Capítulo 1

Cloud computing: elásticas e seguras

Rodrigo Elia Assad[1,2], Vinicius de Melo Rocha[1,2], Emanuell Faustino[2], Felipe Silva Ferraz[1,2] e Silvio Romeiro Lemos Meira[1]

[1]C.E.S.A.R - Centro de Estudos e Sistemas Avaçãdos do Recife {assad@cesar.org.br, vmr@cesar.org.br, Felipe.ferraz@cesar.org.br }
[2]Universidade Federal de Pernambuco {rea@cin.ufpe.br, vmr@cin.ufpe.br, efhl@cin.ufpe.br, fsf3@cin.ufpe.br, srlm@cin.ufpe.br }

Abstract

With increasing connectivity speed and Web systems evolution, emerges the Internet systems, which are more commonly called cloud computing. It´s designates a support platform that provides: management, on-demand use, fitness requirements, rational use of resources and automation of processes related to creation of infrastructure and support that should be secure .

Resumo

Com o aumento da velocidade de conectividade e evolução dos sistemas WEB, começa a surgir os sistemas de Internet, que mais comumente são chamados de Computação nas Nuvens. Este termo designa uma plataforma de suporte a sistemas de software que provê aos seus usuários: gerenciamento, uso sob demanda, adequação às necessidades, racionalização do uso dos recursos e automação dos processos relacionados a criação de infra-estruturas de suporte e que devem ser seguros.

1.1 Introdução a Cloud Computing

Atualmente, Computação nas Nuvens (*Cloud Computing*) tem sido um dos temas mais discutidos na área da Ciência da Computação, tanto como tópico de pesquisa, como de interesse de investimento para as empresas. Os conceitos associados à Computação nas nuvens começaram a ter uma maior visibilidade em meados do ano de 2008, fundamentando-se em 2009. Como fatores que impulsionaram a disseminação destes conceitos e idéias associadas, tem-se a crise econômica e a necessidade de racionalização dos custos de tecnologia da informação e comunicação (TIC).

Computação nas nuvens pode ser vista como uma plataforma de suporte a sistemas de software que provê aos seus usuários: gerenciamento, uso sobre demanda, adequação as

necessidades, racionalização do uso dos recursos e automação dos processos relacionados a criação de infra-estruturas de suporte.

A idéia central do uso de computação nas nuvens está em tornar a infra-estrutura de suporte às aplicações o mais programável possível. Este novo paradigma faz com que os usuários/clientes das empresas que provêm serviços em *clouds* paguem somente pelo consumo real de recursos, sendo os mesmos disponibilizados de imediato a medida que precisem de mais capacidade.

Abaixo estão listados os principais benefícios obtidos com o uso de Computação nas Nuvens

- Agilidade: permite aos usuários crescer a sua infra-estrutura de forma rápida;
- Custo: diminuição dos custos, uma vez que os usuários passam a pagar pelo que utilizam e não mais por toda a infra-estrutura;
- Independência de dispositivos e localização: os usuários passarão a utilizar um *browser* para acessar os seus recursos;
- Compartilhamento: como a idéia básica é que os usuários paguem apenas pelo que utilizem, logo o compartilhamento dos recursos computacionais se faz necessário;
- Disponibilidade: os sistemas precisam estar disponíveis a qualquer momento, logo é necessário redundância;
- Escalabilidade: os sistemas devem possuir escalabilidade horizontal, ou seja, a capacidade de se expandir apenas adicionando mais servidores.
- Segurança: devido a centralização dos sistemas, fica mais fácil implementar questões de segurança
- Manutenção: por não ser necessário instalar os aplicativos nos computadores dos usuários, fica mais fácil realizar a manutenção e atualização dos sistemas.
- Medição: com a centralização o monitoramente torna-se mais fácil, ajudando na tomada de decisão de quando se deve ou não adquirir mais infra-estrutura.

Como se pode observar, estes benefícios são interessantes do ponto de vista de quem desenvolve os sistemas, de quem os administra e de quem os mantêm. Eles permitem que novas tecnologias sejam adicionadas ao dia a dia da empresa sem a necessidade de um investimento elevado.

Dentre as tecnologias que dão suporte ao crescimento elástico dos sistemas computacionais na nuvem, destacam-se: virtualização, *peer-to-peer* e grids computacionais. A junção destas tecnologias associado a estratégias de gerenciamento, segurança e a compreensão básica do funcionamento das aplicações, permitem a criação de um arcabouço de sustentação para os modelos de solução descritos acima, tendo como desafio a exigência de prover todas elas de forma segura.

1.1.1 Principais benefícios

Além da redução de custos, *Cloud Computing* provê aos seus usuários: gerenciamento, uso sobre demanda, adequação as necessidades, racionalização do uso dos recursos e automação dos processos relacionados a criação de infra-estruturas de suporte.

1.2 Sistemas P2P

1.2.1 Arquiteturas P2P

Os grandes responsáveis pelo impulso e popularização dos sistemas distribuídos P2P foram o Napster [13] e Gnutella [14]. Além de protagonizar inúmeras questões judiciais quanto a direitos autorais e pirataria, foi através destas ferramentas que se evidenciou o potencial da tecnologia no que diz respeito a compartilhamento de recursos sem desprender maiores investimentos em hardware. Desde a época desses precursores, a tecnologia P2P vem sofrendo diversas mudanças. Esse tipo de sistema computacional encontra-se classificado dentre alguns tipos de sistemas e com algumas subdivisões, como pode ser observado na Figura 1.

