Aplicação de Teste de Unidade Baseado em *Designs* Combinatoriais no Ambiente *Geotool*

SER 300 - Introdução ao Geoprocessamento
Juliana Marino Balera

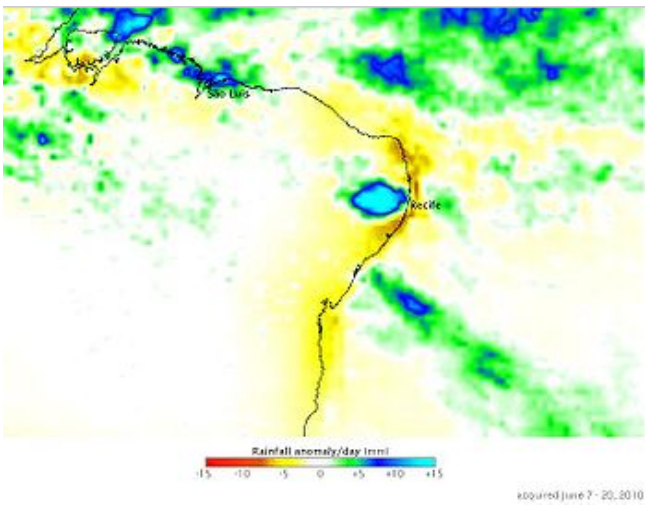
June 16, 2017

1 Introdução

O Brasil é um país em que recorrentemente ouve-se falar de desastres ambientais de grandes proporções, ocasionados tanto pela ação do homem, como pela ação da natureza. Esses desastres tem efeito permanente, e trazem impactos negativos a diversos setores do país, sendo o pior deles os impactos sociais derivadas das inúmeras perdas de vidas que se dão em decorrência desses acidentes ambientais.

No ano de 2010, as enchentes que ocorreram nos estados brasileiros de Alagoas e Pernambuco devido a uma anomalia de precipitação, como ilustrado na Figura 1, levaram a morte de ao menos 53 pessoas e a declaração de situação de emergência por parte de mais de 30 municípios dos dois estados.

Figure 1: Anomalia de preciptação - as regiões coloridas em azul representam índices pluviométricos acima da média. Fonte: NASA.



Mais tarde, no ano de 2011 ocorreu a maior tragédia de clima da história do país, segundo o site G1, devido as chuvas na Região Serrana do Rio de Janeiro. Foram cerca de 506 mortes contabilizadas, fora os impactos em diversos setores, sendo um dos maiores os impactos ambientais, estimados em R$71.4 milhões. A destruição foi de proporções gigantescas, como ilustrado na Figura 2.

Figure 2: Foto aérea de uma das regiões atingidas. Fonte: G1Globo

Esses eventos justificam que a Análise de Riscos de regiões suscetíveis seja feita de maneira adequada. Para isso, é importante a simulação de cenários de desastres ambientais, de maneira que os resultados extraídos proporcionem uma perspectiva a mais a cerca de tais problemas para que os profissionais relacionados a essas áreas possam estar adiantados diante de catástrofes como essas, podendo garantir que os impactos sejam menores do que comumente vemos acontecer.

Nesse contexto, a computação aliada ao geoprocessamento oferecem uma série de ferramentas que estão presentes em diversas partes do processo de prevenção de desastres. O que faz com que cada vez mais profissionais vinculados a essas áreas, como geógrafos e geólogos, que a princípio, não tem conhecimento dos processos a cerca do desenvolvimento de software, lidem com uma rotina típica da área computacional: desenvolvimento de scripts, manipulação de banco de dados, etc. Mesmo sem essa formação, tais profissionais têm se saído muito bem, dada a larga quantidade de publicações de trabalhos relacionados a computação aplicada ao geoprocessamento. No entanto, assim como toda plataforma computacional, os produtos desenvolvidos por esses profissionais não está imune a falhas.

Um produto de software que é amplamente utilizado por profissionais do geoprocessamento é o GeoTool. *GeoTools* é uma biblioteca Java que implementa conceitos de um Sistema de Informações Geográfico (SIG) , onde é possível o usuário desse sistema realizar manipulações de dados espaciais, através da implementação/utilização de métodos/classes disponibilizadas por essa biblioteca.

A ferramenta, no entanto, exige um certo grau de conhecimento de computação por parte do usuário, uma vez que o seu uso, seja pela simples aplicação de suas funcionalidades, quanto para a sua integração como módulo de algum outro sistema, é feito através da escrita de scripts.

Assim como qualquer plataforma computacional, os produtos de software desenvolvidos no contexto de geoprocessamento estão sujeitos a falhas. Além do mais, por se tratar de softwares a frente de processos críticos, é fundamental que estejam funcionando de acordo com os requisitos de sua especificação, afinal, uma simples troca de sinal pode levar a uma análise de risco completamente errada.

Dessa forma, é fundamental que tenhamos em mãos metodologias de teste de software que sejam fáceis de se aplicar e ao mesmo tempo eficientes (oferece grande potencial de revelação de defeitos). Nesse contexto, Designs Combinatoriais vêm chamando têm sido amplamente empregado pela comunidade de desenvolvimento de software, uma vez que contempla as duas características.

