6
Execução das medições

Esse capítulo apresenta os procedimentos realizados para a medição, os resultados dos objetivos do GQM, os cenários de alteração e os resultados obtidos. Além disso, são feitas críticas das medições com relação à confiabilidade. O resultado completo das medições pode ser visto no Apêndice II.

6.1.Coleta dos dados

Em sua maior parte, os dados foram coletados pela ferramenta Borland Together 2008 R2 for Eclipse (TOGETHER, 2010). Essa ferramenta é uma *Integrated Development Tool* (IDE), que realiza a coleta automatizada de diversas métricas, inclusive as: CBO, CC, LCOM (chamada LCOM1 no Together), LOC, NOA, NOC, NOO, WMC (chamada WMPC1 no Together). A Figura 19 apresenta os resultados das métricas na coleta inicial para cada projeto orientado a objetos, e a Figura 20 apresenta os resultados da coleta inicial para cada projeto orientado a agente. Os números em vermelho indicam valores acima do valor máximo recomendado pela ferramenta. Entretanto, esses valores máximos podem ser alterados e esse fato não foi tratado na dissertação.

Resource	CBO	CC	LCOM1	LOC	NOA	NOC	NOO	WMPC1
GeoRiscF-SafetyFactor	62	10	504	1408	21	8	35	52
GeoRisc	52	17	556	4463	23	54	39	103
GeoRiscF-Comparison	36	6	25	297	4	5	10	16
GeoRiscF-QualitativeCombination	32	5	110	512	5	7	16	23
GeoRiscF-Precipitation	31	6	21	367	3	7	10	13
GeoRiscF-Specialist	29	9	102	1276	3	28	34	45

Figura 19 - Medições dos projetos do sistema OO

Resource	CBO	CC	LCOM1	LOC	NOA	NOC	NOO	WMPC1
GeoRiscF-SafetyFactor	64	9	504	1610	24	13	35	37
GeoRisc	52	17	556	5368	23	74	39	103
GeoRiscF-Comparison	33	6	36	509	5	9	12	17
GeoRiscF-QualitativeCombination	34	6	110	797	5	11	16	23
GeoRiscF-Precipitation	35	6	15	585	3	10	11	18
GeoRiscF-Specialist	32	9	96	1472	6	28	34	45

Figura 20 - Medições dos projetos do sistema OA

Dependendo da métrica, os resultados podem se referir ao projeto, a pacotes, a classes e interfaces ou a métodos. As Figuras 19 e 20 apresentam os resultados totais das métricas por projetos ou o valor mais alto de uma métrica em um componente. Para a comparação, foram utilizados os dados das tabelas do Apêndice II, que estão mais detalhados. As Figuras 19 e 20 não apresentam os resultados por classes. Assim, não há nível de detalhe o suficiente para correlacionar elementos arquiteturais de cada uma das tecnologias.

Para coleta da métrica TT foi utilizada a ferramenta *Fixtures for Easy Software Testing* (FEST, 2009). Essa ferramenta é uma coleção de APIs de código aberto que visa simplificar o teste de software. O FEST é composto por diversos módulos e todos podem ser usados com ferramentas de teste unitário.

Como as duas arquiteturas utilizaram um núcleo similar, arquivos de recursos como XML ou propriedades são rigorosamente idênticos, mudando apenas nomes de projetos e URLs.

Como pode ser visto na Figura 21, a comparação realizada entre os dois projetos foi realizada de diferentes formas. Algumas métricas podem medir o tamanho do projeto como um todo, como a LOC, em outros casos foram associados elementos arquiteturais aos resultados das métricas a fim de que elas pudessem ser comparadas, como a CBO e por fim, outras métricas mediram atributos dos projetos e os valores foram comparados, como a TT.