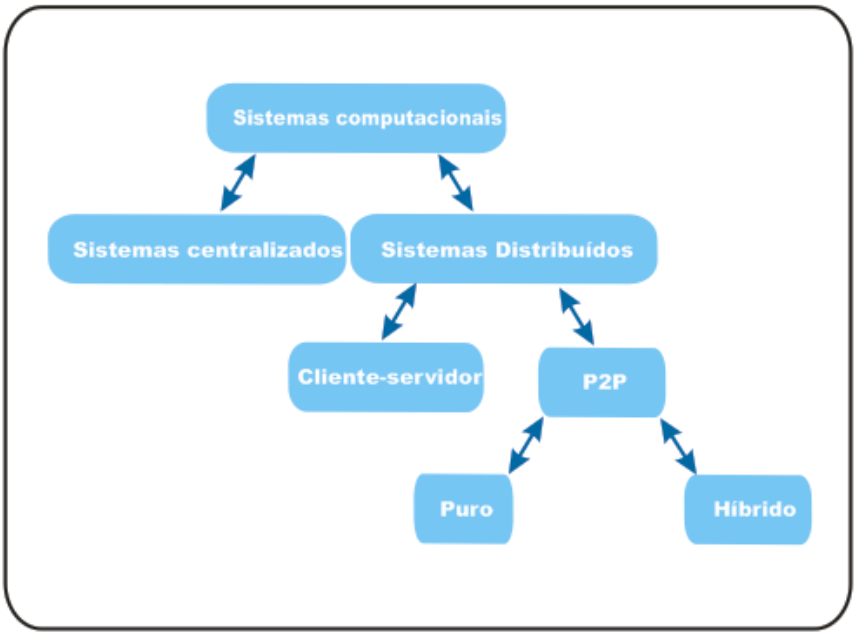

Figura 1 Taxonomia de sistemas P2P

A ilustração mostra que os sistemas tendem a sofrer um processo de descentralização contínuo, onde o primeiro grande avanço se deu com o modelo cliente-servidor. Este por sua vez, consiste numa máquina central que disponibiliza algum tipo de serviço consumido por outras máquinas. A segunda forma de descentralização já mostra o surgimento das aplicações P2P, classificadas por sua forma completamente descentralizada (ou pura) e um modelo híbrido. É comum nos sistemas P2P híbridos a existência de estruturas de controle centralizadas para a utilização de recursos descentralizados. Cada um dos modelos de descentralização possuem suas vantagens e desvantagens. Essas características são abordadas nas seções seguintes.

1.2.2 Arquiteturas puras

As redes denominadas puras possuem como principal característica a não existência de controle central. Todo o funcionamento se dá com uso de um algoritmo descentralizado onde é possível localizar *peers* e/ou serviços [58].

Esta localização é feita fazendo uso da técnica de enchente (*flooding*)[58], onde a mensagem é enviada a todos os computadores diretamente ligados ao emissor e cada máquina que recebe a mensagem faz o mesmo. Para evitar que a rede entre em colapso (*loop*), um tempo

de vida é atribuído à mensagem para os casos em que ela não chegue ao seu destino. Os pontos negativos aqui são:

- Algumas máquinas podem não receber a mensagem, negando assim um serviço que estaria disponível em tese;
- Há um grande volume de mensagens enviadas até que a mesma encontre a máquina de destino.

Para melhor ilustrar o funcionamento, é possível visualizar na Figura 2 o algoritmo de busca em execução. É possível perceber que algumas máquinas não repassam a mensagem recebida devido a limitações no tempo de vida e número de repasse.

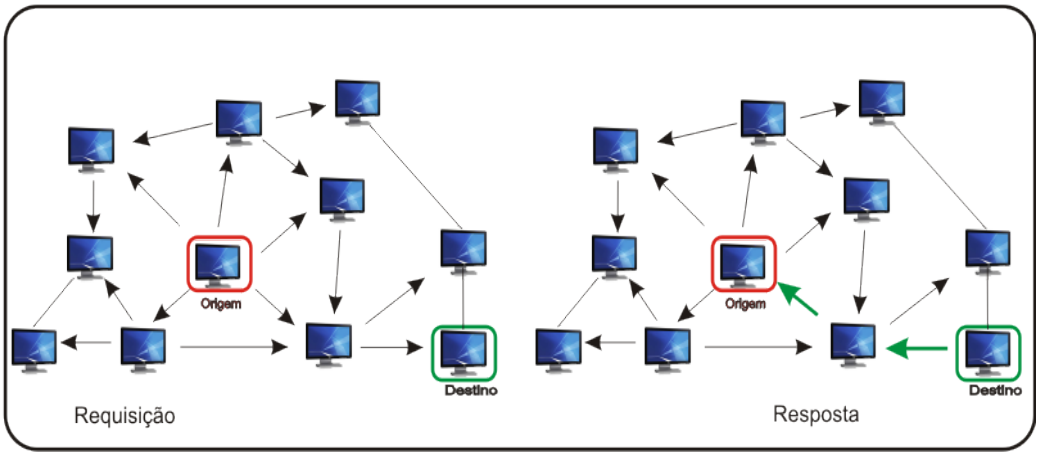

Figura 2 Exemplo de requisição usando a técnica de flood.

