**Projeto de Pesquisa - Docs(Apresentações)**

**PRIMEIRA APRESENTAÇÃO**

“A sorte favorece só a mente preparada” (Isaac Asimov)

“A ciência se compõe de erros que, por sua vez, são os passos até a verdade” (Julio Verne)

**1 - Qualidade & Performance**

**1.1. Visão Inicial**

Um software que tem um desempenho ruim após o término dos testes de

performance este problema será resolvido 100% caso. Entretanto sempre existirão gargalos que serão descobertos somente com o tempo entretanto se for aplicado de forma correta o saberemos as limitações de um sistema.

O crescimento dessa área é proporcional ao crescimento da Web, pois com isso haverá cada vez mais usuários, tornando assim uma necessidade enorme de atender a esses clientes de forma mais rápida e eficiente. Este de maneira mais rápida se refere à menor quantidade de tempo, ou seja, um melhor desempenho do software.

**Exemplo de performance**

Fazendo analogia com um mercado, onde que para efetuar a compra é necessário pegar uma fila e aguardar até que o caixa esteja disponível. O desempenho do mercado está totalmente ligado ao desempenho dos caixas e da quantidade de caixas disponíveis, ou seja, quanto maior a quantidade de caixas melhor será a performance.

E quanto mais rápido o caixa atende é melhor para o cliente que não quer ficar esperando na fila, quando existe filas muito grandes significa que os caixas estão ocupados e assim ocorrerá uma demora no atendimento que seria um problema no desempenho.

Com isso concluíse que a loja virtual deverá funcionar da mesma forma que á local, onde consiga ter várias pessoas olhando os ítens, fazendo compra com mais eficácia do que a física, onde a velocidade e simplicidade se tornam atrativos para a realização da compra.

Como na loja física na loja virtual também terão momentos onde a loja estará vazia e momentos que ela estará cheia e o sistema deve suportar ambas situações. Em momentos que a loja fica cheia é onde aparece os defeitos relacionados à performance que são os gargalos, ou seja, a demora no atendimento pode ocasionar a perca de um cliente, normalmente os clientes não reclamam simplesmente saem.

No decorrer do dia mais e mais pessoas irão entrar no site, tendo horários de píco e entender esse comportamento é necessário para fazer os testes e verificar onde estão os gargalos do sistema ou seja locais onde o mesmo apresenta lentidão.

A lentidão do site é ocasionada devido à limitação dos componentes que suportam o acesso e com a maior quantidade de usuários os mesmo ficam sobrecarregados.

O teste de performance mostra como o site se comporta em determinadas quantidades de clientes/usuários acessando o mesmo, ou seja, mostra o tempo de resposta.

**Necessidade do teste de performance**

Quando se fala de software 2 aspectos são marcados como os mais importantes onde estes são:

* **O sistema funcionar conforme o proposto:** Isso quer dizer que o sistema atenda à todas exigências do comprador ou seja do cliente/usuário, garantir esta parte é a função dos testes funcionais.
* **O sistema ser rápido:** Esta parte é onde entra a necessidade dos testes de performance, aonde através do mesmo é possível garantir que o sistema tenha a velocidade que precisa ter.

O ser humano por natureza não tem paciência para esperar que uma página web aparecer por completa, caso a demora aconteça o site seria penalizado através do aborrecimento e saída do usuário do mesmo.

Visando isso o teste de performance está cada vez sendo mais cobrado e assumindo uma maior importância dentro do cenário da construção deixando de ser um mero detalhe e se tornando uma real necessidade. O site pode ser o mais funcional do mundo, mas se ele não for rápido não vai ter usuários.

Um exemplo de site com alta performance é o Google, onde o sucesso é baseado em inovação(publicidade) e alta performance. E o segredo da mesma é monitorar sempre e testar.

**Evolução dos testes de performance**

Antigamente somente as grandes empresas possuíam ferramentas adequadas para a realização destes testes. Atualmente a popularização de ferramentas *OpenSource* como o JMeter e o OpenSta abriu-se a porta para a criação de outras automatizadas surgisem.

Hoje em dia também temos ferramentas pagas com um maior poderio de eficácia, também temos conceitos como o SLA (*Service Level Agreement)* ou ANS (Acordo de Nível de Serviço), onde é um **acordo** devido à ser um contrato entre duas partes sendo essas o cliente e ou prestador de serviços, **nível** por que estamos falando sobre algo não palpável ou seja um serviço e para caracterizar o mesmo como bom ou ruim é necessário que algo diga isso, ou seja, um indicador de qualidade, já um **serviço** é algo como um conjunto de facilidades que ajuda o cliente à atingir um objetivo como por exemplo, quando você pede um táxi você para pelo transporte de um ponto à outro não compramos o táxi, ou seja, serviço é um conjunto de facilidades que entrega algum valor ao cliente, porque o ajuda à atingir um objetivo.

Então a **SLA** é um contrato entre o cliente e o fornecedor de serviços, onde têm tudo sobre o serviço que será prestado desde como é avaliada a qualidade até os valores envolvidos, no negócio.

**Conceitos básicos**

**1.1 Conceitos básicos em Web**

Não vou me aprofundar muito neste assunto e sim dizer mais superficial sobre esses conceitos relacionados à web, mas eles também são importantes para efetuar os testes de desempenho relacionados à sites ou algum software feito na Web.

* **Internet:** Internet é o nome dado à conexão de computadores, onde partindo desta conexão todos possam se comunicar. Esta conexão pode ser através de um ISP(*Internet Service Provider*) que é o provedor de internet que vem via linha telefônica, ou por sua vez também pode ocorrer uma conexão LAN(*Local Area Network*), onde que não precisa necessariamente estar conectado à um ISP. Temos também a Intranet que é uma rede interna privada de computadores interligadas através de um suíte que assenta aos protocolos da Internet, só que é um uso exclusivo daquele local.
* **Browser:** É um programa que requisita documentos virtuais, sendo estes as páginas HTML, após a sua conexão com a Internet o mesmo requisita uma página do Servidor Web. O navegador tem uma função básica de requisitar e mostrar o conteúdo para algum usuário.
* **Application Server:** É um software que disponibiliza um ambiente para a instalação e execução de certas aplicações, e tem o objetivo principal de deixar o desenvolvedor com menos trabalho, onde abstraia o mesmo de algumas complexidades de um sistema, o dando mais tempo para a resolução de problemas.
* **Web Server:**  O Servidor Web é onde fica disponível todo o conteúdo que o navegador requisita para mostrar ao usuário. Para ter acesso a ele é necessário uma conexão com a internet. Ele trabalha aceitando os pedidos HTTP dos clientes feitos através dos Browsers, e normalmente responde em páginas HTML. Estas páginas podem ser estáticas onde recebem o diretamente o ficheiro que está no servidor. Ou a mesma pode ser dinâmica onde ser criada dinamicamente por outro programa, script.
* **TCP/IP:** Representa um conjunto de comunicação entre computadores em rede.
* **Endereço IP:**  Para que na internet tenha uma certa ordem se desenvolveu o Endereço de IP(*Internet Protocol*), onde que para cada máquina tenha o seu próprio. Estes endereços são constituídos por números. Caso este endereço de IP seja estático pode ser utilizado até para rastrear o usuário entretanto se for uma conexão dial-up o IP que será designado pelo ISP é respectivo daquela sessão, na próxima vez que o utilizador for se conectar com a internet poderá ser outro endereço. Fazendo assim diminuir para o ISP que ao invês de ter um IP para cada cliente terá um apenas para o modem que poderá ser trocado.
* **HTTP:**  É uma sigla do Protocolo de Transferência de Hipertexto é um protocolo de comunicação utilizado para transferir os dados pela internet e intranet. Funciona através de requisição e resposta. Fica na camada de aplicação de acordo com o modelo OSI, onde é a mais próxima ao usuário e o permite através do Browser por exemplo requisitar páginas Web dos Web Server. Sendo uma forma de padronizar a forma de como se consegue acesso à arquivos na internet.
* **XML:** Para superar as limitações do HTML que é a linguagem padrão das páginas web, foi desenvolvido o XML (*EXtensible Markup Language*). A W3C sua criadora trabalhou para que o XML combinasse com a simplicidade e flexibilidade o HTML e acabou criando uma linguagem com um bom formato para a organização dos dados de forma hierarquica e com esses dados organizados. O HTML e o XML possuem semelhanças como a utilização de tag’s, onde cada tag nessas linguagem consiste em uma que se inicia e outra que se fecha o comando, entretanto por sua maior extenbilidade o XML está se tornando cada vez mais utilizado.
* **Cookie:** O cookie tem a função de guardar dados do usuário, onde muitas vezes é guardado, a data que o mesmo entrou, horário. E tem como objetivo manter a persistência dos dados de uma pessoa na internet como por exemplo um preenchimento automático de algum campo.
* **Servidor de Banco de Dados:** É onde fica armazenado todos os dados de um determinado sistema. O Banco de dados comumente armazena os dados para que serem utilizados com o mesmo fim. O modelo de dados mais adotado é um relacional onde possui estruturas com o formato de tabelas e é composto por linhas e colunas. O bando de dados é mantido e acessado pelo SGBD(Sistema Gerenciador de Banco de Dados).
* **Firewall:** É como um filtro com a função de verificar a comunicação que se passam em uma rede podendo a permitir ou bloquear, e faz isso com base em uma examinação no tipo de serviço desejado como por exemplo a entrada em algum site específico na web.
* **Router:** ou Gateway é um equipamento utilizado para que partilhar a sua ligação com a internet por vários PC’s.
* **Proxy:** Serve como intermediário entre o computador(Usuário) e a internet, onde originalmente surgiu com o objetivo de levar internet para uma LAN sendo que para chegar na mesma toda permitindo todos os computadores chegarem ao conteúdo externo considerando à LAN consegue conteúdos internos. Sendo que uma LAN não tem endereço válido na Internet. Sendo assim ela não tem uma conexão direta com a mesma e todas requisições são enviadas para o proxy.
* **Cache:**  A cache é “ao contrário” do cookie onde a mesma armazena na memória dados sobre a página com a finalidade de uma abertura mais veloz da página requisitada.
* **Cluster:** É o nome dado para que quando se tem muitos computadores interligados e fazendo um processamento de forma que se pareça apenas um único computador.