Designs Combinatoriais é um ramo da análise combinatória que tem como objetivo reorganizar um conjunto finito de elemento de acordo com regras de balanço ou simetria. No contexto de Teste de Software, essa técnica é aplicada através do t-way testing, uma metodologia que se baseia na idéia de que a maioria das falhas pode ser revelada a partir da interação de fatores (varáveis do Sistema Sob Teste).

Diante disso, o objetivo desse trabalho é a aplicação do algoritmo TTR na geração de casos de teste de unidade no contexto da ferramenta *GeoTools*. Esse trabalho está organizado em 5 seções. A seção 2 é a fundamentação teórica para a realização do trabalho proposto. A seção 3 contempla a descrição do experimento proposto. Na seção 4 estão reunidos os resultados do experimento. E por fim, a seção 5 faz uma breve conclusão.

2 Fundamentação Teórica

2.1 Teste de Software

Testar um software consiste em submetê-lo a um conjunto de testes, e então, verificar se a saída produzida é a esperada. O processo de teste pode ser dividido em três etapas:

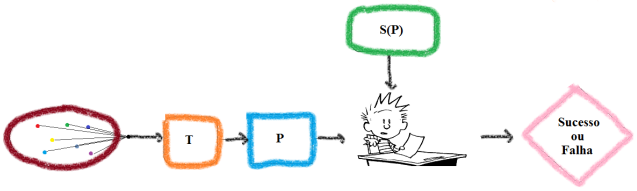
a) **Teste de Unidade:** consiste em testar as menores unidade do software, como por exemplo métodos, rotinas, funções e procedimentos;

b) **Teste de Integração:** ocorre após a fase de teste de unidade estar completa. Consiste em testar a interação entre as unidades testadas, para verificar como elas funcionam em conjunto;

c) **Teste de Sistema:** consiste em testar o sistema como um todo. Te como intuito, avaliar se todas as funcionalidade do sistema foram implementadas e, se elas estão funcionando corretamente.

Todas as etapas ocorrem da mesma forma, no entanto, considerando contextos diferentes dependendo da fase de teste. A Figura 3, ilustra bem esse processo. Apenas alguns pontos do subdomínio de entrada são selecionados para compor os dados de teste, que são transformados em um conjunto de casos de teste T (que no fundo, consiste de uma tupla com os dados de teste e o resultado esperado para esses dados) e submetidos ao programa P. O resultado produzido por P é avaliado pelo testador S(P) que irá determinar se houve sucesso ou falha desse teste.

Figure 3: Processo de Teste de Software. Fonte: [1]



No entanto, saber quais pontos desse domínio selecionar ainda é uma questão em aberto. Selecionar todos os pontos do domínio, ou seja, testar o software de maneira exaustiva não é uma opção, dada a larga quantidade de entradas ao qual o Software Sob Teste pode estar sujeito. Desse modo, a atividade de geração e seleção de casos de teste é um dos maiores desafios da atividade de teste.

Para solucionar esse problema, são empregadas abordagens sistemáticas para a geração desses casos de teste, conhecidas como Técnicas de Teste de Software. No fundo, as Técnicas de Teste tem como intuito contextualizar os artefatos de software que serão utilizados para dar origem aos casos de teste, como podemos ver na Tabela 1. Cada Técnica de Teste possuí um conjunto de Critérios de Teste associada. É a partir dos Critérios de Teste que se obtém os casos de teste.

Table 1: Adaptação da definição de [2] para a classificação das diferentes maneiras de geração de casos de testes [3].

Categoria	**Origem**	**Técnica**	**Exemplo**
1	Requisitos (Informal)	Caixa-Preta	- Ad hoc - Análise do valor limite - Particionamento em classes de equivalência - Teste aleatório - *Designs* combinatoriais/ *t-way testing*
2	Requisitos (Formal)	*Baseado em modelos*	- Baseado em *Statecharts* - Baseado em Máquinas de Estados Finitos - Baseado em B - Baseado em Z
3	UML	Baseado em documentos UML	-Baseado em UML (Máquinas de Estados, Diagramas de Classes, Sequencia, etc.)
4	Código-fonte	Caixa-Branca	- Teste de Fluxo de Controle - Teste de Fluxo de Dados - Teste de Mutação

2.1.1 Designs Combinatoriais

Designs Combinatoriais consiste de um conjunto de técnicas que têm como objetivo a geração de casos de teste a partir da modificação de um conjunto finito de elementos de acordo com alguma regra de balanço ou simetria. Essa técnica tem-se mostrado eficiente na revelação de defeitos por meio da interação de várias variáveis de entrada (fatores). Nos experimentos realizados por [4] e [5], conjuntos de casos de teste que cobrem todas as interações de parâmetros, dois a dois, puderam revelar de 50% a 75% dos defeitos de um programa.

Segundo [6], as técnicas de *designs* combinatoriais podem ser classificadas em *designs* balanceados e *designs* não-balanceados. Matrizes Ortogonais (MO) e Matrizes Ortogonais com Níveis Variados (MONV) são exemplos de *designs* balanceados. Tais técnicas exigem que cada elemento se repita exatamente o mesmo número de vezes. Já as técnicas Matriz de Cobertura (MC) e Matriz de Cobertura com Níveis Variados (MCNV) são exemplos de *designs* não-balanceados, em que os elementos não aparecem exatamente o mesmo número de vezes. i.e., cada elemento aparece no mínimo uma vez. Em função da idéia de revelação

de falhas a partir da interação de fatores, MCNV é a técnica de *designs* combinatoriais preferida entre os testadores, uma vez que, em geral, elas produzem matrizes de dimensões menores que um MONV, uma vez que basta cobrir uma única vez cada elemento.