A técnica mais utilizada na implementação de arquiteturas puras, o qual gerou um avanço significativo na área, é o *Distributed Hash Tables* (DHT)[20]. Utilizadas nas ferramentas pStore [1], Pastiche [2], Oceanstore [3], PeerStore[25] e BitTorrent[15]. As DHTs pertencem à classe de sistemas distribuídos descentralizados e oferecem recursos de localização similar às *hash tables* (chave, valor). Um par de chaves e valor é armazenado na DHT e qualquer participante da rede pode acessar o valor desejado apenas informando a chave associada. As primeiras quatro especificações de DHTs (Pastry [16], Chord [17], CAN[18] e Tapestry [19]) surgiram quase simultaneamente no ano 2001, depois disso, sua utilização se popularizou em aplicações destinadas ao compartilhamento de arquivos na internet. Um estudo mais aprofundado sobre as diferentes implementações de DHT pode ser encontradas em outros trabalhos [20].

As DHTs possuem como principais características:

- Descentralização: os próprios nós criam e mantêm o sistema, sem a necessidade de um servidor;
- Escalabilidade: o sistema suporta a participação de um crescente número de nós simultaneamente;
- Tolerância a erros: o sistema deve ser confiável, mesmo com nós entrando e saindo continuamente da rede.

Para alcançar os objetivos supracitados, as redes DHTs utilizam a técnica de que um nó na rede deve estar em contato direto com apenas uma fração de todos os nós participantes. Isso reduz o custo de manutenção quando um nó entra ou sai do sistema.

Para armazenar um arquivo numa DHT, primeiro se calcula uma chave (geralmente o código *hash* SHA-1 [21] do seu nome ou do seu conteúdo), em seguida esse arquivo é enviado para a rede até ser encontrado o conjunto de nós responsáveis por seu armazenado. Para recuperá-lo, uma mensagem é enviada informando a chave do arquivo desejado. Essa mensagem por sua vez é encaminhada até um nó que possui o conteúdo desejado, que é enviado como resposta. A seguir é descrito uma das implementações de DHT mais utilizadas: o Chord [17].

1.2.3 Arquitetura Chord

A implementação de DHT utilizando Chord se destaca pela sua simplicidade em oferecer uma única operação em que dada uma determinada chave, ela será mapeada para um nó na rede. Segundo Stoica [17], sua arquitetura foi projetada para se adaptar facilmente à entrada e saída de novos *peers* na rede.

Embora seja uma implementação que possui apenas uma tarefa (associar chaves a nós da rede), o Chord possibilita ao desenvolvimento de aplicações P2P uma série de benefícios:

- Balanceamento de carga: as chaves são distribuídas igualmente entre os nós da rede;
- Descentralização: nenhum nó é considerado mais importante que outro;
- Escalabilidade: o uso do Chord é viável mesmo com uma grande quantidade de nós;
- Disponibilidade: ajuste de sistema automático à entrada e saída de novos nós, fazendo que um nó sempre esteja visível na rede;
- Flexibilidade: não é necessário seguir nenhuma regra para o nome das chaves.

Na Figura 3 [17] é ilustrada a localização e armazenamento de dados que podem ser facilmente desenvolvidos acima da camada de Chord, associando a chave à informação no nó a que ela faz parte no sistema.

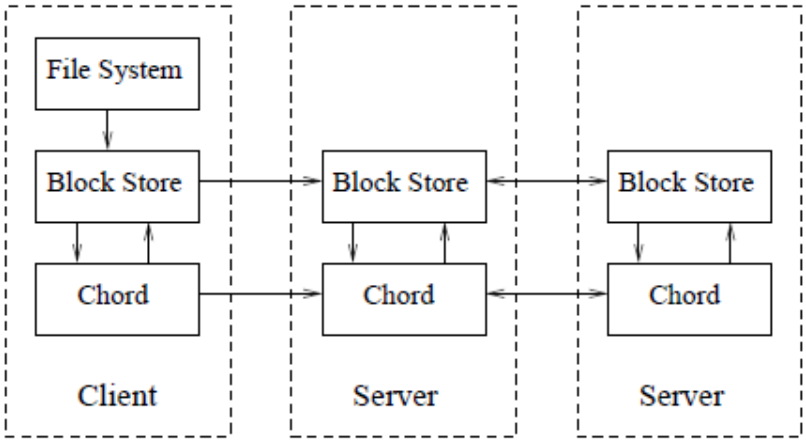

Figura 3 Sugestão de arquitetura de um sistema de armazenamento utilizando Chord. [17]

Para atribuir chaves aos nós, o Chord usa uma variação de *consistent hashing* [22] que cuida do balanceamento de carga, uma vez que cada nó tende a receber naturalmente um mesmo número de chaves. Em trabalhos anteriores ao Chord usando *consistent hashing,* se assumia que cada nó possuísse conhecimento de todos os outros, diferentemente, no Chord cada nó precisa ter conhecimento de apenas uma fração dos outros nós na rede.

Supondo uma rede de *n* nós, um *peer* mantém informações sobre *O(log n)* nós, e para encontrar um determinado nó na rede basta que ele possua apenas uma referência válida.

1.2.4 Arquiteturas híbridas

Em alguns sistemas P2P é necessária a identificação dos *peers* conectados na rede. Para tal, sistemas como o OurBackup[1], que fazem uso de redes sociais para *backup*, utilizam em sua arquitetura um servidor responsável pela autenticação dos usuários, manutenção da rede e dos metadados onde as cópias estão armazenadas. Pode-se ressaltar que a utilização de servidores não é obrigatória para a localização dos *peers* e dos metadados, podendo essa ser feita utilizando as DHT mencionadas anteriormente.