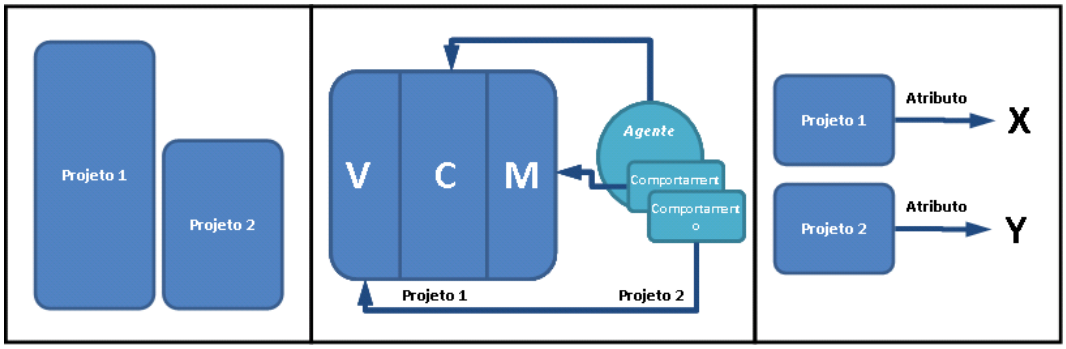

Figura 21 - Tipo de comparações realizadas entre as métricas

6.2. Objetivo Desenvolvimento

Como dito anteriormente, dois sistemas foram construídos para este trabalho, um orientado a objetos e outro orientado a agentes através de JADE. Cada sistema é uma linha de produto que gerou seis projetos, um core e cinco *features*. Após o desenvolvimento dos sistemas o modelo de medição foi aplicado para avaliar o objetivo de desenvolvimento. Nos itens de 6.2.1 à 6.2.4

são apresentados os resultados das métricas selecionadas no modelo GQM. Essas métricas devem responder as seguintes perguntas:

- Qual é a facilidade de compreensão do desenvolvedor na implementação do produto?
- Qual é a facilidade de escrita do desenvolvedor na implementação do produto?

O item 6.2.5 faz um relacionamento do resultado dessas métricas com as perguntas a serem respondidas. Além disso, no item 6.2.6 é feita uma análise de acordo com a confiabilidade das métricas e possíveis ameaças a validade do trabalho.

6.2.1.LOC

No sistema OO foram utilizadas 8323 linhas de código, já o sistema OA utilizou 10341 linhas. Ou seja, para cada linha de código desenvolvida em OO são desenvolvidas 1,24 em OA. Proporcionalmente esse número não “parece” relevante, mas como visto nesse sistema, foram criadas 1909 linhas a mais de um projeto para o outro.

Um motivo desse número superior de linhas é a necessidade de classes de ontologia no SMA e outro motivo é a necessidade de alguns procedimentos pré e pós envio de mensagens entre agentes. Durante o desenvolvimento foram identificados esses procedimentos e para a redução do esforço de implementação, foram criados métodos e heranças de forma a reusar os procedimentos. Desse fato podemos concluir que: é bom, de modo que, são criados mecanismos para o reuso no sistema; é ruim, pois, esses procedimentos devem ser identificados pelo desenvolvedor. Além disso, o número de linhas de código foi maior mesmo com a utilização do reuso nesses procedimentos na aplicação OA.

No entanto, como dito no item 5.1.1 essa métrica não pode ser considerada um indicador comparável de esforço.

6.2.2.NOO

No sistema OO foram utilizados 728 métodos, já no sistema OA foram utilizados 763 métodos. Esses números são bem próximos, proporcionalmente a cada método do sistema OO 1,05 métodos são feitos no sistema AO. Isso nos

mostra que, em relação ao número de métodos das classes, os dois sistemas se equivalem.

No entanto, como temos os dados de outras métricas, podemos chegar a outras conclusões. Como a diferença entre o número de LOC é considerável e a diferença entre a quantidade de métodos é menor, podemos deduzir que, os métodos do sistema OA são maiores do que os métodos do sistema OO. Esse fato pode influenciar na facilidade de leitura do código, visto que, métodos maiores podem ser mais complexos ou podem dificultar a compreensão e a facilidade de leitura do código. Entretanto, vale ressaltar que esse dado não é conclusivo de modo que, há a possibilidade de métodos expressos em mais linhas serem mais fáceis de compreender do que outros métodos expressos em menos linhas.

6.2.3.NOC

No sistema OO foram criadas 109 classes, já no sistema OA foram criadas 145 classes. No sistema OA as classes de controle do padrão MVC foram incorporadas ao agente de interface, no entanto diversas outras classes surgiram para representar os comportamentos dos agentes. Cada agente pode implementar um ou mais *behaviours*, comportamentos, e cada *behaviour* vira uma classe ou uma classe interna, *inner class*. Em média cada agente implementa três comportamentos, resultando em um crescimento significativo no número de classes.

Esse número alto na quantidade de classes do sistema OA pode ser ruim para a manutenção dos sistemas, visto que, quanto maior é o número de classes, mais o desenvolvedor deve se preocupar em saber o vocabulário do sistema, a localização de funcionalidades, entre outros. Nesse contexto, existem pesquisas sobre troca de contexto em projetos (KERSTEN, 2005), no qual o desenvolvedor gasta um intervalo de tempo para atingir um grau de interesse. A soma desses intervalos se torna uma perda de tempo significante com o tempo. Dessa forma, quanto maior o número de classes mais tempo será gasto para atingir o grau de interesse.

6.2.4.WMC

No sistema OO a complexidade somada das classes dos projetos foi de 1262, enquanto no sistema OA a métrica apontou uma complexidade com o

valor de 1414. Ou seja, cada classe desenvolvida em OA é 1,1 vezes mais complexa do que em OO.