**SEGUNDA APRESENTAÇÃO**

**1.2 Conceitos básicos de Testes de Software**

Os testes de software tem como objetivo a encontrada de *bug’s* no sistema. Com isso ajudando na qualidade do software. Uma falha ou um *bug* que não possui uma evidência de o que está causando é equivalente à um criminoso que não se tem provas contra o mesmo, ou seja, não tem como efetuar a correção.

**1.2.1 Conceitos Gerais em Testes de Software**

* **Planejamento de testes:** Em um Planejamento de testes pode se gerar um ou mais planos de testes.
* **Plano de Testes:** Já os Planos de Testes é constituído por, Requisitos de testes, Casos de testes e Cenário de testes que vão ser utilizado nos decorrer dos testes e também contem toda documentação envolvida no planejamento de testes. Os planos de testes devem estar sempre atualizados, devido à integridade dos dados de Teste.
  + **Requisito de Testes:** Estes requisitos representam o que temos que testar em um software, onde um requisito de teste deve gerar um ou mais Casos de Testes, são geralmente criados na fase de Planejamento de testes, como exemplo podemos usar um simples CRUD, o cadastrar seria um dos requisitos de teste e no mesmo teria apenas um caso de teste por ser algo simples. Entretanto um requisito mais complexo poderá exigir mais de um caso de teste.
  + **Caso de Teste ou Teste:** Representa o que de fato devemos testar no sistema. É um desdobramento dos Requisitos de Testes, Entretanto um caso de testes pode satisfazer mais de um Requisitos de Testes. Mas para que o mesmo exista é necessário no mínimo um Requisito de Testes relacionado ao caso de uso específico. Cada caso de teste só pode conter um conjunto de Passos de Teste.
    - **Passo de Teste ou Ação:** É uma ação que se tem que fazer para que se consiga realizar os testes, um Passo de Teste ou Ação pode ser por exemplo: Clicar no botão, Abrir a aplicação… .E cada passo de testes pertence à apenas 1 conjunto de passos de testes.
    - **Tipo de Teste:** Exemplos: Testes Funcionais, de Segurança, de Performance.
  + **Cenário de Teste:** É um conjunto de casos de testes ou de Scripts de Testes que tem como objetivo com o intuito de caracterizar uma grande situação de testes. Aplicado principalmente para Testes de performance. Exemplo:
  + Rode o Script 1 , depois o 2 se der algum erro rode o 4..
    - **Script ou Script de Teste:** Um Script de testes, também é conhecido como Script de automação tem como o objetivo a execução de um conjunto de ações através de uma ferramenta de automação. Normalmente representa uma automatização. E se faz a automação de um ou mais casos de testes através de qualquer ferramenta de Teste.
* **Ponto de Verificação:** É uma ação ou passo específico que tem como o objetivo verificar a validade de um Caso de teste, ou uma regra apenas dentro de Script de testes
* **Ambiente de Teste:** É toda infra-estrutura que onde o software será testado. Onde leva em conta a equipe, a rede , hardware, software, ambiente fisico, documentação. Gerando assim a realização dos testes em condições conhecidas e controladas.
* **Relatório de Teste:** É onde contém todos os dados relacionados aos testes executados, e dependendo do enfoque dado até mesmo os dados de planejamento podem estar inclusos. Existem alguns padrões de relatórios.
* **Modelo de Teste:** Conjunto de técnicas , que tem como o objetivo representar ou modelar algo real, podendo ser algum sistema.
* **Metodologia de Teste:** É um conjunto de técnicas voltadas para a criação e/ou execução de um sistema com o objetivo de aumentar a qualidade de um software.
* **Processo de Testes:** É qualquer processo que represente como acontece o gerenciamento dos testes e realização
* **Teste de Software:** Qualquer teste aplicado em um software.

**1.2.2 Principais tipos de Teste de Software**

Estão como “tipos” de Teste, porque um Teste pode ter mais de um tipo, ou seja, quando está efetuando um teste pode ocorrer ms de um tipo ao mesmo tempo.

* **Testes Funcionais:** Tem como objetivo verificar se o sistema realmente funciona como o proprosto nos requisitos e documentação, considerado o teste mais importante.
* **Testes de Performance:** Tem como o principal objetivo verificar se o sistema funciona na velocidade que precisa funcionar. Demonstra as informações derivadas de carga e tempo.
* **Testes de Usabilidade:** Tem como objetivo principal verificar a facilidade de uso do software para a realização de uma tarefa específica, sabendo do grupo que utilizará à ferramenta.
* **Testes de Cobertura de Código:** É uma métrica onde mostra a efetividade dos testes realizados em uma aplicação. Mostrando o quanto da aplicação foi testada durante uma dada bateria de testes.
* **Testes de Unidade:** Cujo o objetivo é testar uma parte do software somente. E verificar se as mesmas funcionam corretamente.

1. **Testes de Integração:** Visa a garantia de funcionamento de unidades em conjunto.
2. **Testes de UAT:** É um teste onde acontece a validação se o software está como o usuário deseja. Tendo como objetivo dar um aceite ou não para o sistema.
3. **Testes de Segurança:** Tem como objetivo verificar a segurança do software nas mais diversas formas.
4. **Testes de Regressão:** Quando acontece uma atualização o software todo ou uma boa parte do mesmo tem que ser testada toda de novo com o objetivo de verificar se a nova atualização não danificou a versão completa anterior.
5. **Testes de Aplicação Web:** São testes realizado em ambientes Web e em aplicações Web.
6. **Testes Dirigidos a Dados(Data-Driven Test):** É um teste automatizado composto por um ou mais casos de teste (preferencialmente casos simples), podendo ser casos de teste funcional ou de performance. Que utilizam uma quantidade de dados. Como exemplo temos o teste de um login por exemplo onde a partir de dados válidos e inválidos seria permitido a entrada em um suposto sistema, e estes dados seriam utilizados pelo Script de testes automatizados.

**TECNICAS PARA TESTE DE SOFTWARE**

* **Teste Caixa-Branca:**  São testes baseados no código fonte do sistema.
* **Teste Caixa-Preta:** São da aplicação onde não se contém um conhecimento sobre o código fonte do sistema.
* **Teste Caixa-Cinza:** São testes onde se tem conhecimento do código fonte, mas é feito pela parte “Externa”. Este tipo de teste é uma junção dos dois anteriores.

**1.3 Conceitos básicos de Testes de Performance**

Quando se fala em testes de performance, é destacado 4 primordiais conceitos, sendo estes: Vazão(*throughput*), Tempo de Resposta, Carga(comportamento) devido à usuários(simultâneos no sistema) e Gargalo. Onde todos os gráficos e dados relacionados aos testes de performance diz respeito à estes conceitos. Dados como utilização da memória também são pegados durante a bateria de testes. Mas de nada não válidos caso não relacionado aos anteriores.

**Exemplo - Analogia**

Para melhor entender esses 4 conceitos fundamentais, pode se utilizar uma analogia relacionada ao uso de canos, onde a Carga de trabalho que o software será submetido está relacionada à quantidade de água que o cano leva. Tempo de resposta seria o tempo que a água passa por todos os canos. E a vazão por consequência a quantidade de água que consegue passar. Já o gargalo representa os locais onde o cano está com baixo desempenho ou “trava”.

E o maior objetivo dos testes de performance é encontrar o gargalos existentes no sistema.

**1.4 XML(EXtensible Markup Language)**

É uma linguagem baseada em texto simples, projetada para o armazenamento e transporte de dados em formato de texto mais simplificado. E tem como principais características:

* Linguagem de marcação
* Linguagem baseada em tag’s como o HTML
* As tag’s não são pré definidas como no HTML
* Você pode definir suas próprias marcas por isso é conhecida como uma linguagem extensível
* As tag’s são projetadas pra serem auto descritivas
* É recomendação da W3C para o armazenamento e transporte de dados.

