6
Hist-Inspect: A Ferramenta de Medição e Avaliação

Hist-Inspect[1] (Mara et al. 2010b) é a ferramenta que projetamos e implementamos para apoiar esta pesquisa. Ela visa principalmente disponibilizar recursos que contribuam com avaliações sensíveis à história de código. A Seção 6.1 resume as motivações para o desenvolvimento dessa ferramenta. A Seção 6.2 apresenta os recursos disponíveis na Hist-Inspect. A Seção 6.3 apresenta os principais componentes de sua arquitetura e uma visão geral do seu funcionamento, dando ênfase ao suporte à detecção de anomalias. Vale destacar que demais artefatos relacionados à ferramenta estão disponíveis em (Hist-Inspect 2010).

6.1
Levantamento das Necessidades

Como mencionado na introdução, poucos trabalhos têm investigado a influência de se utilizar informações sobre a evolução dos módulos em estratégias de detecção. Para pesquisas desse tipo seriam necessárias métricas que considerassem a evolução dos módulos (Capítulo 4) e sugestões de estratégias que tivessem em sua composição tais métricas (Capítulo 5). Além disso, o suporte de uma ferramenta de detecção também seria imprescindível.

Algumas das características desejáveis seriam que essa ferramenta: (i) fornecesse suporte à análises considerando a evolução de sistemas e (ii) que também pudesse suportar distintas configurações de estratégias de detecção. A menção ao segundo requisito baseia-se em necessidades tanto de usuários pesquisadores interessados em investigar a eficácia de diferentes combinações de estratégias quanto de usuários desenvolvedores interessados em avaliação de código. No caso desses usuários o requisito é justificado por cada necessidade particular de criar e configurar estratégias de acordo com seus interesses (e.g., trocar a combinação de métricas, valores limites, etc).

Como vimos (Seção 3.3), as ferramentas de detecção atuais (Together 2009, inCode 2009, iPlasma 2009, inFusion 2009) não suportam avaliações sensíveis à história de código e disponibilizam ao usuário pouca

[1] Do inglês, ***Hi**story Sensi**t**ive **Inspect**ion*

ou nenhuma liberdade para configurar estratégias. Devido a tais limitações, projetamos e implementamos a ferramenta Hist-Inspect. Podemos definí-la como uma ferramenta que principalmente é capaz de apoiar a aplicação de diferentes estratégias de detecção de forma flexível, sejam elas sensíveis à história ou não. Contudo, outros recursos sensíveis à história também são disponibilizados como gráficos de evolução de propriedades do código (e.g., tamanho, acoplamento, coesão, dentre outros) e métricas sensíveis à história.

6.2 Descrição das Funcionalidades

Nesta seção descrevemos as funcionalidades da Hist-Inspect implementadas para apoiar avaliações sensíveis à história de código. São elas: (i) a geração de gráficos de evolução, (ii) o cálculo de métricas sensíveis à história e (iii) o suporte à estratégias de detecção.

6.2.1 Suporte a Gráficos de Evolução

Ao se considerar análise sensível à história, faz-se necessário acompanhar a evolução de dados históricos que revelem o comportamento do sistema ao longo do seu ciclo de vida (ou parte desse ciclo). No caso da análise de métricas, a utilização de gráficos pode facilitar a visualização dos resultados ao longo das versões, pois o comportamento evolutivo pode ser analisado sem que seja necessário se prender apenas a visualização de valores numéricos. Dessa forma, podemos dizer que gráficos de evolução elevam o nível de abstração relacionado ao acompanhamento da evolução de valores de métricas.

Além disso, observa-se que no contexto de medições dificilmente uma métrica é analisada de forma isolada. Normalmente, várias métricas são analisadas de forma concomitante. Analisar várias métricas ao longo de várias versões de um sistema pode se tornar bastante complicado. Nesse caso, a utilização de gráficos também facilita o acopanhamento da evolução de várias métricas que podem ser inclusive apresentadas em um mesmo gráfico. Tal fato pode contribuir por exemplo com: (1) identificações de padrões de influência que uma métrica tem sobre a outra (e.g., sempre que o tamanho cresceu, a coesão decresceu) ou (2) avaliações de código do tipo "apesar do número de linhas de código ter crescido muito, a complexidade e a coesão não tem sofrido impactos resultantes da evolução do sistema".