Nesses sistemas, o papel do servidor está em oferecer uma interface aos *peers* da rede com diversas operações tais como: autenticação do usuário; manipulação dos dados armazenados por outros *peers* (adicionar, remover, excluir, atualizar); manipulação dos usuários cadastrados no sistema; localização dos *peers* e relacionamento entre eles; dentre outras tarefas que venham a atender os requisitos do sistema.

Essa centralização de informações pode trazer prejuízos de escalonamento no sentido de que sempre vai existir uma exigência maior do servidor à medida que se aumenta o número de requisições, usuários, metadados ou quaisquer outras informações delegadas ao servidor. Porém, sistemas como o eDonkey [23] se mostraram bastante eficientes quanto ao gerenciamento centralizado de informações dessa natureza. Se necessário esse escalonamento também pode ser resolvido com a utilização de *clusters* ou *grids* no lado do servidor, fazendo-se mais importante a disponibilização da interface de comunicação com a máquina cliente.

1.2.5 Segurança e compartilhamento de pedaços de arquivos

Em busca de garantir a segurança e a possibilidade de que diferentes usuários possam compartilhar os mesmos blocos (pedaços) de arquivos diferentes, o identificador de cada bloco é calculado utilizando-se a seguinte formula: $ID = h(h(c))$; onde c é o conteúdo do bloco e h é uma operação de criptografia *hash*. Possibilitando, assim, a identificação e compartilhamento, de forma segura, dos blocos semelhantes, mesmo que seu conteúdo não seja conhecido.

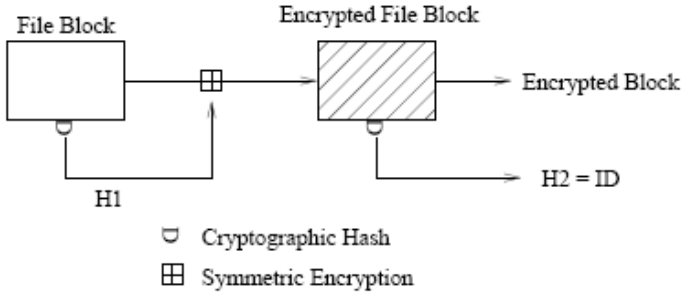

Figura 4 Criptografia dos blocos e geração do identificador. [25]

Na Figura acima se pode ver uma ilustração do modelo descrito acima, na proposição do sistema PeerStore [25]. Nessa abordagem o valor *hash* calculado a partir do conteúdo do bloco é usado como chave para criptografia do mesmo. Por essa razão, esse valor não é utilizado diretamente como identificador, por isso um novo valor *hash* é calculado. Dessa forma, apenas os *peers* que geraram o bloco podem ter acesso ao seu conteúdo, uma vez que geraram o mesmo identificador para blocos semelhantes.

1.2.6 Verificação da consistência dos blocos

De forma a verificar a consistência dos blocos que foram armazenados remotamente, um *peer* deve desafiar periodicamente os seus parceiros para saber se ainda estão guardando as cópias dos blocos a eles confiada. Contudo, solicitar que todos os blocos sejam recuperados apenas para teste não se mostra viável, uma vez que o consumo de banda de rede para tal fim seria muito alto. Assim como o Samsara [8], o sistema PeerStore [25] propõe a forma descrita na figura abaixo para a solução desse problema.

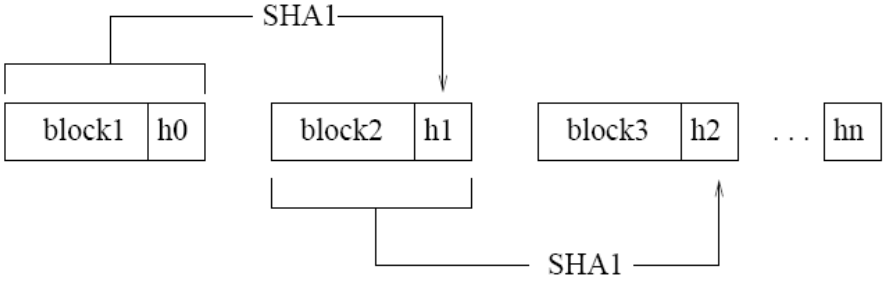

Figura 5 Verificar consistência dos dados armazenados apenas enviando h0 e hn. [25]

Nessa técnica, o *peer* que deseja verificar a consistência dos dados envia um valor único *h0*, assim como uma lista dos *n* blocos em questão. Para responder, o *peer* desafiado deve concatenar o valor *h0* ao conteúdo do primeiro bloco e calcular o código *hash* desse novo valor, gerando assim *h1*. O novo valor h1 é então concatenado ao conteúdo do segundo bloco onde um novo valor *hash h2* será calculado, e assim sucessivamente até que o valor de todos os blocos seja verificado gerando como resultado final *hn*. Esse valor final é então enviado como resposta provando, assim, a presença de todos os blocos.

1.2.7 Disponibilidade

Segundo a norma ISO/IEC 9126 [3], a confiança pode ser definida como sendo a "capacidade do produto de software de manter um nível de desempenho especificado, quando usado em condições especificadas".