Analisando todos os dados das outras métricas mais o da métrica WMC é possível visualizar que no sistema OA foram utilizadas mais linhas de código, mais métodos, mais classes e estas classes obtiveram uma complexidade maior do que no sistema OO.

6.2.5.Análise dos resultados

A métrica LOC foi selecionada, pois a quantidade de linhas do código pode afetar na facilidade de compreensão e na facilidade de escrita. Segundo o resultado dessa métrica o sistema OO necessita de menos linhas de código, indicando que o sistema OO pode ser mais fácil de escrever e de compreender.

A métrica NOO foi selecionada, pois a quantidade de operações pode afetar na facilidade de compreensão e na facilidade de escrita. Segundo o resultado dessa métrica o sistema OO necessita de menos operações, indicando que o sistema OO pode ser mais fácil de escrever e de compreender.

A métrica NOC foi selecionada, pois a quantidade de classes pode afetar na facilidade de compreensão e na facilidade de escrita. Segundo o resultado dessa métrica o sistema OO necessita de menos classes, indicando que o sistema OO pode ser mais fácil de escrever e de compreender.

A métrica WMC foi selecionada, pois a complexidade das classes pode ser um indicador da complexidade de escrita e compreensão da mesma. Segundo o resultado dessa métrica o sistema OO contém menos classes e essas classes são menos complexas do que o sistema AO. Esse fato indica que o sistema OO pode ser mais fácil de escrever e de compreender.

No desenvolvimento do software todas as métricas indicaram que o sistema orientado a objeto é mais fácil de compreender e escrever, demonstrando ter um desempenho melhor para o desenvolvimento do que um sistema orientado a agentes.

6.2.6.Ameaças a validade

A maioria das métricas utilizadas podem apontar para falsos positivos ou falsos negativos. Ou seja, as métricas podem indicar um valor desejado, mas outras circunstâncias podem invalidar esse resultado. Entretanto, esses falsos positivos são casos pontuais, exceções, que as métricas não podem contornar.

No objetivo de desenvolvimento na métrica LOC pode ocorrer do desenvolvedor necessitar de um esforço maior para implementar uma funcionalidade com poucas linhas em uma linguagem, do que, implementar a mesma funcionalidade em outra linguagem mesmo que usando mais linhas para tal.

Na métrica NOO em alguns casos, sistemas com poucas operações e com muitos parâmetros podem ser mais complexos do que sistemas com mais operações e menos parâmetros. Dessa forma, o desenvolvedor necessitaria de um esforço maior para desenvolver o sistema com poucas operações em vista da grande complexidade dessas operações. No caso de sistemas com muitas operações simples como em operações *gets* e *sets* o desenvolvedor tem um esforço baixo.

Como nas outras métricas a NOC pode levar a conclusões errôneas. Um número maior de classes pode indicar que o sistema está mais coeso. Ou seja, um número grande de classes altamente coesas é mais desejável do que um pequeno número de classes pouco coesas.

Com o conjunto selecionado no plano de medições é mais provável que a maior parte das métricas apontem para resultados reais e não que todas apontem para falsos positivos, pois, como dito anteriormente, são exceções.

6.3.Objetivo Operação

Assim como o objetivo de desenvolvimento, o objetivo de operação foi analisado após a construção dos dois sistemas. Nos itens de 6.3.1 e 6.3.2 são apresentados os resultados das métricas selecionadas no modelo GQM. Essas métricas devem responder as seguintes perguntas:

- Quanto tempo demora-se para completar uma tarefa?

O item 6.3.3 faz um relacionamento do resultado dessas métricas com as perguntas a serem respondidas. Além disso, no item 6.3.4 é feita uma análise de acordo com a confiabilidade das métricas e possíveis ameaças a validade do trabalho.

Os passos necessários para executar as tarefas selecionadas para as métricas Tempo da Tarefa e Freqüência de Falha estão descritas e ilustradas em detalhes no Apêndice I. Somente a funcionalidade de Inventário não foi considerada nessa dissertação.

6.3.1.TT

Para esta métrica foi selecionado um caso de uso para cada *feature* da plataforma. E cada caso de uso foi executado dez vezes para minimizar resultados muito dispersos do esperado.

A execução do caso de uso foi realizada pela ferramenta FEST (2009), que simula os passos de uma interação com a interface do sistema, valida as respostas do sistema e informa o tempo de execução. A FEST pode ser integrada com ferramentas de teste unitário, como o (JUNIT) que foi utilizado nesta dissertação. Assim, caso houvesse alguma falha durante a execução de algum caso de uso a falha seria capturada e informada pelo JUnit.

Para a *feature* de Precipitação a ferramenta executou os seguintes passos:

- Adicionou três planos de informação,
- Informou qual era o tipo de cada plano de informação, o contorno, a chuva do dia atual e a chuva dos quatro dias passados,
- Pediu para gerar o mapa de susceptibilidade,
- Verificou se o resultado era o esperado.

