) em uma das ramificações podem semanticamente (ou seja, sua execução) envolver uma operação de escrita em um elemento de estado que também está associado a uma operação de escrita envolvida no alterações

(adições e modificações) feitas pela outra filial e não há operação de escrita da Base entre elas;

b. Referências de campo como `o.a` e `o.b` são consideradas elementos de estado diferentes e podem ser escritas pelas ramificações sem levar ao OA. Por outro lado, `o` também é um elemento de estado diferente, mas não pode ser alterado junto com `o.a` ou `o.b` sem causar interferência; mas eles (`o` e `o.a`) podem ser escritos por ambos os ramos sem levar ao OA;
c. Para *arrays* cada posição corresponde a um elemento de estado diferente. Dessa forma `a[0]` e `a[1]` podem ser escritos por versões de desenvolvedores diferentes e isso não leva a OA. No entanto, `a` também é um elemento de estado diferente, mas não pode ser alterado jundo com `a[0]` e `a[1]` sem causar interferência;
d. Se uma das ramificações apenas remover o código, podemos saber facilmente que não há OA.
e. Se *overriding* depender da execução de um `if`, considerar como OA;
f. Inicializações de um mesmo *array* ou *var*, ou alterações no mesmo *return* em várias linhas, considerar como OA;
g. Para estabelecer se existe OA, não pense na implementação atual do OA, basta seguir a definição conceitual de OA como nos itens anteriores;

7. Por fim, analisando o *commit* do *merge*, verifique se há interferência localmente observável. Não há necessidade de responder a todas as perguntas abaixo; responder apenas um é suficiente.
    a. As alterações em um dos ramos interferem (afetam negativamente) as alterações no outro ramo?
        i. Existe um elemento de estado x que *left* ou *right* calcula um valor diferente de base e *merge*?
        ii. *left*, *right* e base calculam o mesmo valor para x, mas *merge* calcula um valor diferente?
    b. O comportamento das alterações integradas (no *merge commit*) preserva o comportamento pretendido das alterações individuais (nos ramos esquerdo e direito)?
        i. Os novos efeitos da esquerda (direita) em relação à base são preservados na mesclagem, que não possui novos efeitos além dos da esquerda e da direita?
    c. Você consegue pensar em uma especificação (par pré-pós-condição) que é satisfeita pela declaração à esquerda (ou à direita), mas não é satisfeita pela declaração na mesclagem?
        i. A especificação deve referir-se apenas aos elementos de estado que estão transitivamente envolvidos nas mudanças à esquerda (ou à direita).
    d. Você consegue pensar em um teste que passaria na esquerda (ou direita), mas falharia na base e mesclaria?

    i. A asserção de teste deve se referir apenas aos elementos de estado (que podem envolver exceções, conforme explicado anteriormente) que estão transitivamente envolvidos nas mudanças à esquerda (ou à direita).

e. Você consegue pensar em um teste que falharia na esquerda (ou direita), mas passa na base e mescla?
    i. A asserção de teste deve se referir apenas aos elementos de estado que estão transitivamente envolvidos nas mudanças à esquerda (ou à direita).

f. Referências de campo como `o.a` e `o.b` são consideradas elementos de estado diferentes e podem ser alteradas pelas ramificações sem interferência; `o` também é um elemento de estado diferente, mas não pode ser alterado com `o.a` ou `o.b` sem causar interferência. O mesmo se aplica para *a[0]* e *a[1]* (em vez de `o.a` e `o.b`) e `a` (em vez de `o`). O raciocínio subjacente é que `o.a` pode ser alterado sem alterar ou afetar `o` valor armazenado em `o.b`, enquanto quando alteramos `o`, afetamos automaticamente o valor armazenado em `o.a`. Contrastando, ler `o`, como em `this.m(o)`, não significa necessariamente ler `o.a`, `o.b` e todos os campos de `o`. Elementos de tipos primitivos, *strings*, arquivos e fluxos são considerados atômicos; ou seja, alterar diferentes bits do mesmo inteiro está afetando o mesmo elemento de estado portanto, alterando diferentes partes do mesmo arquivo de texto ou gravando no mesmo fluxo;

g. Se as alterações em pelo menos uma das ramificações afetarem uma assinatura de classe (renomeação da classe ou de seus membros; adição, remoção ou alterações de visibilidade de membros da classe), às duas questões anteriores relacionadas ao teste devem assumir que as assinaturas podem ser feitas da mesma forma. Mesmo em todos os *commits* sem afetar o comportamento.