Ao implementarmos o recurso de gráficos de evolução foi considerada tanto a necessidade de se acompanhar a evolução de uma propriedade do código quanto de se acompanhar várias em um mesmo gráfico. A seleção das métricas

a serem analisadas em conjunto ficam à critério do usuário. Os gráficos foram implementados utilizando a biblioteca JFreeChart[2] que possibilitou a obtenção de elevada qualidade dos gráficos, com recursos de alteração de escala dos eixos e seleção de cores, recursos de impressão e possibilidade de salvar os gráficos em arquivos de extensão .png. O gráfico implementado foi o gráfico em linhas, apresentado na Figura 6.1. À princípio, outros tipos de gráficos não foram considerados. Ao lado esquerdo da Figura 6.1 pode ser verificado que foram selecionadas três métricas para que se pudesse acompanhar a evolução dessas no mesmo gráfico.

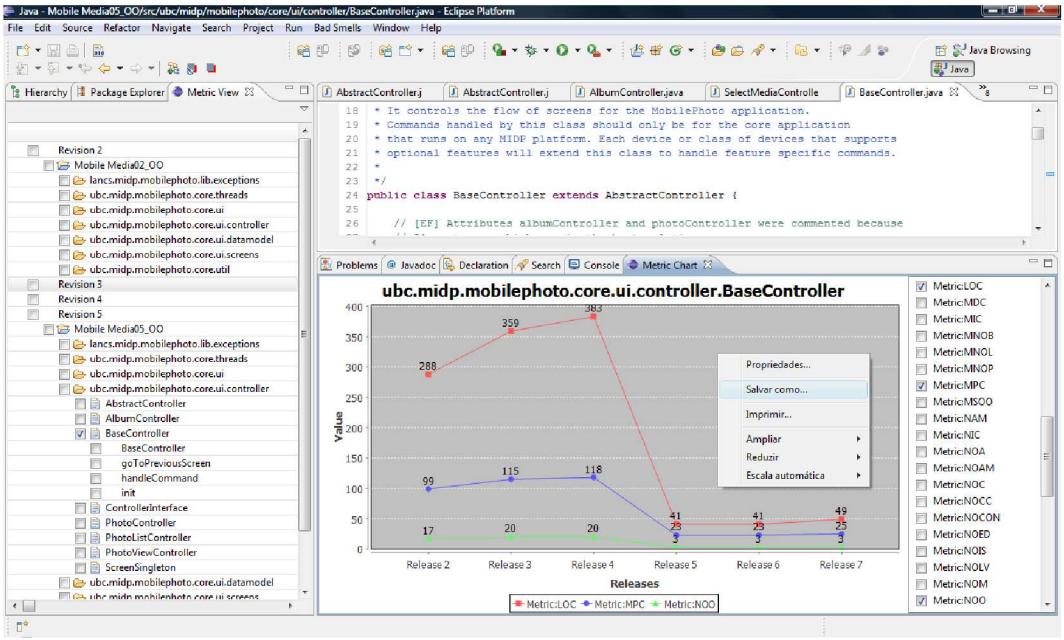

Figura 6.1: Gráficos de evolução através da biblioteca JFreeChart

Atualmente, as métricas convencionais utilizadas pela ferramenta Hist-Inspect e apresentadas nos gráficos não são calculadas pela própria ferramenta. Essa decisão foi tomada pois muitas são as ferramentas que já disponibilizam tais cálculos. Por isso, optou-se por priorizar o desenvolvimento de funcionalidades que não existiam em ferramentas clássicas de detecção. Além disso, acreditamos que seja possível encontrar bibliotecas que talvez já disponibilizem o cálculo de um grande número de métricas convencionais. Sendo assim, preferimos não investir na automação desses cálculos nesse momento e nos concentrar nos recursos sensíveis à história.