Para a *feature* de Combinação Qualitativa a ferramenta executou os seguintes passos:

- Adicionou dois planos de informação,
- Informou o peso de cada plano de informação. Quatro para o uso e ocupação do solo e seis para a declividade,
- Pediu para gerar o mapa de susceptibilidade,
- Verificou se o resultado era o esperado.

Para a *feature* de Comparação a ferramenta executou os seguintes passos:

- Adicionou o plano de informação contendo o inventário,
- Pediu para fazer a comparação entre os mapas de susceptibilidades gerados e o mapa de inventário,
- Verificou se o resultado era o esperado.

Para a *feature* de Fator de Segurança a ferramenta executou os seguintes passos:

- Informou os dados do solo,
- Solicitou a adição do solo,
- Informou os dados de um evento de precipitação,
- Solicitou a adição do evento de precipitação no solo adicionado,

- Pediu para gerar o gráfico do fator de segurança do solo,
- Verificou se o resultado era o esperado.

Para a *feature* do Especialista a ferramenta executou os seguintes passos:

- Requisitou a recomendação,
- Informou os planos de informação e a escada dos planos de informação,
- Requisitou o resultado,
- Verificou se o resultado era o esperado.

As mesmas tarefas foram executadas para cada sistema. A seguir são apresentadas a tabelas com os tempos das tarefas para cada sistema.

Tabela 8 - Sistema OO, tempo em segundos

Execução Feature	1	2	3	4	5	6	7	8	9	10	Média
Precipitação	84,067	107,678	127,822	78,051	77,566	137,007	76,785	77,488	78,910	89,802	93,518
Combinação Qualitativa	142,508	102,849	122,490	437,257	350,893	129,507	480,451	528,669	190,384	523,543	300,885
Comparação	391,173	352,648	337,246	436,252	333,183	353,357	353,123	386,232	414,184	347,369	370,447
Fator de Segurança	7,344	7,265	7,188	10,015	9,015	7,266	10,906	7,422	10,718	7,172	8,431
Especialista	8,063	7,203	7,328	8,312	9,063	9,703	9,047	7,656	7,453	8,500	8,828

Tabela 9 Sistema SMA, tempo em segundos

Execução Feature	1	2	3	4	5	6	7	8	9	10	Média
Precipitação	290,468	131,835	139,555	128,413	75,551	75,676	600,000	600,000	80,286	85,301	220,709
Combinação Qualitativa	238,450	420,678	146,447	136,913	118,882	128,741	156,962	169,852	259,529	145,882	192,228
Comparação	366,003	379,426	355,924	448,539	385,973	350171	414,649	424,345	442,410	475,552	404,299
Fator de Segurança	29,546	12,843	10,968	11,797	14,859	11,077	11,047	14,015	11,718	8,859	13,673
Especialista	15,921	21,202	27,890	15,609	13,906	19,015	17,172	10,672	16,079	11,214	16,868

Como pode ser visto nas Tabelas 8 e 9 a implementação OO obteve melhores tempos na média das tarefas em quase todas as *features*.

As execuções sete e oito da *feature* de precipitação da implementação em SMA não foram finalizadas. Esse fato ocorreu, pois foi definido um tempo limite para a execução das tarefas de dez minutos. As duas execuções atingiram o tempo limite e ficaram com o tempo final de dez minutos.

Vários eventos podem ter causado os tempos piores da implementação em SMA em relação a implementação em OO, alguns exemplos são a comunicação assíncrona entre os agentes, a comunicação entre o agente de interface e a

interface gráfica do usuário e o escalonamento das *threads* de controle dos agentes e da interface gráfica.

No caso das execuções sete e oito da *feature* de precipitação da implementação em SMA muitas hipóteses podem ser levantadas por isso esses casos serão desconsiderados. No entanto, se essas duas exceções não tivessem ocorrido provavelmente os tempos da execução em OO e em SMA seriam muito próximos. Nesse caso, pode-se observar um padrão. As *features* de precipitação, combinação qualitativa e comparação precisam do carregamento de mapas, fazem uma solicitação simples de serviço e recebem respostas simples. No caso da *feature* de combinação qualitativa ficou mais evidente que a utilização de agentes foi eficiente, pois os mapas carregados nessa *feature* são mais complexos e a separação em *threads* dividiu as tarefas de desenhar os mapas e fazer cálculos nos mapas. Vale ressaltar que em casos de mapas simples a implementação OO é melhor do que a de agentes como pode ser visto nas Tabelas 8 e 9.