Definição 2.5 Uma MCNV é uma matriz definida por MCNV(N,$l_1^{k_1} l_2^{k_2} ... l_p^{k_p}$,$t$), onde N é a quantidade de linhas da matriz onde o fator k_1 tem l_1 níveis, ..., e o fator k_p tem l_p níveis. O parâmetro t corresponde ao grau de interação das variáveis, ou *strength*. No contexto de teste de software, cada combinação de fatores (ou seja, cada linha da MCNV) é um caso de teste. Dessa forma, a MCNV é uma *suite* de teste.

Dessa forma, uma MCNV é gerada por *t-way testing*. Por exemplo, o clássico *2-way testing* (ou, *pairwise testing*), produz um MCNV para $t = 2$.

Segundo [7], para se ter uma idéia da redução do conjunto de casos de teste obtida via *designs* combinatoriais, considere que existam 10 fatores $(A, B, \cdots, J)$ e que cada fator possua 5 níveis, ou seja, $A = \{a_1, a_2, \cdots, a_5\}$, $B = \{b_1, b_2, \cdots, b_5\}$, ..., $J = \{j_1, j_2, \cdots, j_5\}$. Se uma combinação exaustiva fosse realizada, teria-se um total de $5^{10} = 9.765.625$ casos de teste da forma $ct = \{a_k, b_k, \cdots j_k\}$. Utilizando um algoritmo para gerar *designs* combinatoriais, como o algoritmo TTR [8, 7] apresentado nesse trabalho, com grau de interação t igual a 2, teria-se 48 casos de teste.

2.2 T-Tuple Reallocation

O algoritmo *T-Tuple Reallocation* (TTR) [9] tem como objetivo gerar MCAs por meio da Técnica de Teste *t-way testing*. Uma visão em alto nível do TTR é apresentada no Algoritmo 1. O conceito geral do TTR é construir uma matriz de *designs* combinatoriais por meio da realocação de tuplas da matriz Θ para a matriz M, e então cada tupla realocada deve cobrir o maior número de tuplas ainda não cobertas considerando o parâmetro chamado *goal* (ζ). Note também que f é o conjunto de fatores e t é o grau de interação (*strength*). Além do mais, cada tupla é uma lista ordenada de elementos. A seguir breves comentários sobre as características do algoritmo.

Algorithm 1 O Algoritmo TTR

```
1: Θ ← constructor(f, t)
2: M ← calculateInitialSolution(Θ)
3: while Θ ≠ ∅ do
4:     ζ ← calculateZeta(M)
5:     (M, Θ) ← main(M, Θ, ζ)
```

Constructor: esse procedimento tem por objetivo gerar todas as tuplas a serem cobertas pelo TTR. Ele as produz da seguinte maneira: todas as interações de fatores de acordo com o *strength* são produzidas e armazenadas em Θ. Depois disso, todas as combinações de valores correspondetes a esses

grupos são feitas. Por exemplo, vamos considerar dois fatores A e B com dois níveis cada e $t = 2$ (*strength*). Será feito a interação de fatores 'AB" com 4 tuplas: (A1, B1), (A1, B2), (A2, B1) e (A2, B2). Cada tupla associada com o parâmetro chamado *flag*, que ajuda o processo de seleção e realocação de tuplas.

i	A	B	C
1	1	1	
2	1	2	
3	2	1	
4	2	2	
5	1		1
6	1		2
7	1		3
8	2		1
9	2		2
10	2		3
11		1	1
12		1	2
13		1	3
14		2	1
15		2	2
16		2	3

i	A	B	C
1	1	1	1
2	1	2	2
3	1	1	3
4	2	2	1
5	2	1	2
6	2	2	3

Figure 4: Exemplo de uma operação do TTR. A esquerda Θ e a direita a matriz M final.

Initial Solution: Esse procedimento consiste em construir uma solução incial, que é feita através da seguinte maneira: a partir das combinações de fatores geradas pelo procedimento *constructor*, TTR seleciona a combinaçao que possuí a maior quantidade de tuplas. Cada tupla então é removida da matriz Θ e realocada na matriz M, uma a uma, na forma de casos de teste.

Goal: É um valor associado a cada linha da matriz M e está relacionado à cobertura de "potencial" de cada linha da matriz. Eles são calculados simplesmente combinando o valor da interação, t, e o número de fatores cobertos por cada tupla em um determinado momento. Por exemplo, considere os seguintes fatores A, B e C com 2, 2 e 3 níveis, respectivamente, e $t = 2$, existem três grupos de fatores possíveis: "AB", "AC" e "BC". Cada interação de fatores terá $(2 * 2) = 4$, $(2 * 3) = 6$ e $(2 * 3) = 6$ tuplas, respectivamente.