Avaliar se um sistema é confiável ou não envolve analisar dados quantitativos e qualitativos. Com base nesse contexto, Sommerville [1] classifica essa avaliação em quatro pilastras. São elas:

- **Disponibilidade**: é a probabilidade de um sistema, em determinado instante, ser operacional e capaz de fornecer os serviços requeridos:
- **Confiabilidade**: é a probabilidade de operação livre de falhas durante um tempo especificado, em um dado ambientem para um propósito especifico;
- **Segurança**: é um atributo que reflete a capacidade do sistema de operar, normal e anormalmente, sem ameaçar as pessoas ou ambiente;
- **Proteção**: é a avaliação do ponto em que o sistema protege a si mesmo de ataques externos, que pode ser acidentais ou deliberados;

Disponibilidade e confiabilidade são em essência probabilidades e, portanto são expressos em valores numéricos. Já segurança e proteção são feitos com base em evidências na organização dos sistemas e normalmente não são expressos em valores numéricos. Frequentemente são medidos através de níveis, e a ordem de grandeza destes denomina se um sistema é mais ou menos seguro/protegido do que outro.

Oliveira [1] concorda com Sommerville [1], quando define disponibilidade em sistemas de *backup* P2P como sendo a probabilidade de recuperação de uma informação num determinado instante *t*. Logo, quando se fala sobre armazenamento de dados em sistemas P2P, sua natureza distribuída contribui para uma maior disponibilidade, uma vez que a distribuição geográfica intrínseca dos pontos de rede reduz as chances de falha simultânea por acidentes catastróficos. No entanto, computadores em uma rede P2P estão disponíveis apenas uma fração de tempo, ou estão sujeitos as mais diversas falhas, o que pode trazer prejuízos a disponibilidade dos dados armazenados. Para reduzir esse risco e aumentar a disponibilidade do sistema, faz-se necessário o uso de redundância dos dados, seja por replicação ou usando técnicas mais elaboradas como o *erasure codes* [2].

1.2.8 Replicação dos dados

A replicação é a mais tradicional forma de garantir disponibilidade, onde *k* cópias idênticas de um dado é copiada em diferentes instâncias dos pontos de rede. O número *k* deve ser definido de forma a aumentar a probabilidade de recuperação de alguma informação dada a probabilidade de um *peer* que a possua estar disponível no instante desejado. O aumento probabilístico da disponibilidade é obtido somando-se as probabilidades de apenas um nó estar disponível, dois nós estarem disponíveis, três nós estarem disponíveis, até *k*. [1] Ou, esse somatório pode ser das probabilidades de as cópias terem falhado no lugar de as cópias estarem disponíveis. Sendo assim, essa disponibilidade pode ser descrita segundo a seguinte equação, onde *f* é a falha e *k* a quantidade de cópias:

$$C = \sum_{i=1}^{k} \binom{k}{i} (1-f)^{i} (f)^{k-i}$$

Equação 1 Cálculo da probabilidade de que uma informação esteja disponível da rede.

Na tabela a seguir são mostrados os resultados obtidos na aplicação da formula.

Tabela 1 Resultados obtidos na aplicação da fórmula para 20% (esquerda) e 30% (direita) de falha.

f	K	C
20%	4	0.998400
20%	5	0.999680
20%	6	0.999936
20%	7	0.999987
20%	8	0.999998

f	k	C
30%	6	0.999271
30%	7	0.999781
30%	8	0.999934
30%	9	0.999980
30%	10	0.999994

Note que para uma taxa de falha de 20% se podem obter cinco noves de precisão efetuando oito cópias da informação, e para uma taxa de falha de 30% podemos obter um resultado bastante similar aumentando o número de cópias em duas.

1.3 Sistemas em Grids

1.3.1 Conceitualização

A literatura sobre *Grid Computing* abrange diversas definições sobre este conceito. Porém, não há uma concordância geral entre os termos utilizados. Para Aidouni *et al*. [21] um *grid* computacional é definido como uma infra-estrutura que envolve o uso colaborativo e integração de computadores, redes, banco de dados e instrumentos científicos pertencentes e gerenciados por múltiplos domínios administrativos visando compartilhar recursos computacionais além dos limites institucionais, a fim de prover uma plataforma poderosa e distribuída de altíssima escala, com um custo muito inferior quando comparada a um supercomputador paralelo.

Por outro lado, para Ian Foster [22], Grid é uma infra-estrutura de software e hardware para integrar recursos computacionais, dados e pessoas dispersas geograficamente formando um ambiente colaborativo distribuído.

Já Krauter [23], define *grid* como um sistema de rede que pode escalar ambientes do tamanho da internet com máquinas geograficamente distribuídas através de múltiplas organizações e domínios administrativos.

1.3.2 Abordagem Histórica

Na década de 80, as primeiras manifestações de computação em grade consistiam em aplicativos que utilizam o paradigma de computação paralela para a promoção de mecanismos de comunicação entre processadores, proporcionando um grande potencial para realizar compartilhamento de memória [24].

A fim de possibilitar novos achados científicos, a comunidade científica se viu com o desafio de trabalhar em pesquisas multidisciplinares nas quais envolviam grandes quantidades de dados a serem processados, disponibilizados e apresentados [24]. A chave principal desse desafio estava em como promover uma infra-estrutura computacional em rede de larga escala e de baixo custo [24][25].

No início da década de 90, impulsionados pelo uso de rede de computadores e computação de larga escala, surgiram uma gama de projetos que faziam uso de computação distribuída. Em 1995, durante uma conferência promovida pela IEEE e ACM sobre computação paralela, foram utilizados 11 redes de computadores para demonstrar a criação de um supercomputador distribuído geograficamente [26]. O projeto foi conduzido por Ian Foster e recebeu o nome de I-Way (*Information Wide Area Year*), ficando conhecido como o primeiro modelo de implementação de grades computacionais [25].