Nos casos em que os agentes deveriam atualizar a interface do usuário com mais freqüência ou alterações maiores, a implementação OO obteve melhores resultados. Esse fato ocorreu, pois há um *delay* na requisição do agente de interface para atualização da interface de modo que são duas *threads* diferentes. Já na implementação OO a execução está sobre controle da *event dispatch thread* e como não há a necessidade de comunicação entre threads esse *delay* não existe.

Durante o desenvolvimento dos passos para cada *feature* foram encontrados alguns defeitos na implementação SMA. Assim, a implementação em SMA teve que sofrer algumas modificações para corrigir esses defeitos. Os defeitos encontrados no desenvolvimento dos testes não foram coletados, mas consistiam basicamente em perda de dados informados na interface e múltiplas chamadas para serviços da ferramenta (ArcGIS). O primeiro caso foi solucionado através da atualização da instância da GUI, que o agente de interface possui. Essa atualização passou a ser realizada a cada evento gerado pela própria GUI, ou o pré-processamento dos dados na GUI e envio dos dados para os agentes através de eventos. No segundo caso, como a ferramenta não aceita múltiplas requisições, o acesso aos recursos foi controlado.

6.3.2.Análise dos resultados

Externamente, a implementação em OO bloqueia a interface sendo liberada somente após o processamento dos dados, no caso da implementação em SMA a interface fica liberada para ser utilizada pelo usuário mesmo com dados processando em paralelo.

Internamente na arquitetura OO operações realizadas a partir de eventos gerados pela interface são diretamente executados pela *event dispatch thread*. Na arquitetura OA eventos gerados pela interface são tratados pelas *threads* de controle dos agentes podendo ser repassadas para outras *threads*.

Quanto a pergunta do modelo de medição sobre o tempo que demora para completar uma tarefa a resposta depende do caso. Se o processamento de um evento gerado pela interface é pequeno ou seqüencial, ou a resposta do processamento faz várias alterações na interface é melhor utilizar a arquitetura OO. Se a interface precisa desenhar objetos com muitas informações e o processamento de eventos gerados pela interface é grande ou paralelo é melhor utilizar a arquitetura OA.

No caso em que a tecnologia OO foi melhor, provavelmente esse fato ocorreu pois o processamento dos eventos gerados pela interface são diretamente executados pela *event dispatch thread*. Como essa thread é responsável pela interface em Java, atualizações de interface freqüentes se tornam mais eficientes.

No caso em que a tecnologia AO foi melhor, provavelmente esse fato ocorreu devido ao *multi-threading* inerente à tecnologia AO, que acaba ocupando melhor a CPU durante as esperas de resposta de periféricos. Ou seja, quando há uma separação de interesses como a apresentação de um mapa grande no monitor e o cálculo das informações contidas no mapa, a tecnologia AO se adéqua melhor pois uma *thread* fica responsável pela resposta da apresentação do mapa no monitor e outra *thread* fica processando as informações do mapa na CPU.

6.3.3.Ameaças a validade

Há apenas uma métrica utilizada para o objetivo operação. Apenas essa métrica pode não ser o suficiente para entender todos os comportamentos relacionados ao objetivo de operação das plataformas.

A métrica Tempo de Tarefa é útil para verificar o tempo médio gasto para realizar uma tarefa, mas essa métrica deveria ser aplicada a todos os casos de uso do sistema e não somente aos casos de uso principais. Essa atividade não foi realizada em virtude de falta de tempo e de recursos.

6.4.Objetivo Manutenção

O objetivo manutenção foi dividido em duas fases. Primeiro foram coletadas as métricas definidas no modelo de medição. Nos itens de 6.4.1 à 6.4.6 são apresentados os resultados das métricas selecionadas no modelo GQM. Essas métricas visam responder as perguntas:

- Qual é o esforço do desenvolvedor na compreensão do produto?
- Qual é o esforço do desenvolvedor na alteração do produto?

Na segunda fase foram feitas modificações nos dois sistemas de acordo com dois cenários e essas alterações foram analisadas conforme apresentado no item 6.4.7.

O item 6.4.8 faz um relacionamento do resultado dessas métricas com as perguntas a serem respondidas. Além disso, no item 6.4.9 é feita uma análise de acordo com a confiabilidade das métricas e possíveis ameaças a validade do trabalho.

6.4.1.CBO

O acoplamento entre objetos é avaliado entre as classes do sistema. Dessa forma, foram comparadas classes com funcionalidades semelhantes dos dois sistemas. Como os agentes de interface da implementação OA realizam o papel de visão e controle, foram somados os resultados das classes *View* e *Controller* do sistema OO para serem comparadas com as classes dos comportamentos e dos agentes de interface de cada *feature*. As classes Model do sistema OO foram comparadas com os comportamentos dos agentes especialistas nos modelos.