Main: Esse procedimento transforma a matriz produzida pela solução inicial na solução final (conjunto final de casos de teste). Depois de construir Θ, a matriz inicial M e o cálculo de cada meta (ζ), o procedimento *main* constrói casos de teste para cobrir a maior quantidade de tuplas por meio da realocação gradual de tuplas de Θ em M. Para cada iteração, a combinação de fatores com mais tuplas descobertas é selecionada, e cada uma de suas tuplas é temporaria-

mente combinada com cada linha da matriz M. Então, é calculada a quantidade de tuplas não cobertas por esta linha temporária e comparada com o valor da meta para essa linha. Se esses valores forem iguais, a tupla será reatribuída permanentemente a essa linha de M, caso contrário, ela é comparada com a próxima linha de M até que se encaixe em algum lugar. O procedimento realiza essa comparação até que todas as tuplas dessa combinação de fatores sejam realocadas.

É importante considerar o caso em que uma tupla é comparada com todas as linhas da matriz M, no entanto, não se encaixa em nenhum lugar. Para este caso, o TTR gera uma "marcação" e o algoritmo passa para a próxima combinação de fatores. Quando esta combinação é novamente selecionado por *main* como a combinação com a maior quantidade de tuplas descobertas, essa marcação indica que o valor da meta deve ser reduzido. Isso garante que todas as tuplas sejam cobertas pelo algoritmo.

Consideremos a Figura 4 onde vemos os fatores A, B e C, com 2, 2 e 3 níveis, respectivamente. Vamos também considerar $t = 2$. Todas as tuplas geradas pelo procedimento *constructor*, ou seja, o Θ inicial, podem ser visualizadas na matriz mais à esquerda na Figura 4. A matriz mais à direita é a solução final (matriz M). Para conseguir isso, o procedimento *calcularInitialSolution* seleciona o grupo de fatores com o maior número de tuplas, neste caso "AC" correspondente às tuplas 5-10 (6 tuplas) no Θ inicial que são realocados em M. Observe que a combinação de fatores " BC " também possui 6 tuplas (tuplas 11-16) e pode ser selecionado. O TTR sempre escolhe a combinação de fatores que armazena a maior quantidade de tuplas de acordo com a entrada de fatores. Posteriormente, o procedimento *calculateZeta* encontra as metas (ζ) de acordo com o número de combinações de fatores que não foram cobertas. Neste ponto, todos as metas são 2 ("AB" e " BC" não estão cobertos). Então, o procedimento *main* procura a combinação de fatores que possuí a maior quantidade de tuplas ainda não cobertas. A tupla 11 é combinada com a tupla 5 e isso gera o caso de teste 1 na matriz M. Assim, esse processo aborda duas tuplas (1 e 11) do Θ inicial e isso é exatamente igual a meta (ζ). Em seguida, não é necessário executar qualquer outra ação relacionada ao caso de teste 1 em M . Em seguida, a tupla 12 é selecionada e o processo de comparação é repetido.

3 Estudo de Caso: GeoTools

GeoTools é uma biblioteca Java de código aberto que implementa conceitos de um Sistema de Informações Geográfico (SIG), onde é possível o usuário realizar manipulações de dados espaciais, através da implementação/utilização de um conjunto de APIs disponibilizadas por essa biblioteca. A biblioteca *GeoTools* oferece uma série de recursos, sendo os principais:

- Definição de interfaces para a utilização de conceitos espaciais e estruturas de dados;
- Uma API de acesso a dados espaciais, que oferece suporte de acesso a recursos,transações e bloqueio entre threads;

- Plugins *GeoTools*: abra o sistema plug-in permitindo que você utilize os formatos adicionais da biblioteca;
- Fornecer capacidades adicionais a partir da instalação da biblioteca central;
- O GeoTools inclui um banco de dados muito extenso de projeções de mapas definidas pelos números de referência *European Petroleum Survey Group* (EPSG);

Além disso, a biblioteca *GeoTools* implementa os seguintes padrões da *Open Geospatial Consortium* (OGC).

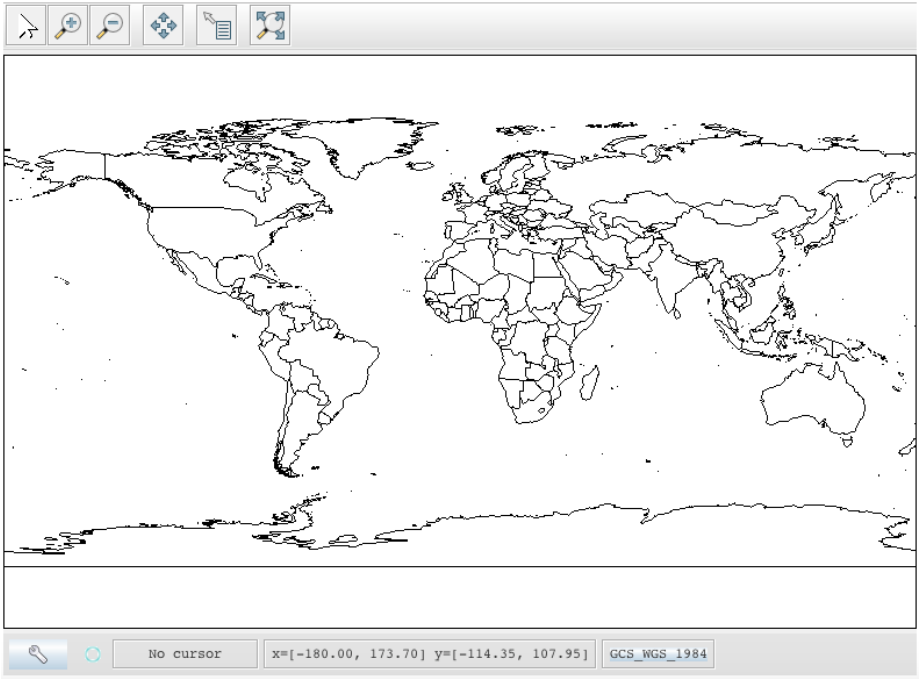
- OGC codificação de estruturas de dados e mecanismo de renderização;
- OGC modelo de recurso geral, incluindo suporte de recursos simples;
- OGC representação de informações raster;
- Filtro OGC e *Common Constraint Language* (CQL);
- Clientes para Web Feature Service, Web Map Service e suporte experimental para Web Process Service;
- Geometria ISO 19107;