O sucesso do projeto incentivou diretamente a criação de mais de 70 aplicações, como também sua continuidade, que foi patrocinada por diversas instituições governamentais norte-americanas [26]. Consequentemente, no ano seguinte, a DARPA (*Defense Advanced Research Projects Agency*), desenvolveu e financiou um projeto de pesquisa para a construção de uma ferramenta de computação distribuída em larga escala conhecida como GLOBUS, conduzido também por Ian Foster e pelo pesquisador Carl Kesselman [25][24].

Durante a Conferência de Supercomputação(SC98), um fórum foi estabelecido para estudar, discutir e implementar padrões e melhores práticas no uso de grades computacionais. Em 2000, o eGrid (*European Grid Fórum*) e o *AsianPacific Grid Fórum* se uniram e fundaram o *Global Grid Fórum*. Anos mais tarde, ocorreu outra fusão entre a *Global Grid Fórum* e a *Enterprise Grid Alliance* (EGA), que resultou na fundação do *Open Grid Fórum* (OGF) [27].

O *Open Grid Fórum* é uma comunidade de usuários e desenvolvedores que lideram estudos de padronização para grades computacionais. A comunidade consiste em milhares de pessoas especializadas na indústria e na pesquisa, representando mais de 400 organizações em mais de 50 países ao redor do mundo.

1.3.3 Visão Geral sobre Grids

O termo *grid* surgiu nos Estados Unidos como referência a rede de distribuição de energia elétrica (*Power grids*). Semelhante a essas redes, existe uma transparência de como o recurso utilizado está sendo de fato produzido ou armazenado. Como não é possível armazenar a energia elétrica distribuída pela rede, ela deverá ser consumida ou será perdida.

Analogamente a essas redes, os ciclos de processamento de CPU também não podem ser armazenados. Consequentemente, em períodos de ociosidade do processador, o processamento poderia ser efetuado sem o usuário final saber onde exatamente a computação está sendo executada.

A infra-estrutura de ambas as redes são heterogêneas e são formadas por diferentes componentes: usinas de produção de energia elétrica, linhas de transmissão, transformadores e, do outro lado, computadores, servidores, conexões de rede, dispositivos de armazenamento. Adicionalmente, tanto redes elétricas como *grids* computacionais são pervasivas, ou seja, o usuário pode ter acesso ao recurso desejado em qualquer parte da rede onde exista um ponto de entrada, como ilustrado na figura 6 [28].

Figura 6 Visão Geral de Grid Computacional

Este modelo distribuído de arquitetura computacional traz diversos benefícios além do compartilhamento dos ciclos de CPU e espaço de armazenamento. Ele também provê a capacidade de processar paralelamente aplicações subdivididas em partes menores, que podem ser distribuídas em milhares de computadores ao mesmo tempo visando aumentar o desempenho dessas aplicações. Essa abordagem se mostra muita interessante do ponto de vista da confiabilidade do sistema, visto que a ocorrência de falha em pontos específicos do *grid* não compromete o funcionamento total da rede [29].

Para controlar o acesso a todos os recursos de um *grid* é necessária uma camada de *software* que fica localizada entre a interface com o usuário e o sistema operacional. Essa camada de software é denominada *middleware.* Ele é geralmente constituído por módulos com APIs de alto nível, que proporcionam a sua integração com aplicações desenvolvidas em diversas linguagens de programação, e interfaces de baixo nível que permitem a sua independência do restante da rede [29], como está ilustrado na figura 7.

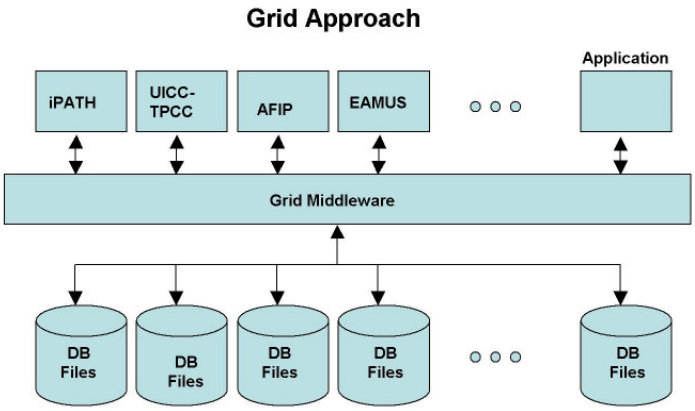

Figura 7 Camada de Software Middleware

Segundo Ian Foster [30], existem três características básicas que compõem *Grids* Computacionais:

1. Um *Grid* não possui um controle centralizado, onde as questões de segurança, políticas de acesso e controle do uso de recursos são consideradas dentro dos diferentes domínios que constituem o *Grid;*
2. O *Grid* é formado por protocolos abertos e padronizados para proporcionar alto grau de interoperabilidade no ambiente;
3. O *Grid* deve fornecer QoS (Qualidade de Serviço) com o propósito de atender diferentes requisitos de tempo de resposta, vazão, disponibilidade, segurança e alocação de diversos recursos dentro de um sistema heterogêneo e complexo.

A ideia de compartilhamento de recursos não é nova, pois a própria *web* pode ser considerada como um serviço de compartilhamento de informações sobre a camada da internet. Desde as iniciativas pioneiras em grades computacionais até os projetos mais recentes, todos procuram explorar a ociosidade das máquinas para realizar análises e cálculos que seriam feitos exclusivamente em um único servidor central.