**Vantagens:**

* **Tecnologia Agnóstico:** Por ser texto simples XML é uma linguagem independente. Podendo ser utilizada por qualquer tecnologia para o armazenamento de dados e transmissão.
* **Legível:** É de fácil compreensão.
* **Extensible:** em XML, temos as tag’s personalizadas que facilita o uso da linguagem.
* **Fácil validação:** Usando o XSD(descreve a estrutura do XML), DTD(Conjunto de declarações que permitem dizer se o documento é da família SGML) a estrutura XML pode ser validada facilmente.

**Desvantagens:**

* **Sintaxe redundante:** normalmente contém muitos termos repetidos.
* **Detalhado:** Devido aos detalhes o XML fica sendo um arquivo grande o que dificulta, devido aos custos envolvidos tanto no transporte como no armazenamento.

**XML PARSING**

É o ato de se passar pelo arquivo XML e ver seus dados e também poder os modificar.

**XML PARSER**

O XML parser é uma forma onde se pode acessar ou modificar dados de um documento XML, onde o java dá suporte com uma série de analisadores.

**ANALISADORES**

Os seguintes são os analisadores em java:

* **DOM Parser:** Ele faz a leitura do documento a partir do carregamento completo do mesmo para a memória e cria uma árvore hierarquica completa do mesmo.
* **SAX Parser:** Analisa o documento baseado em gatilhos e não o carrega completamente na memória.
* **JDOM Parser:** Analisa o documento de maneira semelhante ao DOM, mas de uma maneira mais simplificada.
* **StAX Parser:** Analisa o documento semelhante ao SAX entretanto de maneira mais eficiente.
* **XPath Parser:** Analisa com base na expressão e é utilizado com o XSLT(Linguagem de marcação do XML).
* **DOM4J Parser:**  É uma biblioteca em java para analisar XML, XPath, XLST e fornece suporte para o DOM, SAX e JAXP

**QUANDO UTILIZAR DOM Parser**

O DOM Parser é recomendado quando é necessário acessar os dados de um documento XML mais de uma vez, quando precisa saber muito sobre o documento e fazer algumas alterações no mesmo como classificar certos elementos.

**INTERFACES**

* **Node:** É o tipo de dados base do DOM;
* **Element:** A maioria dos objetos são elementos quando acontece a abertura e fechamento de uma tag é considerado elemento.
* **Atrr:** Representa um atributo de um elemento.
* **Text:** É um texto que pode ser encontrado em um elemento ou em um Atributo.
* **Document:** É todo o documento XML.

**TERCEIRA APRESENTAÇÃO**

**TESTE BASEADO EM MODELOS E XMI**

**Teste baseado em modelos:**

* **Porque utilizar modelos ?:** A utilização dos modelos se deve ao aumento da complexidade dos sistemas, com isso acontece da equipe sentir dificuldades de entender como o mesmo funciona. Além da econômia (Modelos são econômicos) os mesmos ajudam a específicar e entender melhor as funcionalidades do sistema.
* **Definição e características:** Os testes baseados em modelos como o próprio nome descreve são casos de testes derivados de modelos, ou de parte deles, sendo estes relacionados ao sistema que será testado. São considerados testes caixa-preta, devido ao desconhecimento do código que estará por trás so software. Geralmente relacionado aos requisitos funcionais. E tem como objetivo a automação dos testes. Onde em vez de se criar casos de testes derivados do documento de requisitos de cria a partir de um modelo de como se espera que o sistema funcione. Partindo deste conceito ferramentas são utilizadas para que assim possa fazer estes testes. Então sua eficácia estará relacionada ao nível dos modelos.
* **Vantagens:** 
  + A geração de testes começa mais cedo no ciclo de desenvolvimento devido aos modelos serem criados antes da implementação.
  + Evita a ambiguidade que pode estar presente dentro do documento de requisitos.
  + Reduz o tempo dos testes, dando um maior tempo para implementação e outros passos importantes do ciclo de desenvolvimento.
  + Reduz os custos relacionados à parte de teste.
  + Não é necessário um conhecimento sobre programação, fazendo assim mais fácil para encontrar profissionais para fazerem estes serviços.
  + Mudando o critério dos testes é possível a criação de vários suites de testes.
    - **Suites de testes:** É um conjunto de casos de teste onde cada caso de teste pode conter um script de testes diferente. Onde podem ser executados sequencialmente e podendo parar o mesmo caso encontre algum erro. E com sua utilização os testadores não ficam perdidos caso aconteça algum erro em um teste específico. Com sua utilização se dá uma certa organização do trabalho e melhor visualização dos resultados em questão de detalhes.

**Processo:**

O processo é composto por 5 etapas:

* **Modelagem do sistema:** Nesta parte é criado modelos que são chamados de modelos abstratos por serem menores e mais simples do que o sistema em um todo. E acontece a omição dos detalhes do sistema, ou seja, é específicado somente o necessário. Contém as saídas esperadas para que assim seja possível uma análise depois. E acontece a utilização de ferramentas adequadas como ferramentas CASE por exemplo, para verificar sua consistência, e esta verificação acontece para que assim não se tenha conflitos entre os requisitos.
* **Geração de testes abstratos:** Nesta parte é derivados casos de testes ou suites de testes dos modelos criados anteriormente, respeitando regras sendo estas responsáveis pelos testes que vão ser extraídos do modelo. Devido à simplicidade dos modelos os testes criados são abstratos, ou seja, esse conjunto de testes gerados não são diretamente executáveis, decorrente da simplicidade dos modelos. É feito através de uma ferramenta, e a maioria destas também gera uma matriz de rastreamento de requisitose outros relatórios de cobertura. Nesta matriz este rastreamento de requisitos funciona da seguinte forma: Ele liga os requisitos funcionais com os testes gerados, para que assim fique melhor explicado o objetivo de cada teste. Já os relatórios de cobertura dizem quais dos testes gerados realmente estão funcionando como o combinado, ou seja, quais deles realmente serão utilizados com eficácia.
* **Concretização dos testes:** Nesta parte é elaborado a partir dos testes abstratos, testes executáveis. Realizado através de uma ferramenta de transformação, que para isso utiliza vários templates e mapeamentos que traduzem cada caso de teste abstrato em um script de testes executável ou escrevendo em um código adaptador que realiza esta função. E esta etapa tem como o objetivo ser a ponte entre a geração dos testes e a realização dos mesmos.
* **Execução dos Testes:**  Esta etapa como o nome mesmo diz, está relacionada à execução dos testes que foram extraídos dos modelos. Entretanto estes podem ser realizados basicamente de duas maneiras sendo elas :
  + Online:
    - Onde as ferramentas de testes se conectam diretamente com o sistema em casos de aplicações web;
    - Testes realizados dinâmicamente, ou seja, realizada após serem produzidos;
    - As próprias ferramentas de testes gerenciam os testes e a gravação dos resultados.
  + Offline:
    - Nas offline acontece semelhante a diferença é que os testes não são realizados dinâmicamente, ou seja, todos os testes são gerados e podem ser visualizados pelo computador antes da execução dos mesmos. Esta execução pode ser automaticamente após a elaboração, entretanto pode-se também fazer os testes em outra plataforma devido à elaboração ter sido feita toda antes da execução.
* **Análise de resultados:** Nessa ultima etapa eu verifico os resultados dos testes e também é realizado todas as medidas necessárias para efetuar as correções. E para cada falha que for relatada é necessário informar onde se está o erro que determinou a falha. Os erros podem estar nos casos de testes aplicados, ou nos modelos, ou no código adaptado.

**Exemplo de ferramenta - MaTeLo:** É uma ferramenta offline que só funciona em windows, que o usuário pode adicionar bibliotecas de entradas que serão utilizadas para definir as atribuições das entradas, adicionar bibliotecas de saídas, que serão utilizadas para fazer a verificação dos resultados que a própria ferramenta faz. A mesma simula ações do usuário, gera casos de testes dentre outras funcionalidades.

***Fonte :*** *http://pt.slideshare.net/andrezapbl/teste-baseado-em-modelo*.

**XMI**

Durante o desenvolver do Software pode ser necessário fazer a troca de ferramentas, e o que fazer com os dados que já estão na outra?, na Engenharia de software gasta muitos esforços para a contrução de ferramentas de ajudem no desenvolvimento, ou seja, etapas que fazem parte do ciclo de vida do software como: Documentação, Modelagem, Desenvolvimento e Manutenção. E para melhor usufruir de suas funcionalidades, seria ótimo que elas pudessem se comunicar entre si, e mudar de ferramentas e manter o mínimo de impacto possível. Com essas possibilidades veio a criação do XMI.