Os resultados de métricas convencionais utilizados atualmente são os gerados pela ferramenta Together a qual os disponibiliza em arquivos de extensão .mtbl. Tal ferramenta foi escolhida pois ela suporta um vasto número de métricas. Além disso, ela é bastante utilizada, o que no momento da

[2]http://www.jfree.org/jfreechart/

escolha consideramos como um fator de credibilidade em relação aos valores apresentados.

6.2.2 Apresentação de Métricas Sensíveis à História

A implementação dessas métricas foi motivada pela necessidade de viabilizar a criação e aplicação de estratégias de detecção sensíveis à história. Como vimos no Capítulo 4, através do cálculo dessas métricas é possível obter informações como: a quatidade de vezes que um dado módulo sofreu aumento de linhas de código ao longo de sua história (rniLOC), a variação média de linhas de código em cada versão (rdocLOC), o aumento percentual da complexidade de um módulo em relação a sua versão anterior (rpiWMC), e assim por diante. Todas as métricas implementadas estão apresentadas no Capítulo 4 desta dissertação. Como mostra a Figura 6.2, recursos como *auto-complete* das métricas a serem visualizadas bem como *hide-show* das métricas apresentadas ao usuário também estão implementados.

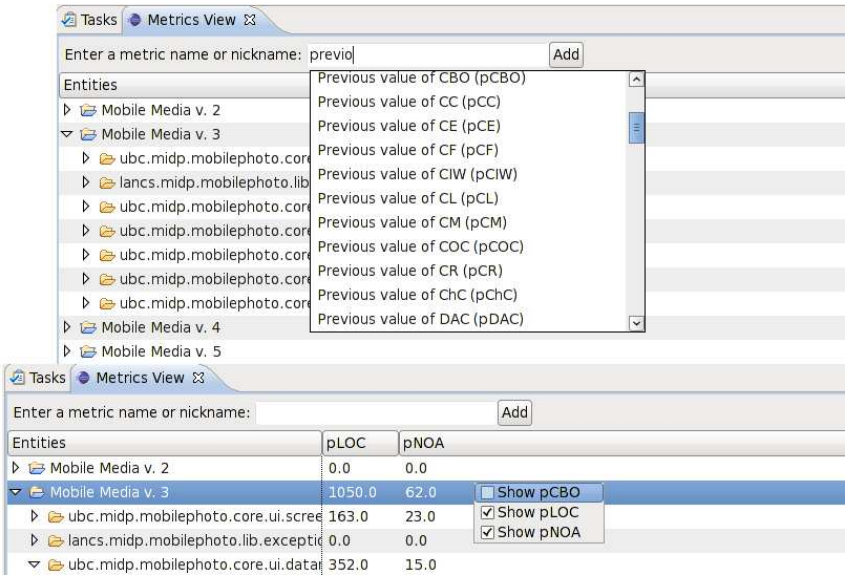

Figura 6.2: Apresentação de métricas sensíveis à história

6.2.3 Suporte a Estratégias de Detecção

Durante o processo de inspeção automática de um determinado módulo se uma regra é satisfeita, então aquele módulo deve ser apresentado ao usuário por possuir os sintomas especificados por aquela regra. Esse é exatamente o objetivo das estratégias de detecção: apresentar ao usuário todos os módulos em conformidade com as métricas, limites e operadores de composição que

integram a estratégia. Nesse contexto, é bastante comum que usuários tenham o interesse de avaliar outros tipos de estratégias ou até o de adaptar estratégias já existentes. Recursos como esse deveriam ser facilmente disponibilizado pelas ferramentas de detecção. Entretanto, isso não é sempre observado (Seção 3.3). Na maioria dos casos, não se sabe nem que tipos de estratégias estão sendo consideradas nas detecções pois essas informações não são transparentes ao usuário.