h. Se as alterações em uma das ramificações são simplesmente alterações de espaço em branco ou alterações nos comentários, claramente não há interferência.

i. Se as alterações em uma das ramificações são simplesmente refatorações estruturais (extrair campo, método, classe, renomeação, etc.) podemos dizer não haver interferência.
    i. Alterando "`if (x==null) return; y=x; if (x!=null) y=x.n();`" em "`if (x==null) return; y=x; y=x.n();`" é uma refatoração, mas não é estrutural, portanto, pode levar a interferências. Portanto, não podemos dizer não haver interferência sem analisar as mudanças no outro ramo.
    ii. Alterar a 'interface' de uma classe `C` não é uma refatoração de `C`. Por exemplo, alterar o método `m` de `C` de "`void m(){this.x=1;}`" para "`int m(){this.x=1; return this.x;}`" não é uma refatoração de `C`, pois isso quebra os clientes de `C`. Mas essa mudança

na declaração de `m`, junto com as mudanças que corrigem todos os clientes de m, é uma refatoração de todo o sistema se os clientes são devidamente fixados de uma forma que preserva o comportamento. Portanto, se uma classe `D` que chama m como em "`c.m();r=c.x`" é alterada para ter "`r=c.m()`", podemos dizer que `D` foi refatorado (mesmo que `C` não tenha sido). Se `D` for a classe em análise, podemos considerar que ela foi refatorada estruturalmente, portanto, não há interferência observável localmente.

j. Se uma declaração de campo (ao invés de um método) foi alterada em ambas as ramificações, a análise fica mais fácil: há interferência apenas se ambas as ramificações alterarem a parte de inicialização da declaração com valores diferentes. É equivalente às duas ramificações alterando a mesma instrução de atribuição no mesmo corpo do método.
    i. Nesses casos, não há necessidade de preencher a coluna OA, pois nem consideramos que a análise poderia ser chamada em tais situações (o *framework* de mineração não invoca a análise para esses cenários)
k. Em muitas, mas não em todas as situações, a possibilidade de uma OA não implicar em interferência localmente observável por vários motivos:
    i. a possibilidade de OA nunca se realizar devido, por exemplo, a uma condição que nunca é verdadeira;
    ii. os OA detectados já existiam na base, e apareceram em nossa análise porque as mudanças nas linhas envolvidas na análise tinham apenas mudanças de espaço em branco ou refatorações estruturais;
    iii. o OA detectado não existia na base, mas não tem efeito prejudicial (por exemplo, quando as mudanças em esquerda e direita atribuem o mesmo valor à mesma variável; temos OA, mas nenhuma interferência real).
    iv. Por isso temos que responder uma das perguntas acima, ao invés de simplesmente checar se a análise foi positiva nos itens anteriores desta lista.
l. Em muitas, mas não em todas as situações, a falta de DF, CF ou OA implica na falta de interferência localmente observável. Isso ocorre especialmente porque as análises podem não ser sólidas (podem levar a falsos negativos, como quando o código usa reflexão, por exemplo), e porque a interferência pode estar associada a outros tipos de análise que não implementamos (como uma análise que verifica se as alterações de uma ramificação afetam o fluxo de controle das alterações da outra ramificação).
m. Se uma das ramificações apenas remover o código, podemos saber facilmente que não há OA, mas ainda pode haver interferência localmente observável.