O XMI( XML Metadata Interchange) é uma especificação XML que permite representar modelos com o objetivo de se manter uma comunicação clara entre ferramentas de modelagem(Interoperabilidade), repositórios de *metadados\**, e outras ferramentas do desenvolvimento. E permite trabalhar com objetos, ou seja, com troca de dados padronizados. E o XMI foi criado baseado no UML(Unified Modeling Language - Utilizado para descrever metamodelos, neste caso utilizada para análise e design), XML (eXtensible Markup Language - Utilizado para melhor hierarquização dos dados e fácil leitura por ser um documento simples , neste caso usado para a sintaxe) e o MOF(Meta Object Facility - Utilizado para definições de metadados e gerenciamento)

*Metadados\* : Do grego “Meta” quer dizer “alem de”, com isso quer dizer que tem algo além dos simples dados, sendo estes com o objetivo de manter certa ordem para a fácil manipulação e hierarquização dos dados dentro de um arquivo, exemplo de arquivos de metadados sera o XML e o HTML.*

**Model Based Testing (MBT) - XMI**

Os testes baseados em modelos, como ja explicado anteriormente utiliza de uma ferramenta para que a geração dos casos de testes sejam possíveis. Pode ser utilizado o documento XMI para isso, como dito anteriormente este documento guarda particularidades relacionadas aos modelos, particularidades estas que são utilizadas para a comunição entre ferramentas. Então para a utilização deste tipo de documento para a criação de testes pode acontecer da seguinte maneira:

* Após a conclusão da modelagem se exporta o modelo como arquivo XMI, que nada mais é do que uma especificação de um arquivo XML.
* Com a exportação já feita, o que acontece? O que a ferramenta faz? As respostas são simples. Acontece que a ferramenta pega o arquivo faz um parser no mesmo, ou seja, o lê e pega o que lhe é conveniente, e cria casos de testes possíveis partindo daquele modelo. E os manipula para que aconteça a criação de vários suites de testes de acordo com os critérios impostos.

**Vantagens - XMI no MBT**

* **Compatibilidade:** Mesmo cada ferramenta possuindo seu formato muitas trabalham com a extensão XMI. E há possibilidade do mesmo arquivo para a utilização em mais partes do projeto.
* **Facilidade de leitura:** Como o XMI é uma especificação XML, ele possui várias bibliotecas com o objetivo de fazer leitura e escrita dentro do arquivo, tornando assim mais fácil sua leitura e manipulação.

**Desvantagens - XMI no MBT**

* **Tamanho:** Por ser uma extensão do XML , o arquivo XMI herda um dos seus defeitos que é o grande detalhamento de seus dados. Fazendo assim ser um arquivo grande necessitando de um maior espaço na memória para manipulá-lo, e um maior espaço em disco para armazenagem. Este tamanho demasiado pode gerar gargalos no software de conversão, ferramenta está responsável por criar os artefatos de testes.

*Obs.: O XMI é um conjunto tecnologias, como UML, XML e MOF. Onde com isso herda também outros desvantagens quanto vantagens relacionadas às outras tecnologias.*

**QUARTA APRESENTAÇÃO**

**UML( UNIFIED MODELING LANGUAGE)**

**UML(UNIFIED MODELING LANGUAGE)**

O UML é uma linguagem de modelagem unificada que tem como objetivo sistematizar a área de modelagem. E a mesma tem como função representar um software de modo padronizado através de modelos, permitindo que assim seus desenvolvedores possam visualizar seus trabalhos mais simples e de forma padronizada. Diminuindo assim a complexidade que pode ser gerado através de um documento de requisitos extenso. Além da parte gráfica se é possível perceber uma semântica envolvida para que fique ainda melhor o entendimento, independentemente do processo. A descrição é realizada tendo em vista a orientação a objetos.

O desenvolvimento de um sistema utilizando o UML é feito em 5 etapas onde estas são:

· **Análise de requisitos:** Nesta parte é onde captura o que o cliente realmente necessita. E esta captura é feita através dos casos de uso, onde estes são compostos por “Atores externos”, ou seja, utilizadores do sistema são interligados com as suas respectivas funcionalidades, mostrando assim interação entre o ator e o caso de uso. Estes atores do diagrama de casos de uso são modelados com relacionamentos que possuem uma comunicação associativa entre eles ou são desmembrados em hierarquia. Neste diagrama também é especificado o que o ator necessita para a utilização desta funcionalidade. E com isto facilita a visão em relação para o cliente saber o que se pode esperar do sistema, sem saber como realmente será implementado. A fase de análise de requisitos também está direcionada com a parte de negociação do projeto, não só com a parte de desenvolvimento em si.

· **Análise:** Esta parte tem o objetivo de se preocupar com as primeiras abstrações, ou seja, as primeiras classes e objetos que serão utilizados para a solução do domínio do problema. Para esta parte é utilizado os diagramas de classe, onde estes fazem uma ligação entre as classes e também como uma se relaciona com a outra, dizendo também os objetos presentes nas mesmas. Para desenvolver os casos de uso também são observadas as relações entre as classes, onde podem definir se as ligações serão: include, extend, generalização , está parte será melhor explicado quando adentramos mais a fundo no diagrama de casos de uso. Este diagrama conterá somente classes que estarão destinadas a solucionar somente os problemas relacionados aos domínios do sistema, ou seja, técnicas que gerenciem o banco de dados, comunicação, interface, concorrência, outras classes não estarão presentes neste diagrama.

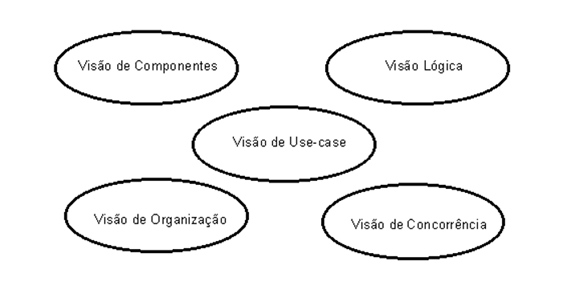
· **Design (Projeto):** Nesta parte é onde ocorre a expansão dos resultados obtidos na etapa de análise, ou seja, novas classes serão adicionadas, acontecerá também a mesclagem da parte relacionada aos domínios e da infraestrutura, permitindo modificações em ambas partes. Com isso ocorre um melhor detalhamento das especificações para a parte de programação em si.

· **Programação:** Nesta fase é onde ocorre a conversão para código as classes especificadas anteriormente podendo ser em qualquer linguagem orientada a objetos. O que fará a conversão ser fácil ou complicada será em função do tipo da linguagem escolhida. No momento da criação dos modelos é recomendado não fazer o mesmo traduzindo-o para código. As fases anteriores têm como o objetivo um melhor entendimento de como o sistema deve funcionar, dependerá do analista verificar e fazer modificações no seu conteúdo, e após isto os modelos não estarão mais exercendo a função de demonstração real do sistema e sim de como o mesmo deve funcionar. Fazendo assim da programação uma fase distinta onde acontece somente a conversão dos modelos para a linha de código.

· **Testes:** Geralmente acontece nesta parte os testes de unidade, integração e aceite. Onde todos já foram mencionados. O interessante desta parte relacionada ao desenvolvimento UML é a parte de testes de aceite (aceitação), onde o mesmo é executado de forma “caixa-preta” (com desconhecimento do código por trás do software) para verificar se o software está funcionando como o Diagrama de casos de uso feitos no início do projeto.

**Visões**

O ideal seria que fosse possível a descrição de um sistema complexo em apenas uma página sem contradições nem ambiguidades entre os requisitos, onde possa ser facilmente interpretado e isto é impossível onde um único gráfico não é possível descrever todo o sistema com as informações necessárias.

Como o sistema é composto por diversos aspectos é possível ver o mesmo de vários pontos de vista diferentes.

· **Visão “use-case”:** Descreve as funcionalidades do sistema do ponto de vista dos atores externos (usuários), o que fazer para conseguir acessar tal funcionalidade, esta parte mostra como o sistema de fato irá se comportar e pode ser ilustrada com o Diagrama de casos de uso e com o Diagrama de atividades.

· **Visão lógica:** É uma parte feita pelos desenvolvedores e analistas que tem como o objetivo verificar como as funcionalidades serão implementadas baseadas nos casos de uso feitos anteriormente. Esta parte observa o sistema e elabora como ele vai ser no seu interior como as classes, objetos e relacionamentos que estarão presentes. E também as colaborações dinâmicas também são estudadas, ou seja, as informações que passarão de uma classe para outra e assim por diante. Interfaces e estruturas de classe estarão presentes nesta parte do sistema juntamente com a persistência de dados e concorrência. A estrutura estática a que se diz respeito à estrutura das classes e objetos são definidas pelos Diagramas de Classe e Objeto. Já a parte relacionada com o modelamento dinâmico e como o sistema irá se comportar é descrita pelos diagramas de sequência, estado, colaboração e atividade.

· **Visão de componentes:** Sendo uma descrição da implementação dos módulos e suas dependências são feitas também por desenvolvedores e consiste de uma visão relacionada aos componentes presentes nos diagramas.