Dessa forma, para a utilização dos recursos da biblioteca, se torna necessário que o usuário programe a sua própria interface de acesso a API, através da classe **main**, de maneira a ele próprio gerenciar suas funcionalidades. O exemplo de classe **main** da Figura 5 ilustra a classe *QuickStart*, onde o usuário cria uma interface de acesso as APIs *GeoTools* para visualizar o *shapefile* da Figura 6.

Figure 5: *Script* para de abertura de mapa. Fonte: *GeoTools* web.

```
Pacote  org . Geotools . Tutorial . Quickstart ;

Importar  java.io.File ;

Importar  org.geotools.data.FileDataStore ;
Importar  org.geotools.data.FileDataStoreFinder ;
Importar  org.geotools.data.simple.SimpleFeatureSource ;
Importar  org.geotools.map.FeatureLayer ;
Importar  org.geotools.map.Layer ;
Importar  org.geotools.map.MapContent ;
Importar  org.geotools.styling.SLD ;
Importar  org.geotools.styling.Style ;
Importar  org.geotools.swing.JMapFrame ;
Importe  org.geotools.swing.data.JFileDataStoreChooser ;

/ **
* Solicita ao usuário um shapefile e exibe o conteúdo na tela em um quadro de mapa.
* <P>
* Este é o aplicativo GeoTools Quickstart usado na documentação e nos tutoriais. *
* /
Classe pública  Quickstart {

    / **
     * Aplicação de demonstração do Quickstart do GeoTools. Solicita ao usuário um shapefile e exibe seus
     *
conteúdos na tela em um quadro de mapa      * /
    public  static  void  main ( String []  args )  throws  Exception  {
        // exibe uma caixa de diálogo de seleção de
        arquivo de  armazenamento de dados para shapefiles File file  =  JFileDataStoreChooser . ShowOpenFile ( "shp" ,  nulo );
        Se  ( arquivo  ==  nulo )  {
            retornar ;
        }

        FileDataStore  store  =  FileDataStoreFinder . GetDataStore ( arquivo );
        SimpleFeatureSource  featureSource  =  store . GetFeatureSource ();

        // Crie um conteúdo do mapa e adicione nosso shapefile a
        MapContent  map  =  MapContent novo  (); Mapa . SetTitle ( "Início Rápido" );


        Style  style  =  SLD . CreateSimpleStyle ( featureSource . GetSchema ());
        Camada  layer  =  new  FeatureLayer ( featureSource ,  style );
        Mapa . AddLayer ( camada );

        // Agora exiba o mapa
        JMapFrame . ShowMap ( mapa );
    }

}
```

Figure 6: Visualização de *shapefile*. Fonte: *GeoTools* web.



Outro exemplo simples, é a abertura de um *shapefile* em um sistema de coordenadas diferente do original. A Figura 7 ilustra um trecho da classe **main** utilizada para implementar essa funcionalidade. Observe que, as transformações matemáticas utilizadas estão encapsuladas em apenas alguns métodos. A Figura 8 ilustra o *shapefile* original, e a Figura 9 ilustra o *shapefile* após a mudança do sistema de coordenadas, onde claramente podemos ver as deformações na parte superior a esquerda da imagem.

Figure 7: Trecho de código que implementa o recurso de mudança de sistema de coordenadas. Fonte: *GeoTools* web.

```
CoordinateReferenceSystem dataCRS = schema.GetCoordinateReferenceSystem();
CoordinateReferenceSystem worldCRS = map.GetCoordinateReferenceSystem();
booleano branda = verdadeiro; // permite algum erro devido a diferentes datums
MathTransform transform = CRS.findMathTransform(dataCRS, worldCRS, branda);
```

Figure 8: Mapa no sistema de coordenadas *GCS WGS84*. Fonte: *GeoTools* web.