Em todos os casos os agentes de interface OA tiveram um acoplamento ligeiramente mais alto do que as classes *View* e *Controller* do sistema OO somadas. Já nas classes de modelo as diferenças foram mais significantes. Todas as classes *Model* do sistema OO obtiveram um acoplamento em média quatro vezes menor do que os comportamentos dos agentes da implementação OA que realizam as funcionalidades referentes aos modelos do sistema OO.

6.4.2.LCOM

Assim como o acoplamento entre objetos a falta de coesão nos métodos foi avaliada através das classes com funcionalidades semelhantes dos dois sistemas. Na LCOM um número baixo indica alta coesão entre os métodos da classe.

As classes controle e visão dos dois sistemas tiveram valores médios, ou seja, nem baixa nem alta coesão. Apenas um resultado foi mais elevado, esse resultado foi o de fator de segurança do sistema OO. Isso ocorreu, pois esse modelo precisa armazenar muitos valores através de métodos *get* e *set* e esses métodos acabam aumentado os valores da LCOM, pois se relacionam a somente um atributo da classe. No sistema OO esses métodos ficaram no *Controller*, e no sistema OA esses métodos não são necessários, pois os esses valores são passados para os agentes via eventos.

Em geral as classes controle e visão do sistema OA tiveram valores mais baixos do que as classes do sistema OO. Indicando a melhor coesão entre as classes de controle e visão do sistema OA. Como comentado anteriormente esse fato é motivado pela comunicação entre o agente de interface e a interface via eventos e não métodos *gets* e *sets*.

Já as classes de modelo os resultados foram mais interessantes. As classes do sistema OO que realizam operações mais simples obtiveram melhores resultados do que as classes do sistema OA. No entanto, as classes do sistema OO que realizam muitas operações obtiveram piores resultados do que o sistema OA. Esse fato pode ser um indicador que o sistema OA pode lidar naturalmente melhor com tarefas complexas do que o sistema OO.

No entanto, como dito anteriormente, essa métrica é muito criticada na literatura e nesse trabalho foi encontrado um problema que é o caso dos *gets* e *sets* que encapsulam atributos de uma classe, Entretanto, essa métrica entende esses métodos como métodos não coesos na classe.

6.4.3.NOA

As medições do número de atributos dos sistemas indicaram que em média no sistema OA as classes possuem 0,1 atributos a mais do que no sistema OO. Esse fato ocorre, pois, os comportamentos dos agentes necessitam de algumas informações contidas nos agentes, assim, é necessário que os

agentes instanciem comportamentos passando parâmetros e os comportamentos armazenem esses parâmetros.

6.4.4.WMC

Como apresentado no item 6.2.4 do objetivo desenvolvimento a métrica WMC apresentou melhores resultados para o sistema OO do que o sistema OA. Tendo classes mais simples a implementação OO pode necessitar de menos esforço de alteração em uma manutenção do que a implementação OA.

6.4.5.NOO

Como apresentado no item 6.2.2 do objetivo desenvolvimentos a métrica NOO apresentou melhores resultados para o sistema OO do que o sistema OA. Tendo menos métodos a implementação OO pode necessitar que menos métodos do sistema necessitem serem compreendidos e alterados em uma manutenção do que a implementação OA.

Como dito anteriormente, dependendo de uma arquitetura ou linguagem o número de métodos a serem alterados pode variar. Uma arquitetura que necessite de menos métodos do que outra pode em uma manutenção precisar que mais métodos sejam alterados do que outra arquitetura que necessite de mais métodos.

6.4.6.NOC

Como apresentado no item 6.2.2 do objetivo desenvolvimentos a métrica NOC apresentou melhores resultados para o sistema OO do que o sistema OA. Tendo menos classes a implementação OO pode necessitar que menos classes do sistema necessitem serem compreendidas e alteradas em uma manutenção do que a implementação OA.

Como dito anteriormente, dependendo de uma arquitetura ou linguagem o número de classes a serem alteradas pode variar. Uma arquitetura que necessite de menos classes do que outra pode em uma manutenção precisar que mais classes sejam alteradas do que outra arquitetura que necessite de mais classes.

6.4.7.Cenários de manutenção do sistema

Para avaliar melhor a manutenibilidade dos dois sistemas, foram desenvolvidos cenários de alteração, evolução, para serem aplicados nos sistemas. A partir dessas alterações os sistemas foram medidos e analisados.

6.4.7.1.Cenário 1 – Precipitação automática

Nesse cenário, o modelo de precipitação deve ser executado sem a intervenção de um usuário. O sistema deve obter os dados de precipitação diariamente a partir do site da GEORIO (2003), fazer o tratamento desses dados, gerar um mapa de precipitação e gerar a susceptibilidade de escorregamentos de acordo com o modelo de precipitação.