O grande diferencial da Hist-Inspect está na liberdade que ela concede ao usuário na configuração das estratégias, não só em relação a valores limites que se deseje ajustar, mas também nas métricas consideradas. Além disso, assim como no mecanismo de Detecção Experta (Seção 3.4), a ferramenta proposta permite que usuários realizem especificações em alto nível. Dessa forma, algoritmos de detecção são gerados automaticamente a partir de tais especificações, sem a necessidade de codificar classes ou métodos. Diferentes configurações de estratégias podem ser consideradas de acordo com as necessidades particulares de cada usuário. Inclusive, é possível considerar estratégias formadas apenas por métricas convencionais, apenas por métricas sensíveis à história ou pela combinação de ambas essas estratégias. Tal característica faz com que a Hist-Inspect disponibilize recursos não encontrados em nenhuma das ferramentas relacionadas. A Figura 6.3 apresenta uma tela com resultados de detecções realizadas pela Hist-Inspect. Tais resultados são apresentados na forma de relatórios HTML. Tal fato permite que uma pessoa consiga acompanhar o resultado das avaliações de sistemas, mesmo que ela não tenha a ferramenta instalada.

Figura 6.3: Relatório HTML das anomalias detectadas

Utilização de Linguagem Específica de Domínio do Tipo Interna

Para possibilitar flexibilidade na configuração de estratégias, a ferramenta proposta utiliza uma linguagem específica de domínio (DSL) (Deursen et al., 2000). Optou-se por uma DSL do tipo interna (Hudak 1996), decisão que nos permitiu reutilizar todos os elementos sintáticos e também o interpretador da linguagem base utilizada, o JavaScript. Tais reutilizações trouxeram como principal vantagem a rápida implementação da funcionalidade considerada.

Por exemplo, o fato de termos utilizado uma linguagem específica de domínio do tipo interna, ou seja, que toma como base outra linguagem, nos desobrigou da necessidade de especificar uma gramática (BNF) ou outros elementos necessários na definição de linguagens. Todos os elementos de especificação de JavaScript como operadores lógicos (OU e E, representados respectivamente como "||" e "&&") e operadores matemáticos (como "==", ">", "<", "≥"), dentre outros, são automaticamente disponibilizadas para serem utilizados na definição das estratégias. Já as métricas utilizadas são entendidas como simples variáveis pelo interpretador de JavaScript.

Chamamos nossa linguagem de especificação de estratégias de DSSL[3] (Linguagem de Especificação de Estratégias de Detecção). Vimos na Seção 3.4 que na abordagem DETEX as regras eram especificadas em Cartões de Regras utilizando também uma linguagem específica de domínio a qual foi chamada de SADSL. Na Hist-Inspect, as especificações de estratégias e anomalias são realizadas em arquivos de configuração chamados Catálogo de Estratégias e Catálogo de Anomalias, respectivamente. Exemplos desses catálogos são apresentados pelas Figuras 6.4(a) e 6.4(b).

```
<?xml version="1.0" encoding="UTF-8"?>
<rule-catalog>
    <rule id="sampleRule1"
        name="Sample Rule 1"
        anomaly="GC"
        expression="LOC > 244 || CBO > 5" />
    <rule id="sampleRule2"
        name="Sample Rule 2"
        anomaly="FE"
        expression="LOC > 100 && WMC > 20" />
</rule-catalog>
```

6.4(a): Catálogo XML de estratégias

```
<?xml version="1.0" encoding="UTF-8"?>
<anomaly-catalog>
    <anomaly id="GC"
        name="God Class"
        applyTo="class" />
    <anomaly id="FE"
        name="Feature Envy"
        applyTo="method" />
</anomaly-catalog>
```

6.4(b): Catálogo XML de anomalias

Figura 6.4: Exemplos de especificação de estratégias e respectivas anomalias

A principal diferença entre um Cartão de Regras da abordagem DETEX e um Catálogo de Estratégias da Hist-Inspect é que um Cartão de Regras é responsável pela detecção de uma única anomalia, cujo nome da anomalia é

[3]Do inglês, ***D**etection **S**trategy **S**pecification **L**anguage*

o próprio nome do cartão. Sendo assim, o número de cartões é diretamente proporcional ao número de anomalias que se deseja detectar. Já um único Catálogo de Estratégias da Hist-Inspect considera todas as estratégias que precisam ser executadas, independente se serão analisadas um ou dez tipos de anomalias. As anomalias consideradas no Catálogo de Estratégias, por sua vez, devem ter sido especificadas no Catálogo de Anomalias.