1.3.4 Organizações virtuais

Grupos que estão dispersos geograficamente e que estão dispostos a doar tempo de processamento ocioso para a resolução de problemas podem ser agrupados no que chamamos de organizações virtuais (*Vitual Organization* - VO). Uma VO pode ser comparada a um *cluster* ou um agrupamento de computadores que estão sob o mesmo domínio administrativo, no qual são determinadas regras de compartilhamento de recursos. É importante ressaltar que a união de várias VO's através de redes de computadores (LAN – *Local Area Network* ou MAN – *Metropolitan Area Network*) é o que realmente forma uma *grid* na sua forma mais conceitual. Na figura 8 abaixo é ilustrado a formação de um *grid* computacional com a disposição de várias VO's ao redor do mundo.

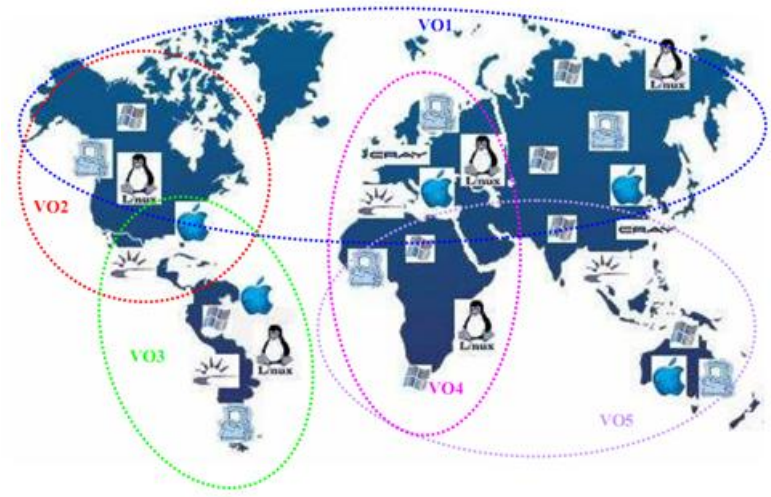

Figura 8 Organizações Virtuais formando uma grid

1.3.5 Componentes de uma grade computacional

Durante a evolução do estudo e implementações de grades computacionais, diversas arquiteturas foram propostas sem seguir nenhuma padronização comum (comunicação, dados, plataformas e protocolos), gerando um impacto negativo na interoperabilização dos sistemas.

Com a melhoria e aperfeiçoamento dos protocolos e tecnologias aplicadas na Internet e nas próprias grades computacionais, se estabeleceu uma componentização mínima necessária para o estabelecimento de ambientes de grade, sendo formado por: interface do usuário, segurança, agente de busca de informações, escalonadores, gerenciamento de dados e gerenciamento de tarefas e recursos. Essas funcionalidades serão melhor descritas nas próximas seções.

1.3.6 Interface do Usuário

Um ambiente de *grid* computacional pode disponibilizar diversos tipos de interface com usuário. Dentre as mais utilizadas, é possível destacar: interpretador de comandos, aplicação cliente pré-definida e portais de serviços (*web services*).

A alocação, distribuição e execução de tarefas através de um interpretador de comandos é o mecanimo mais primitivo porém mais usado e preferido por usuários mais experientes. Através dele é possível inserir comandos ou até criar *scripts* que serão executados e interpretados por um interpretador comumente chamado de *shell*.

As aplicações cliente dedicadas a exibição da interface gráfica são aplicativos *desktop* que rodam na estação do usuário e possuem um canal de comunicação com a grade. Esse tipo de aplicação executa uma lista limitada de tarefas e comandos simples, requisitando ou executando-os na grade. Um exemplo desse tipo de operação seria a submissão de dados por meio de aquivos em lote (sistemas *batch*) ou utilizando XML.

Portais *web* são as interface mais utilizadas na atualidade devido a expansão da Internet em todo mundo. Um portal de grade pode disponibilizar uma série de serviços para uma determinada comunidade, necessitando apenas que os usuários cumpram os requisitos mínimos de acesso. O mais interessante desse tipo de interface é que os administradores podem incluir ou remover serviços dinamicamente sem interferência nas estações dos usuários.

1.3.7 Segurança

Para entender os problemas de segurança envolvidos em sistemas distribuídos que utilizam os conceitos de *grids*, é conveniente apresentar um cenário genérico que ilustre um ambiente de computação distribuída e seus recursos envolvidos em instituições distintas. Na figura 9 é apresentado este cenário.