6.4.7.2.Cenário 2 – Parâmetros do modelo de combinação qualitativa

Nesse cenário, o modelo de combinação qualitativa deve ser executado sem a entrada dos pesos das camadas. Dessa forma o modelo deve testar valores e validar usando o modelo de comparação com o inventário de escorregamentos na área das camadas informadas. Assim os dois modelos devem se comunicar a fim de encontrar o valor ideal para o modelo de combinação qualitativa.

6.4.7.3.Resultados dos cenários de manutenção do sistema

Os cenários de manutenção do sistema provocaram adições ou modificações de linhas de código, de classes, entre outros. A seguir são discutidas essas alterações com relação a cada cenário.

No Cenário 1 na implementação OO foram adicionadas 19 linhas de código em classes já existentes e foi criada uma classe nova que totalizou 49 linhas de código. Na implementação OA foram adicionadas 13 linhas de código e foram criadas duas novas classes que contêm 63 linhas.

As modificações feitas para o Cenário 1 na implementação OO foram a adição de um método que totaliza 12 linhas no núcleo da aplicação para prover uma nova funcionalidade de editar mapas. Essa alteração também foi realizada na implementação OA. Outra modificação realizada na implementação OO foi a adição de uma *Thread* que realiza a tarefa de gerar a precipitação automaticamente. A adição dessa *Thread* para o modelo de precipitação custou

7 linhas, enquanto no modelo OA foi adicionado um comportamento para o agente de interface que realiza as mesmas operações da *Thread* da implementação OO. Essa adição na implementação OA custou uma linha que justamente é a adição de um comportamento.

Tanto a *Thread* da implementação OO quanto o comportamento da implementação OA são ativados por um evento temporal. Após a ativação as duas acessam o site da GEORIO (2003), colhem informações do site, fazem o tratamento dessas informações e requisitam para os seus respectivos serviços de geração de mapa de susceptibilidade.

As alterações das duas implementações no Cenário 1 foram bem pontuais, ou seja, não foram espalhadas pelo sistema. Entretanto, duas características devem ser observadas. A primeira é que a adição de um novo componente no sistema OA foi menos custosa, resumindo-se à adição de um comportamento para um agente. Enquanto a codificação desse novo componente foi menos custoso no sistema OO.

No Cenário 2 na implementação OO foram adicionadas 57 linhas de código em classes já existentes, foram alteradas sete linhas e não foi adicionada nenhuma classe nova. Na implementação OA foram adicionadas 36 linhas de código, foram alteradas 5 linhas e foi criada uma nova classe que contêm 169 linhas.

As modificações feitas para o Cenário 2 na implementação OO foram a adição de 7 linhas na classe de visão representando um novo botão na interface, a adição de 4 linhas na classe de controle e alteração de 7 linhas na mesma classe para capturar o evento gerado pelo novo botão e diferenciar de outros eventos. No modelo foram adicionadas 46 linhas para realizar as funções necessárias para o Cenário 2.

Na implementação OA foram adicionadas as mesmas 7 linhas na classe de interface representando um novo botão, nessa classe também foram adicionadas mais 6 linhas para empacotar o evento gerado pela interface para ser enviado para o agente de interface e 4 linhas foram alteradas para diferenciar o evento gerado pelo novo botão dos outros eventos. Uma linha foi adicionada no vocabulário do agente de interface e essa agente recebeu mais 20 linhas para tratar o evento gerado pela interface. O agente de serviço recebeu duas novas linhas que representam a adição de um comportamento novo e um *template* de mensagem que esse comportamento é responsável por tratar. O novo comportamento realiza as operações necessárias para o Cenário 2.

As alterações das duas implementações no Cenário 2 não foram tão pontuais como as alterações do Cenário 1, ou seja, foram espalhadas pelo sistema. As alterações foram feitas em todas as camadas das duas implementações. As alterações feitas na camada de visão-controle da implementação OO se espalharam menos do que as alterações da implementação OA. Somente dois pontos[4] foram alterados na implementação OO enquanto na implementação OA foram alterados quatro pontos diferentes.

Na camada de modelo as duas implementações tiveram poucas alterações, entretanto, algumas características devem ser observadas. A primeira é que a adição de um novo componente no sistema OA foi pouco custosa, se resumindo a adição de um comportamento para um agente, assim como no Cenário 1. Na implementação OO foram adicionados três novos métodos no modelo com somente um visível para as outras camadas. A alteração na implementação OO foi pequena, mas para acessar as funcionalidades da *feature* de comparação a biblioteca da *feature* teve que ser adicionada a biblioteca da combinação qualitativa e um modelo de comparação teve que ser instanciado pelo modelo da comparação qualitativa. Enquanto na implementação OA essas modificações não foram necessárias, se resumindo a envios de mensagens do agente de combinação qualitativa para o agente de comparação.