Como mostra a Figura 6.4(a), as expressões associada aos elementos do tipo `expression` do Catálogo de Estratégias correspondem as métricas, operadores de comparação e valores limites de uma estratégia. As expresões associadas ao elemento `anomaly` dizem respeito a que tipo de anomalia as respectivas estratégias vão ser responsáveis por detectar. No elemeto `applyTo` do Catálogo de Anomalias (Figura 6.4(b)) fica especificado que tipos de módulos devem ser inspecionados (classes, métodos ou pacotes) para cada tipo de anomalia considerada.

6.3 Visão Geral da Arquitetura e Funcionamento

Essa seção apresenta o fluxo de funcionamento e alguns aspectos técnicos da ferramenta proposta. As funcionalidades da Hist-Inspect foram implementadas em dois módulos principais utilizando a linguagem Java. O primeiro é o módulo de geração de gráficos de evolução, implementado através da plataforma Eclipse. O segundo é o módulo de geração de relatórios de detecções, resultantes da aplicação de estratégias de detecção. Por enquanto, esse segundo módulo é disponibilizado através de uma interface de linha de comando. Entretanto, existe a intenção de migrá-lo para que também esteja sobre a plataforma do Eclipse. Por possuir as funcionalidades que julgamos mais relevantes ao contexto de estratégias de detecção, iremos apresentar apenas o módulo responsável pelo suporte à estratégias de detecção. Os integrantes desse módulo são apresentados pela Figura 6.5.

Como mostra a Figura 6.5, o módulo toma como entrada um arquivo chamado *Lista de Arquivos de Métricas*. Essa lista possui a localização dos arquivos que possuem os valores de métricas convencionais em cada uma das versões do sistema. O *Oráculo de Anomalias* é uma entrada opcional e apenas é fornecida se o usuário conhecer qual é a saída de detecção esperada para cada anomalia e desejar calcular medidas de precisão e revocação (Moha et al. 2010) das estratégias consideradas. Ambas as entradas são implementadas através de arquivos XML. Os n arquivos endereçados na *Lista de Arquivos de Métricas* e que possuem as métricas convencionais das n versões do sistema são arquivos