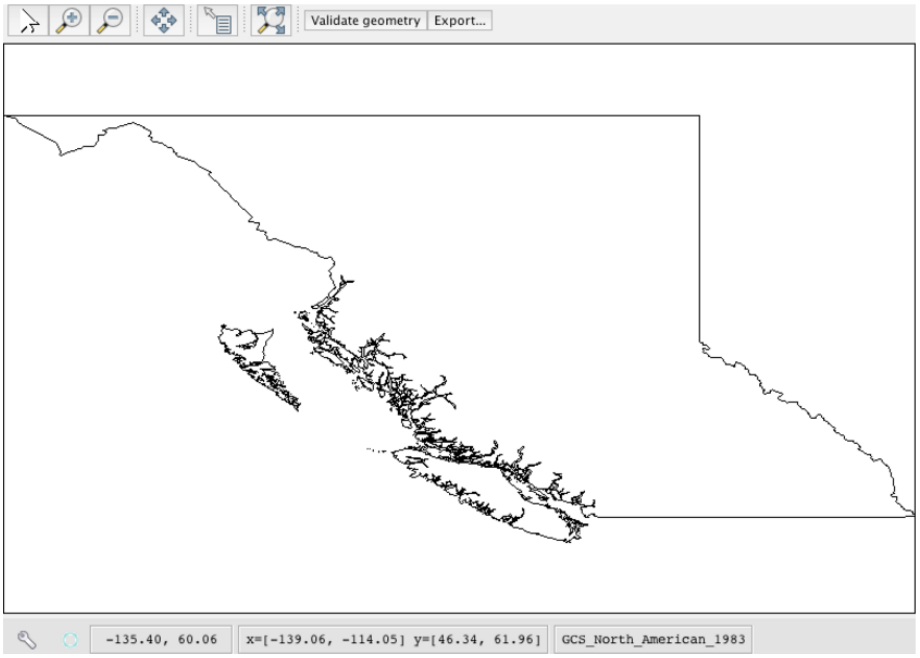
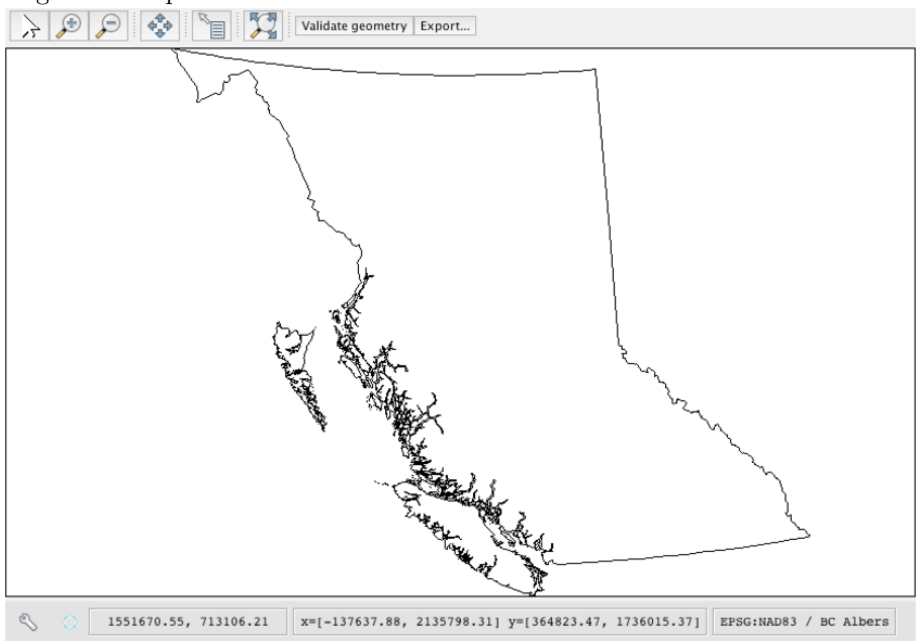


Figure 9: Mapa no sistema de coordenadas *BC Albers*. Fonte: *GeoTools* web.



4 Experimento

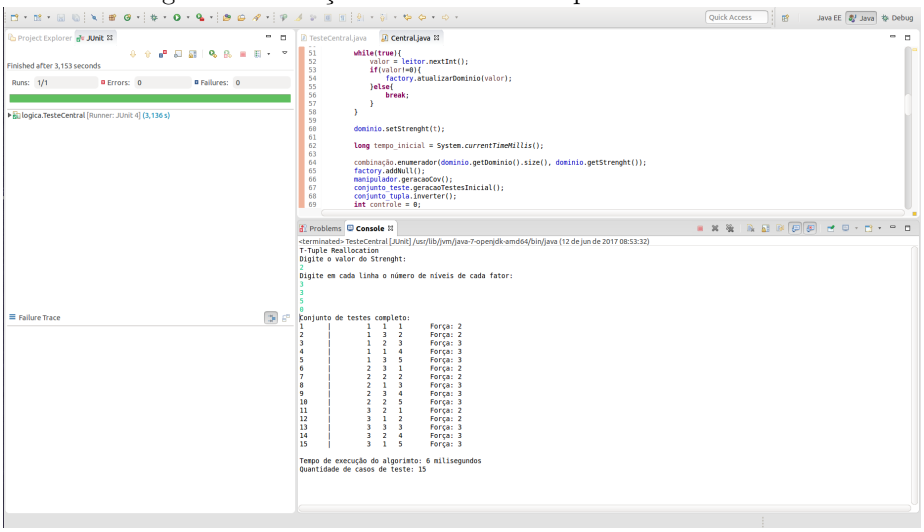
O objetivo desse experimento é testar por meio de *designs* combinatoriais os métodos *linearizeArc* e *assertTolerance* da classe *Circle.java*. Essa classe implementa as funcionalidades e caracteríticas do geo-objeto círculo. O método *linearizeArc* tem o intuíto de que dado 2 pontos definindo um arco no círculo, interpola o círculo para uma coleção de pontos conectados que se aproximam do arco com base no valor de tolerância. O valor de tolerância especifica a distância máxima entre um acorde e o círculo. O método *tolerance* por sua vez, tem o intuito de calcular a distância máxima entre um acorde e o círculo. Com base na identificação de seus fatores, foram definidas as classes de equivalência como ilustrado na Tabela 2.