· **Visão de concorrência:** Trata-se de uma visão relacionada aos processos e processadores, ou seja, se há um sistema de gerenciamento, verificação de processos que estejam sendo executados em segundo plano. Este tipo de visão é exemplificada nos diagramas dinâmicos como o de sequência, colaboração, estado, atividades. E também por diagramas de implementação como o de componentes e execução.

· **Visão de organização:** Esta parte é representada pelo diagrama de execução. E esta visão é relacionada à organização física dos sistemas, os computadores, periféricos e como eles se conectam entre si. Sendo uma etapa realizada pelos desenvolvedores, integradores e testadores.

[**http://www.etelg.com.br/paginaete/downloads/informatica/apostila\_uml.pdf**](http://www.etelg.com.br/paginaete/downloads/informatica/apostila_uml.pdf)

[**https://pt.wikipedia.org/wiki/UML**](https://pt.wikipedia.org/wiki/UML)

**UML - CASOS DE USO**

O diagrama de caso tem como finalidade mostrar(documentar) ao usuário o que “o sistema faz” do ponto de vista do usuário. Descrevendo as principais funções do sistema e a relação destas funções com os usuários do sistema. Este diagrama apresenta apenas as funções e relação com os usuários e não entra em aspéctos mais detalhados “como o sistema faz”. Ele tem como objetivo auxiliar a comunicação entre os analistas e o cliente, descrevendo e mostrando as principais funcionalidade do seu sistema.

**Elementos:**

O Diagrama de Caso de Uso é representado da seguinte forma:

* atores;
* casos de uso;
* relacionamentos entre estes elementos.

Estes relacionamentos podem ser:

* associações entre atores e casos de uso;
* generalizações entre os atores;
* generalizações, *extends* e *includes* entre os casos de uso.

*casos de uso* podem opcionalmente estar envolvidos por um retângulo que representa os limites do sistema.

## Relação entre Casos de Uso

Quando estamos construindo o diagrama de Casos de Uso, sempre nos deparamos com relações que irão ocorrer entre as funções. Neste diagrama temos 3 tipos de relações:

**Include**(inclusão) , **Extend**(Extensão) e **Generalization**(Generalização [ou Herança]).

## **Include:**

Quando o caso de uso A “inclui” o caso de uso B, significa que sempre que o caso de uso A for executado o caso de uso B também será executado. A direção do relacionamento é do caso de uso que está incluindo para o caso de uso incluído.

## **Extend:**

Quando o caso de uso B estende o caso de uso A, significa que quando o caso de uso A for executado o caso de uso B poderá (poderá – talvez não seja) ser executado também. A direção do relacionamento é do caso de uso extensor (aqui o caso de uso B) para o caso de uso estendido (aqui o caso de uso A).

## **Generalization:**

Quando o caso de uso B generaliza o caso de uso C isso significa que, além de fazer tudo que nele está especificado (ele = B), ele também executará tudo que está especificado no caso de uso C.

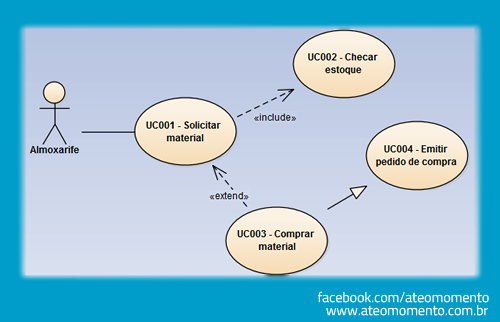
Muitos profissionais falam que isso não deve ser compreendido como a herança da orientação a objetos, mas na minha opinião deve ser sim, apenas (em tempo de modelagem de caso de uso) estamos num nível de abstração diferente, mas o produto final desta modelagem será software codificado.

A direção do relacionamento é sempre do generalizador (aqui o caso de uso B) para o generalizado (caso de uso C).

### 

### Explicando…

No diagrama temos quatro Casos de Uso, e três relacionamentos diferentes: Include, Extend e Generalization.



### Explicando o Include

O caso de uso “Solicitar Material” faz include no caso de uso “Checar Estoque”. Isso se dá porque sempre que houver a solicitação de material sempre haverá a consulta ao estoque para saber se o material está disponível.

Se sempre haverá, o relacionamento correto é o include.

### Explicando o Extend

O caso de uso “Comprar Material” estende o caso de uso “Solicitar Material”. Isso se dá porque quando houver a solicitação de material, caso o material não exista em estoque (após consulta via o caso de uso “Checar estoque”) poderá ser solicitado a compra do item.

Mas também poderá não ser solicitada a compra, pois o item pode existir em estoque. Se poderá ser solicitada a compra (e não sempre será solicitada a compra) o relacionamento correto é o extend.

### Explicando o Generalization

O caso de uso “Comprar Material” generaliza o caso de uso “Emitir pedido de compra”. Isso se dá porque no caso de uso “Emitir pedido de compra” existe especificação de como se realiza o pedido de compra, processo que não se dá somente no contexto do almoxarifado, mas é o mesmo em qualquer área do negócio.

Dessa forma, não justifica-se duplicar a especificação pertinente em outro caso de uso, basta reaproveitar o que já está pronto mas generalizado a ponto de poder ser aproveitado por alguém que o especialize.

[*http://www.ateomomento.com.br/caso-de-uso-include-extend-e-generalizacao/*](http://www.ateomomento.com.br/caso-de-uso-include-extend-e-generalizacao/)

[**http://www.devmedia.com.br/o-que-e-uml-e-diagramas-de-caso-de-uso-introducao-pratica-a-uml/23408**](http://www.devmedia.com.br/o-que-e-uml-e-diagramas-de-caso-de-uso-introducao-pratica-a-uml/23408)

[*http://homepages.dcc.ufmg.br/~figueiredo/disciplinas/aulas/uml-casos-de-uso\_v01.pdf*](http://homepages.dcc.ufmg.br/~figueiredo/disciplinas/aulas/uml-casos-de-uso_v01.pdf)

[*http://www.dsc.ufcg.edu.br/~sampaio/cursos/2007.1/Graduacao/SI-II/Uml/diagramas/usecases/usecases.htm*](http://www.dsc.ufcg.edu.br/~sampaio/cursos/2007.1/Graduacao/SI-II/Uml/diagramas/usecases/usecases.htm)

**Parser XMI - Tudo até agora.**

O primeiro passo para se fazer o *parser* do XMI, como em comum no XML é fazer a importação do mesmo para o programa e após para sua memória. Chamando uma classe com a função de pegar todos os dados necessários para extrair sua parte relacionada à UML. Para a construção desta classe é necessário que informe o nome do documento a ser analisado (XMI). Nesta classe acontece a inicialização das listas internas que tem o objetivo de ser preenchidas com dados dos diagramas presentes em uma classe específica que citarei posteriormente. Na medida que os diagramas são encontrados com base nos dados padrões encontrados no documento e os procurados pelo *parser*, eles são adicionados à lista de diagramas UML. Na mesma classe é possível fazer o *get* dos Diagramas fazendo assim poder ver os dados encontrados, e assim verificar a qualidade do *parser* (Se o mesmo conseguiu absorver todos os diagramas internos).

A lista de diagramas UML, que foram citadas no parágrafo anterior, é composta por uma classe dentro do *parser*. Onde na mesma contém atributos como nome, e id onde todo diagrama é composto de um numero identificador (id) e um nome (que pode ser um campo em branco). Como os Diagramas são basicamente compostos de elementos e associações a classe de diagramas também possui uma lista de elementos e associações que são parte dos diagramas de casos de uso da UML. Conforme o *parser* é necessário adicionar elementos, e associações na lista, onde cada um dos dois não podem estar duplicados. Isto define que há contradições dentro do mesmo modelo. As associações também podem dar um erro relacionado as mesmas estarem envolvidas com outros elementos que não estão no diagrama isso causa uma Inconsistência nos dados. E o *parser* irá verificar e mostrar tal erro caso ocorra. A remoção de elementos também pode ser feita dentro desta classe UML.

Já a classe de elemento UML, é uma classe que gera outras. O que faz dela ser uma classe genérica que contém algo que é em comum para cada elemento da UML. E expandindo para as classes (filhas) para algo que é mais específico, como um elemento do diagrama de casos de uso por exemplo. Cada elemento possui um identificador único, e um *label* que é o nome do ator por exemplo. Onde o ID é utilizado para diferenciação de dois elementos em um diagrama, já as *labels* podem ser repetir dentro do mesmo diagrama se necessário, onde nesta classe é feita a inserção dos mesmos através do construtor e também os *getter* para ambos os dados.

Como os Elementos as associações também possuem uma classe genérica, que representa a relação de dois elementos UML que fazem algum tipo de interação. Onde também possuem um ID e pode (não necessariamente) ter um rótulo que é armazenado como uma *label*. Como a associação é feita através de dois elementos os mesmos são atributos da classe onde se pode pegar os mesmos através de simples métodos *get.* Importante ressaltar que a classe não trata qual elemento vem primeiro, somente que os mesmos possuem uma associação dentro do diagrama.