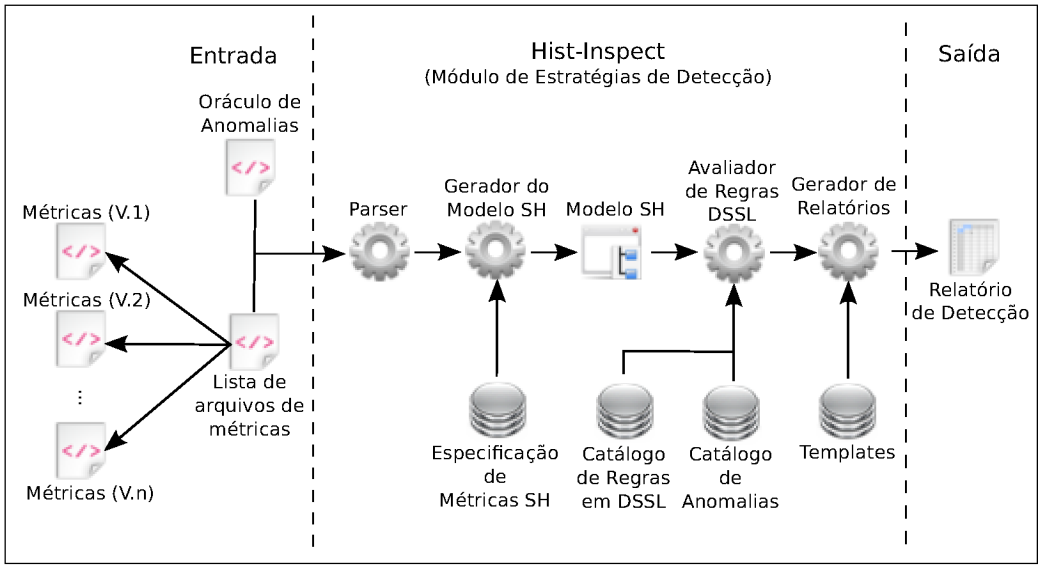

Figura 6.5: Funcionamento e elementos principais da arquitetura da Hist-Inspect

`.mtbl`[4]. Esses arquivos são os resultantes dos cálculos de métricas convencionais gerados pela ferramenta Together (Together 2009). Conforme justificado na Seção 6.2.1, optou-se por não efetuar o cálculo de métricas convencionais e fornecer os resultados dessas métricas através de arquivos de entrada. Isso para que pudéssemos em um primeiro momento nos concentrar principalmente na implementação de recursos sensíveis à história.

O elemento *Parser*, na Figura 6.5, transforma a representação textual do XML de entrada em um modelo de domínio a ser utilizado pela aplicação. Entretanto, o modelo gerado pelo *Parser* armazena apenas informações de métricas convencionais. O *Gerador do Modelo SH* é quem lê as *Especificações das Métricas Sensíveis a História* e transformar o modelo convencional em um *Modelo Senvível à História*. Nesse momento, cada um dos módulos do sistema tem associado a si métricas convencionais e métricas sensíveis à história cujos resultados poderão ser avaliados na execução das estratégias.

Grande parte do *Parser* que converte os arquivos de entrada em classes a serem utilizadas pelo sistema foi gerado utilizando o *framework* XMLBeans[5]. A Especificação de Métricas SH foi feita através de classes que implementam uma interface Java específica para tais métricas. A partir do *Modelo com Métricas SH* de cada entidade é transferida ao *Avaliador de Regras* a responsabilidade de realizar as detecções. Nesse caso, o *Avaliador de Regras* carrega o Catálogo de Estratégias, que contém as especificações das estratégias em DSSL, e o Catálogo de Anomalias, que contém as especificações das anomalias mencionadas

[4]Extenção do arquivo gerado pelo Together com os resultados de métricas convencionais solicitadas.

[5]http://xmlbeans.apache.org/

nas estratégias. Esses dois catálogos são implementados através de arquivos XML, como ilustra a Figura 6.4. O *Avaliador de Regras* gerencia o contexto de execução do JavaScript, criando as variáveis que representam as métricas utilizadas por cada estratégia e atribuindo os seus respectivos valores de acordo com a entidade que está sendo avaliada (pacote, classe ou método). Finalmente, as estratégias indicadas pelo atributo `expression` (Figura 6.4(a)) são aplicadas aos módulos do sistema. Caso a expressão de detecção seja verdadeira para uma determinado módulo, o *Avaliador de Regras* destaca a ocorrência da anomalia, a qual é indicada pelo elemento `anomaly` (Figura 6.4(a)).

Como mostra a etapa final da Figura 6.5, o *Gerador de Relatórios* combina as informações produzidas pelo *Avaliador de Regras* com um *Template HTML*, gerando um *Relatório de Detecção* que será apresentado ao usuário. O *Gerador de Relatórios* processa templates de relatórios utilizando a linguagem de templates Velocity[6]. Opcionalmente, é possivel também inserir nesse relatório de detecções informações sobre falsos positivos e negativos, precisão e revocação das estratégias aplicadas. Esses são calculados a partir do elemento *Oráculo de Anomalias*, caso esse tenha sido fornecido como entrada. A Figura 6.3 apresentou um exemplo de relatório HTML de detecção gerado pela ferramenta.

[6] http://velocity.apache.org/