Table 2: Geração das classes de equivalência a partir da análise do valor limite dos fatores.

métodos	**fatores**	**classes de equivalencia**
linearizeArc	(int) cx	-1, 0, 1
	(int) cy	-1, 0, 1
	(double) radius	-1.0, 0.0, 1.0
	(vector(int)) ordinates	-1, 0, menor 6, 6, maior 6
tolerance	(int) cx	-1, 0, 1
	(int) cy	-1, 0, 1
	(double) radius	-1.0, 0.0, 1.0
	(double) tolerance	-1.0, 0.0, 1.0
	(vector(int)) ordinates	-1, 0, menor 6, 6, maior 6

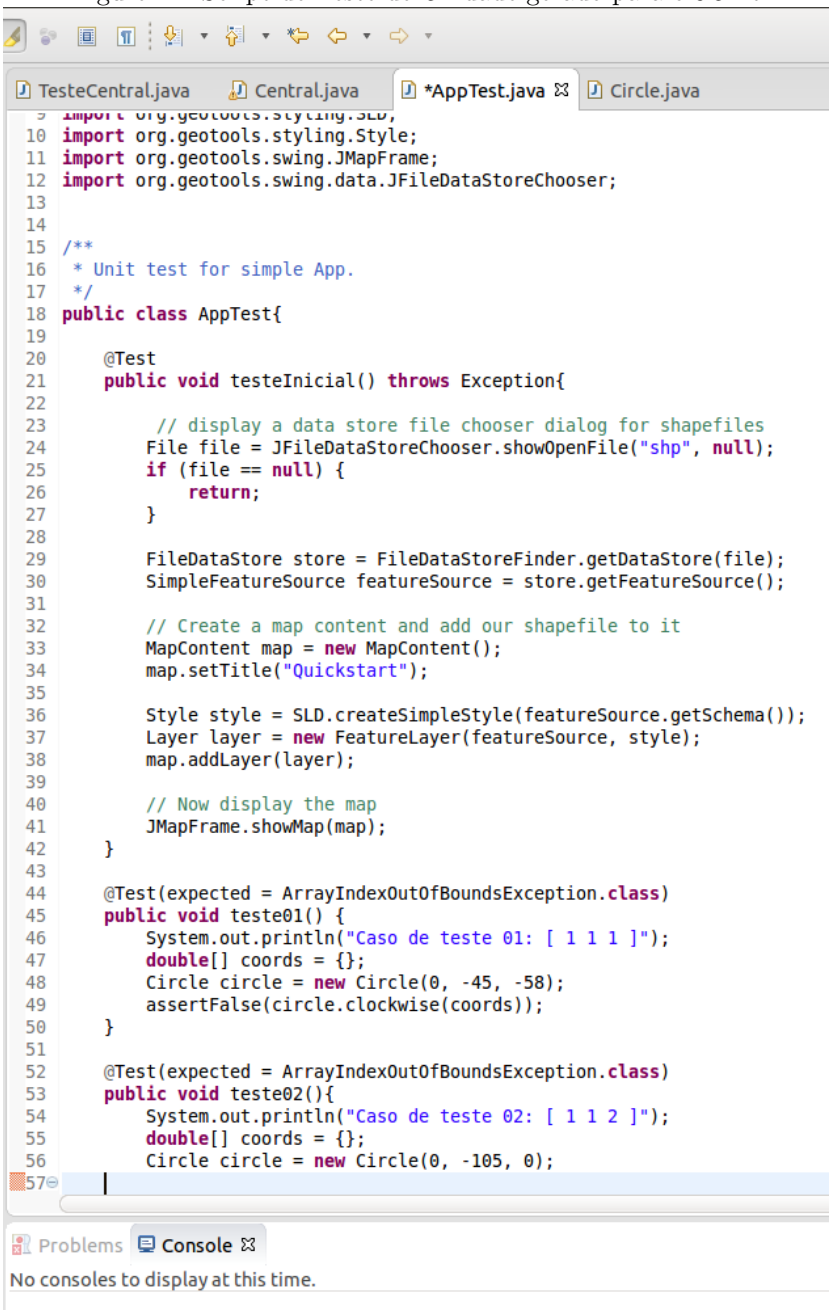
Em seguida, os dados foram submetidos ao TTR, como ilustrado na Figura 10, onde para cada método, foi gerado um conjunto de casos de teste, cada um com 15 instâncias.

Figure 10: Geração de casos de teste por meio do TTR.



Após feito esse passo, é necessário a geração de um *script* de Teste de Unidade por meio da biblioteca Java JUnit. Para isso, precisamos simular um exemplo de operação geográfica que exercite a criação de círculos. Foi utilizada então a operação de sobreposição de mapas. Os arquivos *shapefile* utilizados foram extraídos do repositório http://www.naturalearthdata.com. Onde a simulação ocorre através da criação de amostras em formato de círculo.