**Parser XMI - Tudo até agora.**

O primeiro passo para se fazer o *parser* do XMI, como em comum no XML é fazer a importação do mesmo para o programa e após para sua memória. Chamando uma classe com a função de pegar todos os dados necessários para extrair sua parte relacionada à UML. Para a construção desta classe é necessário que informe o nome do documento a ser analisado (XMI). Nesta classe acontece a inicialização das listas internas que tem o objetivo de ser preenchidas com dados dos diagramas presentes em uma classe específica que citarei posteriormente. Na medida que os diagramas são encontrados com base nos dados padrões encontrados no documento e os procurados pelo *parser*, eles são adicionados à lista de diagramas UML. Na mesma classe é possível fazer o *get* dos Diagramas fazendo assim poder ver os dados encontrados, e assim verificar a qualidade do *parser* (Se o mesmo conseguiu absorver todos os diagramas internos).

A lista de diagramas UML, que foram citadas no parágrafo anterior, é composta por uma classe dentro do *parser*. Onde na mesma contém atributos como nome, e id onde todo diagrama é composto de um numero identificador (id) e um nome (que pode ser um campo em branco). Como os Diagramas são basicamente compostos de elementos e associações a classe de diagramas também possui uma lista de elementos e associações que são parte dos diagramas de casos de uso da UML. Conforme o *parser* é necessário adicionar elementos, e associações na lista, onde cada um dos dois não podem estar duplicados. Isto define que há cont1mradições dentro do mesmo modelo. As associações também podem dar um erro relacionado as mesmas estarem envolvidas com outros elementos que não estão no diagrama isso causa uma Inconsistência nos dados. E o *parser* irá verificar e mostrar tal erro caso ocorra. A remoção de elementos também pode ser feita dentro desta classe UML.

Já a classe de elemento UML, é uma classe que gera outras. O que faz dela ser uma classe genérica que contém algo que é em comum para cada elemento da UML. E expandindo para as classes (filhas) para algo que é mais específico, como um elemento do diagrama de casos de uso por exemplo. Cada elemento possui um identificador único, e um *label* que é o nome do ator por exemplo. Onde o ID é utilizado para diferenciação de dois elementos em um diagrama, já as *labels* podem ser repetir dentro do mesmo diagrama se necessário, onde nesta classe é feita a inserção dos mesmos através do construtor e também os *getter* para ambos os dados.

Como os Elementos as associações também possuem uma classe genérica, que representa a relação de dois elementos UML que fazem algum tipo de interação. Onde também possuem um ID e pode (não necessariamente) ter um rótulo que é armazenado como uma *label*. Como a associação é feita através de dois elementos os mesmos são atributos da classe onde se pode pegar os mesmos através de simples métodos *get.* Importante ressaltar que a classe não trata qual elemento vem primeiro, somente que os mesmos possuem uma associação dentro do diagrama.

**Parser Atualizado - Padrão (Diagrama de casos de uso).**

Analisando o documento xmi extraído do astah, foi possível verificar alguns padrões. Vale destacar que o documento devido à grandeza de detalhes é muito extenso, e carrega consigo partes inúteis para a confecção do parser, uma análise somente de partes importantes se faz possível com um auxílio de uma ferramenta para verificar modificações como as presentes no GitHub, BitBucket entre outros. A utilizada foi o WinMerge devido a sua simplicidade e pequeno tamanho. Os padrões encontrados relacionados ao diagrama UML foram que os dados eram inseridos no documento de acordo com sua ordem de inserção e sua exclusão só era possível usando o Ctrl+D após de selecionar o que deseja excluir na ferramenta(Astah). Um ator possui alguns dados que serão expostos a seguir:

<UML:Actor xmi.id=”yur -555abc222 ” name = “Nome+do+ator” \*\* isRoot=”false” \*\*\*>

<UML:ModelElement.namespace>

<UML:Namespace xmi.idref=”lc – 555abc222”/>

</UML:ModelElement.namespace>

<UML:ModelElement.visibility xmi.value =”public”/>

<UML:ModelElement.stereotype>

<UML:Stereotype xmi.idref=”2s5 – 555abc222”/>

</UML:ModelElement.stereotype>

</UML:Actor>

Após análise desta parte é possível pegar o número identificador do Ator e o seu nome que está presente no modelo, pode estar em branco ou não, estes dados sao importantes para o parser, onde que com eles ele pode verificar a exitência de outro elemento com o mesmo ID e se houver ocorreu uma incônsistência no modelo. Ele armazena estes dados e o armazena como um elemento do modelo. As tags em azul representam os nodos que devem ser procurados dentro do documento para que assim se encontre todos os dados que necessita(Dados neste caso presente já nos atributos). A parte sublinhada é igual para todo o documento, o que modifica é os dados que estão antes do hífen.

<UML:UseCase xmi.id=”aaa -555abc222 ” name = “Nome+do+Caso” \*\* isRoot=”false” \*\*\*>

<UML:ModelElement.namespace>

<UML:Namespace xmi.idref=”luc – 555abc222”/>

</UML:ModelElement.namespace>

<UML:ModelElement.visibility xmi.value =”public”/>

</UML:UseCase>

As tags acima representam um caso de uso no documento, o primeiro ator neste caso está associado à este caso de uso, mas o mesmo não é representado através do caso de uso, como citado anteriormente a ordem de gravação no documento de dá à ordem de inserção do elemento no modelo abstrato. Até aqui podemos verificar que o documento coloca os elementos separadamente independente de suas ligações, isso facilita o encontro dos mesmos e pegar dados pertencentes aos elementos separadamente da sua associação.

<UML:Association xmi.id=”bbb -555abc222 ” name = “Nome+Associação(Opcional)” \*\*>

<UML:ModelElement.namespace>

<UML:Namespace xmi.idref=”lc – 555abc222”/>

</UML:ModelElement.namespace>

<UML:ModelElement.visibility xmi.value =”public”/>

<UML:Association.connection>

<UML:AssociationEnd xmi.id=”ccc – 555abc222” name=”(Opcional)” \*\*\*\*\*\*\*\*\* >

<UML:ModelElement.namespace>

<UML:Namespace xmi.idref=”lc – 555abc222”/>

</UML:ModelElement.namespace>

<UML:ModelElement.visibility xmi.value =”private”/>

<UML:Feature.owner>

<UML:Classifier xmi.idref=”aaa -555abc222”/> #MESMO CASO DE USO

</UML:Feature.owner>

<UML:Feature.visibility xmi.value=”private”/>

<UML:StructuralFeature.multiplicity>

<UML:Multiplicty xmi.id=”luc – 555abc222”>

<UML:Multiplicity.range>

<UML:MultiplicityRange xmi.id=”jjj -555abc222” lowerValue=”” upperValue=””/>

</UML:Multiplicity.range>

</UML:Multiplicty>

</UML:StructuralFeature.multiplicity>

<UML:AssociationEnd.participant>

<UML:Classifier xmi.idref=”yur -555abc222”/>#MESMO DO ATOR

</UML:AssociationEnd.participant>

<UML:AssociationEnd.association>

<UML:Association xmi.idref=”bbb -555abc222”/> #MESMO DA ASSOCIAÇÃO

</UML:AssociationEnd.association>

<UML:AssociationEnd.visibility xmi.value=”private”/>

</UML:AssociationEnd>

<UML:AssociationEnd xmi.id=”cch – 555abc222” name=”(Opcional)” \*\*\*\*\*\*\*\*\* >

<UML:ModelElement.namespace>

<UML:Namespace xmi.idref=”lc – 555abc222”/>

</UML:ModelElement.namespace>

<UML:ModelElement.visibility xmi.value =”private”/>

<UML:Feature.owner>

<UML:Classifier xmi.idref=”yur -555abc222”/>#MESMO DO ATOR

</UML:Feature.owner>

<UML:Feature.visibility xmi.value=”private”/>

<UML:StructuralFeature.multiplicity>

<UML:Multiplicty xmi.id=”vvv – 555abc222”>

<UML:Multiplicity.range>

<UML:MultiplicityRange xmi.id=”opp -555abc222” lowerValue=”” upperValue=””/>

</UML:Multiplicity.range>

</UML:Multiplicty>

</UML:StructuralFeature.multiplicity>

<UML:AssociationEnd.participant>

<UML:Classifier xmi.idref=”aaa -555abc222”/>#MESMO DO CASO DE USO

</UML:AssociationEnd.participant>

<UML:AssociationEnd.association>

<UML:Association xmi.idref=”bbb -555abc222”/>#MESMO DA ASSOCIAÇÃO

</UML:AssociationEnd.association>

<UML:AssociationEnd.visibility xmi.value=”private”/>

</UML:AssociationEnd>

</UML:Association.connection>

</UML:Association>

<UML:AssociationEnd xmi.idref=”ccc -555abc222”/>

<UML:AssociationEnd xmi.idref=”cch -555abc222”/>

Esta é uma parte crucial para a construção do parser, ela é a associação de dois elementos no nosso caso os elementos são o Ator e o Caso de uso, caso haja alguma outra conexão mesmo que seja ao mesmo caso de uso ela será arquivada em outra associação. Neste caso é uma ligação simples entre ambos, ligações como includes e extends ainda não foram analisadas e não encontramos possíveis diferenças, mas para o parser o importante é a ligação entre elementos A e B e não sua ordem, e até agora nada foi mencionado ainda relacionado ao seu tipo de ligação dando assim o mesmo tratamento para as 4 (Simples, Include, Extend, Generation). Nesta parte do documento podemos verificar como é expressada a associação entre elementos no documento XMI, onde os dados importantes para o entendimento estão destacados, Para cada associação temos um proprietário do recurso destacado pela tag FeatureOwner , nela se contém em quem está ligando e na Participant contém os dados de quem está sendo ligado, pode se verificar que esta ligação é representada tanto do caso de uso para o ator , quanto do ator para o caso de uso. E cada ligação desta é representada por uma associação final diferente com números identificadores diferentes, mas a associação é mencionada dentro das tags AssociationEnd, para ilustrar a qual associação este final pertence.