6.4.8.Análise dos resultados

A métrica CBO foi selecionada, pois o acoplamento entre objetos pode afetar na facilidade de compreensão e na facilidade de alteração. Segundo o resultado dessa métrica o sistema OO tem acoplamento menor, indicando que o sistema OO pode ser mais fácil de alterar e de compreender.

A métrica LCOM foi selecionada, pois a coesão dos objetos pode afetar na facilidade de compreensão e na facilidade de alteração. Porém a métrica é bastante criticada na literatura. Em particular uma das críticas se refere aos erros de medição no caso de getters e setters. Essas deficiências da métrica LCOM põem em dúvida a validade de quaisquer conclusões que a envolvam. Segundo o resultado dessa métrica a camada de visão-controle do sistema OA tem coesão maior do que o sistema OO, enquanto na camada de modelo *features* com operações mais simples tiveram coesão maior na implementação OO e

4 Ponto representa uma área próxima, ou seja, linhas subseqüentes. Se uma alteração foi feita

features mais complexas tiveram coesão maior na implementação OA. Esses resultados indicam que em geral o sistema OA pode ser mais fácil de alterar e de compreender. No entanto, esse resultado não é conclusivo, em virtude das deficiências da LCOM.

A métrica NOA foi selecionada, pois o número de atributos pode afetar na facilidade de compreensão e na facilidade de alteração. Segundo o resultado dessa métrica o sistema OO tem menor número de atributos, indicando que o sistema OO pode ser mais fácil de alterar e de compreender.

A métrica WMC foi selecionada, pois a complexidade das classes pode ser um indicador da complexidade de alteração e compreensão da mesma. Segundo o resultado dessa métrica o sistema OO contém menos classes e essas classes são menos complexas do que o sistema AO. Esse fato indica que o sistema OO pode ser mais fácil de alterar e de compreender.

A métrica NOO foi selecionada, pois a quantidade de operações pode afetar na facilidade de compreensão e na facilidade de alteração. Segundo o resultado dessa métrica o sistema OO necessita de menos operações, indicando que o sistema OO pode ser mais fácil de alterar e de compreender.

A métrica NOC foi selecionada, pois a quantidade de classes pode afetar na facilidade de compreensão e na facilidade de alteração. Segundo o resultado dessa métrica o sistema OO necessita de menos classes, indicando que o sistema OO pode ser mais fácil de alterar e de compreender.

No Cenário 1 de manutenção do sistema os mesmos pontos foram alterados nas duas implementações e o mesmo número de componentes foi adicionado às duas implementações. Entretanto, foram necessárias menos linhas de código na implementação OO para a alteração do sistema.

No Cenário 2 de manutenção do sistema mais pontos foram alterados na implementação OA e um componente foi adicionado na implementação OA enquanto nenhum foi adicionado na implementação OO. Além disso, foram necessárias menos linhas de código na implementação OO para a alteração do sistema. Entretanto a implementação OO criou uma dependência explícita da *feature* de combinação qualitativa com a de comparação.

No objetivo manutenção do software a maioria das métricas e cenários de alteração indicaram que o sistema orientado a objeto é mais fácil de compreender e alterar, demonstrando ter um desempenho melhor para a manutenção do que um sistema orientado a agentes.

---

em outro método ou em outra classe foi considerado como um novo ponto.

6.4.9.Ameaças a validade

Como comentado no objetivo desenvolvimento, a maioria das métricas utilizadas podem apontar para falsos positivos ou falsos negativos. No entanto foi selecionado um conjunto de métricas para minimizar o aparecimento de eventuais falsos positivos ou falsos negativos.

A medição de acoplamento e coesão de sistemas não é uma tarefa simples. As métricas CBO e LCOM visam auxiliar na medição dessas duas qualidades dos sistemas, entretanto, essas métricas não são as únicas e não são uma bala de prata em relação a medição dessas duas qualidades. Como em outras métricas essas duas podem apontar falsos positivos ou falsos negativos.

A métrica NOA mede o número de atributos. A quantidade de atributos de um sistema pode não ser um complicador na manutenção ou no entendimento do sistema. Uma arquitetura com menos atributos que outra pode necessitar que esses atributos sejam modificados com mais freqüência que outra arquitetura que necessite de mais atributos.

A métrica NOO mede o número de métodos. A quantidade de métodos de um sistema pode não ser um complicador na manutenção ou no entendimento do sistema. Uma arquitetura com menos métodos que outra pode necessitar que esses métodos sejam modificados com mais freqüência que outra arquitetura que necessite de mais métodos.

A métrica NOC mede o número de classes. A quantidade de classes de um sistema pode não ser um complicador na manutenção ou no entendimento do sistema. Uma arquitetura com menos classes que outra pode necessitar que essas classes sejam modificadas com mais freqüência que outra arquitetura que necessite de mais classes.