Figure 11: Script de Teste de Unidade gerado para o JUnit.



Nenhum dos 30 casos de teste gerados foi rejeitado pelo *script* de teste.

5 Considerações Finais

Esse trabalho mostrou a aplicação da Técnica de Teste de Software Designs Combinatoriais na biblioteca *GeoTools* no contexto do Teste de Unidade. Dada a cuidadosa avaliação da biblioteca, é possível concluir que Sistemas de Informações Geográficos são sistemas complexos, uma vez que são plataforma compostas de diversos módulos que integram uma série de serviços, como serviços web e banco de dados. Dessa forma, é importante que sejam testados de maneira adequada, utilizando um conjunto de técnicas de Teste de Software, de modo a diminuir a possibilidade da existência de falhas.

No contexto desse trabalho, apenas algumas unidades do sistema foram consideradas. Por esse motivo, foi pertinente a utilização de apenas uma única técnica de teste. A classe alvo dos testes foi a classe Circle, sendo as unidades consideradas os métodos *linearizeArc* e *tolerance*. Foi feita análise dos fatores envolvidos a esses métodos e a partir da análise do valor limite, foram definidas as classes de equivalência. Com esses valores em mãos, foi gerada a matriz de dados de teste a partir do algoritmo TTR. Em seguida, para cada um dos dados de teste, que correspondem as entradas que serão submetidas aos programas a serem testados, foi gerado as saídas esperadas. Dessa forma, o Script de teste foi construído por meio do framework Java JUnit 4.

Foram consideradas exceções e retornos das funções e nenhum caso de teste do script de teste foi rejeitado durante a execução. Uma das razões atribuídas a esses resultados pode ser o fato de ser testado apenas dois métodos de apenas uma única classe, o que é muito pouco para testar o software de maneira adequada. No entanto, é suficiente para o intuito desse trabalho, que é mostrar a facilidade da aplicação da técnica de Designs Combinatoriais por parte de profissionais que não tem prática com programação de software, na área de geoprocessamento.

É importante ressaltar que a análise foi feita de maneira superficial, e que em um projeto de teste real, é fundamental que a abordagem de teste seja aplicada de maneira sistemática e rigorosa, inclusive fazendo uso de um conjunto de técnicas de teste, e não de apenas uma única.

Por fim, é possível concluir que a técnica de Designs Combinatoriais é uma técnica fácil de ser aplicada no contexto de geoprocessamento, uma vez que não necessita de um extenso conhecimento do sistema como um todo e do domínio de técnicas de Teste de Software por parte de profissionais que não possuem formação específica na área de computação, mas que tem a necessidade de lidar com tarefas típicas do desenvolvimento de software. Além disso, torna-se um incentivo a esses profissionais a começarem a incluir o abito de testar os artefatos de software produzidos por eles.

References

[1] M. E. Delamaro, J. C. Maldonado, and M. Jino. *Introdução ao Teste de Software*. Elsevier, Brasil, 2007.

[2] A. P. Mathur. *Foundations of Software Testing*. Dorling Kindersley (India), Pearson Education in South Asia, Delhi, India, 2008. 689 p.

[3] V. A. SANTIAGO JÚNIOR. *SOLIMVA: A Methodology for Generating Model-Based Test Cases from Natural Lagrange Requirements and Detecting Incompletness in Sostware Specifications*. PhD thesis, Instituto Nacional de Pesquisas Espaciais (INPE), 2011.

[4] S. R. Dalal, A. Jain, N. Karunanithi, J. M. Leaton, C. M. Lott, G. C. Patton, and B.M. Horowitz. Model-based testing in pratice. In *Proceedings 21st International Conference on Software Engineering*, pages 285–294, Nova York, 1999. AMC.

[5] K. C. Tai and Y. Lei. A test generation strategy for pairwise testing. *IEEE Transactions on Software Engineering*, pages 109–111, 2002.

[6] V. Santiago, W. P. Silva, and N. L. Vijaykumar. Shortening test case execution time for embedded software. In *Proceedings of the 2nd IEEE International Conference SSIRI*, pages 81–88, 2008.

[7] J. M. Balera and V. A. Alexandre de SANTIAGO JÚNIOR. A controlled experiment for combinatorial testing. In *Proceedings...*, pages 1–10. Simpósio Brasileiro de Teste de Software Sistemático e Automatizado, 1. (SAST), 2016. Setores de Atividade: Pesquisa e desenvolvimento científico.

[8] J. M. Balera and V. A. SANTIAGO JÚNIOR. T-tuple reallocation: An algorithm to create mixed-level covering arrays to support software test case generation. In *15th International Conference on Computational Science and Its Applications (ICCSA)*, pages 503–517. Springer International Publishing, Berlin, Heidlberg, 2015.

[9] J. M. Balera and V. A. Santiago Júnior. T-tuple reallocation: An algorithm to create mixed-level covering arrays to support software test case generation. In O. Gervasi, B. Murgante, S. Misra, M. L. Gavrilova, A. M. A. Rocha, C. Torre, D. Taniar, and B. O. Apduhan, editors, *Computational Science and Its Applications – ICCSA 2015*, volume 9158 of *Lecture Notes in Computer Science (LNCS)*, pages 503–517. Springer International Publishing, 2015.