**Parser Atualizado - Padrão (Diagrama de casos de uso) INCLUDES, EXTEND’S e GENERALIZATION.**

Includes, extends e generalization são casos de dependências dentro da UML, ou seja, um elemento depende do outro. Includes e Extends só podem ser realizados somente entre Casos de Uso, já no caso de generalization podem ser feitas tanto entre casos de uso quanto entre Atores.

O funcionamento do include dentro da semântica do diagrama de casos de uso significa que quando SEMPRE que um caso de uso “X” for executado o caso de uso “Z” também será executado. Sabendo dessa afirmação temos um elemento base, ou seja, que através dele o outro será executado, que no nosso exemplo é o caso de uso “X”, e temos também um elemento que é uma adição ao elemento base, que voltando ao exemplo me refiro ao caso de uso “Z”. Neste tipo de dependência o sentido da ligação é do elemento base para o elemento de adição.

O funcionamento do extend dentro da mesma semântica diz que quando um caso de uso “X” for executado, um caso de uso “H” PODE ser executado não necessariamente. E neste caso ainda temos o elemento base que a partir dele se é possível o acesso a nossa extensão, que neste caso continua sendo o nosso elemento “X”, e temos a extensão que é o caso de uso “H” que pode ser executado ou não quando se executa o seu caso de uso base, sua execução DEPENDE de um evento ou critério específico. Nesta dependência o sentido da ligação é o contrário do include, não parte do caso de uso base e sim da sua dependência que neste caso é o seu extend.

O Generalization se difere das demais dependências por também poder acontecer entre Atores, ela tem o seu significado igual ao da Herança. Sua ligação está do elemento filho para o seu “progenitor”, onde este filho herda todos as permissões do seu respectivo “progenitor”.

Diferentemente das associações que não possuem uma importância em saber o seu sentido as dependências possuem uma grande necessidade de salvar os seus elementos base diferentes do seu elemento dependente. Para isto criei uma classe UmlDependency como base e herdando desta eu criei uma classe para armazenar um include e um extend, generalization será elaborado nesta próxima semana.

Obs.: Nesta parte se deve ficar atento para não se confundir, devido o sentido das ligações includes e extends. Mas de acordo com os nomes Base e Addition ou Extension é possível diferenciar os elementos base dos dependentes facilmente e assim colocar dentro da estrutura criada.

**Busca por diagrama dentro da lista - Parser XMI**

Dentro do documento XMI, cada elemento, associação dependência e até mesmo o modelo possui um número identificador único que contém letras e números. Exemplo de id: z2-fbodabcvd5a54ewSA. A parte em vermelho até acontece de se repetir de um diagrama para o outro, mas nunca dentro do mesmo. E a parte em Azul não se repete nunca de um diagrama para o outro mas dentro do mesmo ela é fixa. Sabendo destas informações podemos concluir que a parte em vermelho é o que difere um elemento do outro dentro do documento XMI, e a parte em azul é comum para cada elemento e diferente de outros diagramas.

Para realizar esta busca é necessário do id do elemento, associação ou dependência que se deseja procurar o seu respectivo diagrama. No caso da minha pesquisa ainda é feita uma pesquisa linear simples, onde se compara somente as partes do id que são comum dentro do diagrama, e com isto é possível encontrar o diagrama que se necessita para fazer as inserções e exclusões necessárias.

**Verificação da Associação END - Parser XMI**

Para se fazer esta verificação é simples, dentro da associação end possui uma tag que referencia o id da associação que a engloba, caso estes sejam diferentes, significa que possui algum erro, caso contrário está tudo em ordem.

**Diagrama de Atividades**

O diagrama de atividades foi criado com o objetivo de ilustrar a sequência de atividades, e tendo suporte para mostrar paralelismo e também comportamentos condições (decisão). Os comportamentos condicionais são ilustrados pelos losangos que são denominados “branches” (caso receba uma transição de entrada e duas de saída, ou seja, passa de uma para duas atividades, importante observar que cada transição de saída pode ser tomada por vez, onde se tem uma condição caso ela seja verdadeira tomará um caminho do branche caso contrário tomara outro caminho “else”) ou “merge” (caso receba duas transições de entrada e somente uma de saída, ou seja, é no merge que se marca o final de um branche).OBS.: Não é necessário sempre explicitar o losango para desvios e intercalações, pois uma atividade pode ter muitas transições de saída e de entrada, sua utilização deixa mais claro o desvio e a intercalação.

Já para fazer separações onde o comportamento é em paralelo é utilizado “Forks” (Ele consiste de uma barra onde possui uma transição de entrada e múltiplas saídas no mínimo 2, isto significa que quando executarmos a transição de entrada as de saídas podem ser executadas paralelamente, que não significa que vão ser executadas simultaneamente. Quando se tem separações fork isto quer dizer que a sequência entre elas não importam e o que realmente importa é os caminhos que individuais que cada separação vai ter. Como exemplo podemos ter dois departamentos que esteja trabalhando para a entrega de um produto onde um determinado departamento pode tomar conta da embalagem enquanto o outro da emissão da nota fiscal, não importa a ordem desde que os caminhos individuais que cada um conclua o seu objetivo) e junções “Joins”(Depois de uma separação fork precisamos sincronizar novamente as separações criadas, a junção join só faz a transição de saída caso as duas transições de entrada tenham sido concluídas)OBS.: Quase sempre que houver um fork haverá um join, esta regra só não é aplicada caso tenha uma transição condicional saindo de um fork onde pode ocasionar me 2 joins.

Um diagrama de atividades também permite que você escolha a ordem a ser seguida (atividades) , determinando as regras essenciais de sequência devem ser realizadas para que se possa acessar tal atividade. A decomposição de atividades é um fator que pode aumentar a granularidade de detalhes dividindo uma atividade em subatividades, onde quem decidirá fazer esta decomposição será será o analista.

**Análise XMI - Diagrama de Atividades**

<UML:Pseudostate> → Este elemento pode referenciar tanto um nodo Inicial, quanto um nodo de decisão(branche e merge).

É composto por Id e name, o nome contém o tipo como (InitialNode ou DecisionNode dependendo de qual Pseudostate está sendo referenciado), portanto a partir do name é possível saber o seu tipo. Ele tem a referência do CompositeState(composto de estados do diagrama) e também referência as transições ligadas ao elemento, podendo ser 1 de saída caso mesmo seja um InitialNode, ou 3 podendo ser 2 de entrada e 1 de saída(Merge), ou 1 de entrada e duas de saída(Branche).

Mesmo que a partir do nome podemos verificar o seu tipo, o mais correto é verificar a partir de um atributo do elemento (kind) que contém o seu tipo sendo “junction” para Branche e Merge, e “initial” para Nodos iniciais.

<UML:ActionState> → É uma atividade do sistema ela contém um ID e Name, e também tem referência das transições de saída(outgoing) e entrada (incoming). Este elemento também contém outro elemento dentro do mesmo que diz qual é a ação no qual o mesmo ta entrando(ainda não implementado).

<UML:FinalState> → Representa o final do diagrama, onde contém apenas os dados de Nome , ID e transição de chegada.

<UML:Transition>→ Representa as transições entre os elementos do diagrama. Também é composta por um ID e name que pode ser vazio.

**JSON(JavaScript Object Notation)**

A Notação de objetos JavaScript é uma formatação leve para intercâmbio de dados. Sendo que a mesma é de fácil compreensão para as pessoas e de fácil analisação e criação por programas. Sendo uma formatação totalmente independente da linguagem de programação sendo compreendido por diversas linguagens de programação, fazendo da mesma uma ótima opção para o intercâmbio de dados.

O JSON possui duas estruturas onde uma delas é uma divisão entre nome/valor, onde em algumas linguagens podem equivaler à objetos, struct dentre outras caracterizações. A outra forma de organizar os elementos seriam uma lista ordenada de valores como em um array, vetor , lista, sequence.

Como as estruturas citadas estão presentes na maioria das atuais linguagens de programação. Para um formato independente é aceitável a utilização destas estruturas já que a maioria suportam tais estruturas.

No formato citado os objetos são representados a partir da abertura de chaves “**{**“ e o encerramento é a partir do fechar da mesma “**}**”. Um objeto é constituído de um conjunto desordenado de pares, estes pares são de nome/valor, onde cada nome é seguido por “**:**” e os pares são separados por “**,**”.

Um array é representado a partir da abertura de um colchete “**[**“ e seu encerramento é realizado a partir do fechamento do mesmo “**]**”. Um array é um conjunto ordenados de valores, onde cada valor é separado por “**,**”. Um valor pode ser uma String, object, number, array, false, true, null.

**Exemplo de JSON:**

[

{ “id”: “A”, “type”: “UMLAssociation”, “name”: “AssocA”,

“ladoesquerdo”: “B”, “ladodireito”: “C”,

{“id”:”B”, “type”: “UMLActor”, “name”: “ActorB”},

{“id”:”C”, “type”: “UMLUseCase”, “name”: “UseCase1”}

}

]

JSON (JavaScript Object Notation)

Introdução:

O JSON é um modelo de armazenamento e transmissão de informações no formato de texto. Mesmo sendo muito simples é muito utilizado em aplicações Web devida ser mais compacto do que o XML, fazendo assim um parser para a notação ser mais veloz do que um para um documento XML.

Sintaxe:

Sua forma de representar as informações é muito simples. Para cada valor é atribuído um rótulo que o representa( descreve o seu significado ). Obs.: Esta forma é derivada de como o JavaScript representa tais informações.

Um par nome/valor o nome é representado entre aspas duplas(“ ”), seguido de dois pontos( : ), e após disto segue o valor onde o mesmo pode ser de apenas 3 tipos básicos: String, Númerico(real ou inteiro) ou Booleano. Obs.: também pode se ter representação de objetos nulos.

Exemplos:

Representação do ano 2016 = “ano”: 2016

Representação de um número real = “valor”: 1.55

Representação de uma String= “site”: “www.google.com”

Representação de um número negativo= “temperatura”: -1

Representação de um booleano= “estudante”: true

Representação de um objeto nulo= “dinheiro”: null

A partir dos tipos básicos podemos formar tipos mais complexos que são eles Arrays e Objetos. Os Arrays são delimitados por colchetes( [ ] ) com seus elementos separados por vírgula. Enquanto os objetos são delimitados por chaves( { } ) e os elementos também são separados por vírgula. Com isto um objeto pode representar qualquer coisa virtualmente como qualquer tipo de informação e isto é interessante para a transmissão de dados do parser para qualquer outra linguagem ou ferramenta.

Exemplos:

Representação de um array de inteiros= [1, 2, 3, 5, 4]

Representação de uma matriz de inteiros= [

[1,2,4],

[3,2,1],

[4,5,6]

]

Representação de um objeto=

{

“titulo”: “JSON(JavaScript Object Notation)”,

“resumo”: “Introdução sobre JSON”,

“tipos de valores”: [“String”, “numerico”, “booleano”, “null”]

}

**JSON x XML:**

Sabendo que ambos formatos tem a função de transmissão de dados e neste requisitos se tornam concorrentes então quais são as semelhanças entre cada uma? E as principais diferenças?

Semelhanças: \* Representam informação em forma de texto

\* Natureza auto-descritiva

\* Capacidade de representar informações complexas, difícil de representar de forma tabular (objetos compostos: Objetos dentro de Objetos, relações de hierarquia, array, atributos multivalorados dentre outras possibilidades).

\* Ambos podem transportar informações em aplicações AJAX

\* Ambos são considerados padrões de representação de dados. XML é um padrão da W3C, enquanto o JSON foi formalizado na RFC 4627

\* Ambos são independentes das linguagem de programação, ou seja, podem ser acessados por qualquer linguagem de programação, através de API específicas.

Diferenças: \* JSON não é uma linguagem de marcação , e nem possui abertura e fechamento de tags

\* As informações de um JSON não podem ser validadas através de esquemas como em XML por exemplo que pode ser validado através de esquemas XML ou DTD(document type definition)

\* JSON possui uma representação mais compacta das informações

\* JSON não permite execução de instruções de processamento como em XML

\*JSON é destinado somente para o transporte de dados enquanto o XML possui muitas outras utilidades. Atualmente existem banco de dados armazenados em XML e estruturados através de SGBD’s nativo XML.

Porque a utilização do JSON e não de um XML? Devido a forma do JSON representar os dados ser mais compacta, fácil, rápida e também praticamente todas as linguagens de programação possuem um parser para tal linguagem. Mesmo o XML sendo uma linguagem mais rica e com um maior grau de maturidade para tal atividade é melhor a utilização da notação JSON.

Obs.: para java possui mais de 20 parsers de JSON definidos e uma biblioteca que é a “gson”, diferentemente do XML que possui API’s já nativas em java que é o DOM e SAX onde basta utilizar, e possuem sua forma de analisar bem definida enquanto para o JSON, não possuem nada tão bem definido por isto quantidade de parsers.

Site com mais detalhes:

https://tools.ietf.org/html/rfc4627

DTD = Definição de tipo de documento, define se o mesmo é da família SGML(HTML, XML, SGML)

API = Interface de programação de aplicativos

AJAX = É uma forma de utilizar o JavaScript, trabalhando com o DOM(Document Object Model), CSS(Cascading Style Sheets) e XML. O AJAX trabalha de forma assíncrona com o XML, por isto não é necessário o reload na página que o mesmo está utilizando. Para o servidor isto não muda de forma alguma, entretanto agora o mesmo retornará para a página apenas os dados que página precisa sem marcação ou apresentação, com a utilização do ajax o usuário percebe que grande parte da página não é alterada, somente as partes que necessitam atualização.

**(MBT) - Model Based Testing - Project Summary**

Para a concretização de testes automatizados baseados em modelos UML é necessário que se possa analisar como os elementos, associações estão se relacionando dentro de um modelo, além de outros dados importantes para tal tarefa como dados de testes. Para conseguirmos fazer uma análise de um modelo, foi utilizado uma extensão do XML criado pela W3C para a representação de diagramas UML, que é o XMI, com o mesmo é possível pegar dados dos elementos, transições e dados de testes armazenados nas TaggedValue, sendo o formato mais utilizado para a representação se torna possível exportar o diagrama necessário a partir de uma ferramenta de modelagem. Obs.: Como o XMI é uma extensão XML que é uma linguagem extensível, o mesmo possui algumas liberdades que fazem tal documento variar da ferramenta escolhida para a modelagem, sabendo disto a ferramenta escolhida foi o Astah.

Com o Astah fazemos a exportação do xmi, já com documento em mãos a tarefa era fazer um Parser que possa analisar o documento e armazenar os dados, em classes ou estruturas que nos desse a liberdade de manipular seus dados, para criar casos de testes eficazes e transportar os mesmos para qualquer ferramenta geradora de testes. Para gerar estes dados dois diagramas foram necessários, o diagrama de casos de uso onde que com o mesmo se pode ter uma ideia geral de como o sistema a ser testado funciona, e o diagrama de atividades que representa os passos que o mesmo realiza para a execução de tais casos de testes. No momento o Parser já efetua ambas leituras faltando apenas algumas modificações em relação aos diagramas de atividades.

Após a análise se fez necessário o transporte destes dados para uma ferramenta geradora de testes, para isto se pode usar o XML que nos daria suporte para tal tarefa, entretanto mesmo que ele consiga representar dados complexos o documento não é compacto, e se torna um problema para o transporte de uma grande quantidade de dados. Sabendo disto veio a necessidade da utilização de um formato de fácil interpretação e extração de dados, que fosse compacto e que representasse igual ou melhor do que o XML os elementos dos diagramas, e foi escolhido o JSON que é uma Notação de Objetos em JavaScript que possui uma forma bem mais compacta, e de fácil analise quanto para humanos quanto para programas. O JSON já é bastante utilizado em navegadores para o transporte de dados, fazendo a navegação na internet mais agradável e dinâmica quando usado juntamente com o AJAX, além de conter vários analisadores em praticamente todas as linguagens atuais de programação o que torna ainda mais fácil o transporte destes dados para as ferramentas geradoras de testes, que poderão ter sido implementadas em qualquer linguagem praticamente.

Então após a análise o Parser deve criar um documento no formato JSON para intercâmbio destes dados(ainda não implementado), com isto pronto estes dados de testes serão enviados para a ferramenta SeleniumIDE para que a mesma gere os casos de testes e os execute sem a utilização da técnica de Capture&Replay bastante utilizada atualmente.