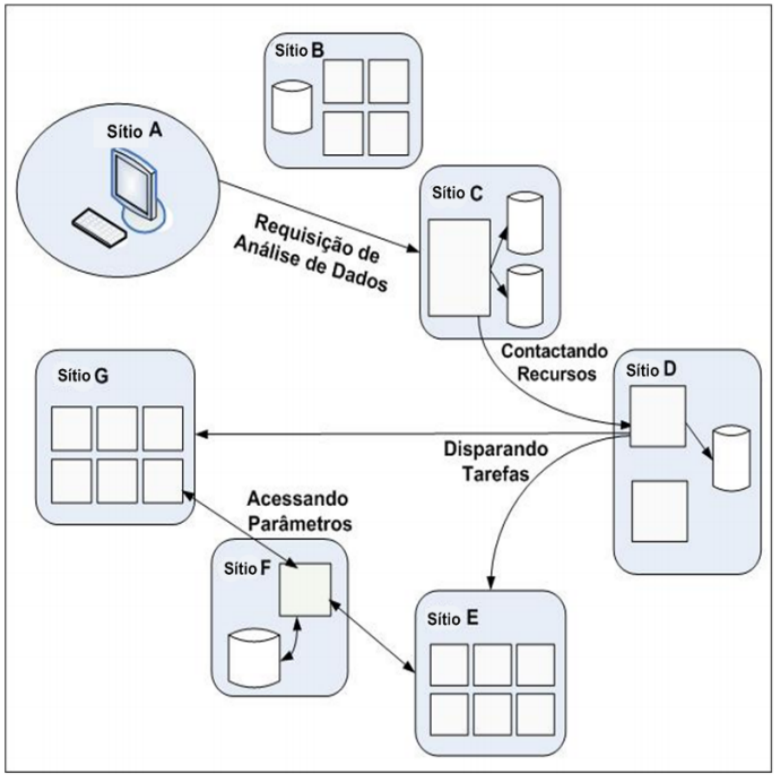

Figura 9 Arquitetura de segurança para grids

Dentro desse cenário é possível observar que existem sete sítios (A-G) que dispõem de recursos e necessidades distintas. Os processos de comunicação entre esse sítios (representados pelas setas de interligação) são realizados da seguinte maneira: A solicita um procedimento de análise de dados para C; C conecta-se com D para adquirir informações sobre recursos; D solicita e distribui tarefas para G e E; G e E consultam parâmetros em F.

Foster descreve os seguintes pontos a serem considerados num cenário clássico de funcionamento de uma *grid* computacional:

1. Os usuários das organizações multi-institucionais formam um grupo extenso e dinâmico. Os participantes desses grupos, constituintes da organização virtual, podem ser alterados, incluídos ou excluídos dependendo da necessidade e da quantidade de recursos disponíveis pela instituição.
2. A disponibilidade de recursos é vasta porém dinâmica.
3. O processamento dos recursos computacionais apresentam características distribuídas.
4. Os processos podem criar e destruir conexões (*unicast ou multicast*) dinamicamente em tempo de execução.

5. Os recursos podem ser acessados por diferentes formas de autenticação, autorização e política devido a divergência do ambiente de segurança empregado pelas instituições participantes.

A política de proteção de uma infra-estrutura de grade visa previnir ataques aos recursos disponibilizados pela rede, congestionamento de recursos e a inclusão de dados espúrios que impossibilitem a legitimidade da informação. Consequentemente, o ambiente deve contemplar políticas de autenticação, controle de acesso, integridade, confidencialidade e irretratabilidade.

1.3.8 Agentes de Busca de Informações

O agente de busca de informações, ou *Broker,* é o componente responsável por manter um banco de informações sobre a disponibilidade de recursos e serviços que fazem parte da rede, além de informar aos usuários conectados. Também compete ao *Broker* a atividade de controlar a informação do estado (disponível, trabalhando, ocupado, indisponível ou falho) e da localização de recursos através de descrições publicadas de nome e de diretório (DNS e LDLAP).

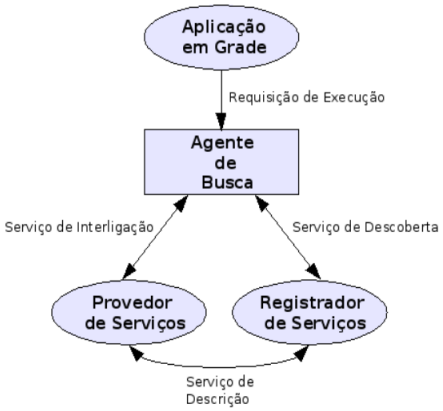

Figura 10 Agentes de busca

A figura 10 apresenta como o *Broker* realiza a busca por informações de recursos disponíveis na rede para execução de uma tarefa específica. Quando há uma necessidade por parte da aplicação de realizar essa busca, uma requisição é disparada para o agente de busca que, por sua vez, consulta o provedor e registrador de serviços em busca da disponibilidade do recurso no ambiente.

1.3.9 Escalonadores

Escalonadores são componentes responsáveis pela coordenação da execução dos processos computacionais. Esses componentes são aplicados exaustivamente em sistemas operacionais multiprogramados para o escalonamento de processos e execução de aplicações paralelas.

Um exemplo de atividade que um escalonador realiza é a alocação de tarefas computacionais para um conjunto pré-definido de unidades de processamento. Esse método é normalmente empregado em *clusters,* onde os processadores são, na maioria dos casos, homogêneos.

Os escalonadores desenvolvidos ou adaptados para o uso em grades computacionais são conhecidos como meta-escalonadores (*metascheduler*). Os meta-escalonadores são compostos por ferramentas para designar prioridade de execução, re-submissão de tarefas e escalonamento multinível. Para a execução desse tipo de tarefa, os componentes da rede recebem informações sobre a capacidade do processamento disponível e informações sobre o estado dos recursos.

Os escalonadores para grades computacionais são compreendidos como sendo o escalonador global dos escalonadores locais e equipamentos que compõem a infra-estrutura. Pelo fato da interação e submissão de tarefas na grade computacional, um meta-escalonador precisa ter compatibilidade com ferramentas de segurança, sistemas de informação e ambiente de execução da grade. A figura 11 apresenta a visão esquemática de uma arquitetura que é aplicado metaescalonamento.

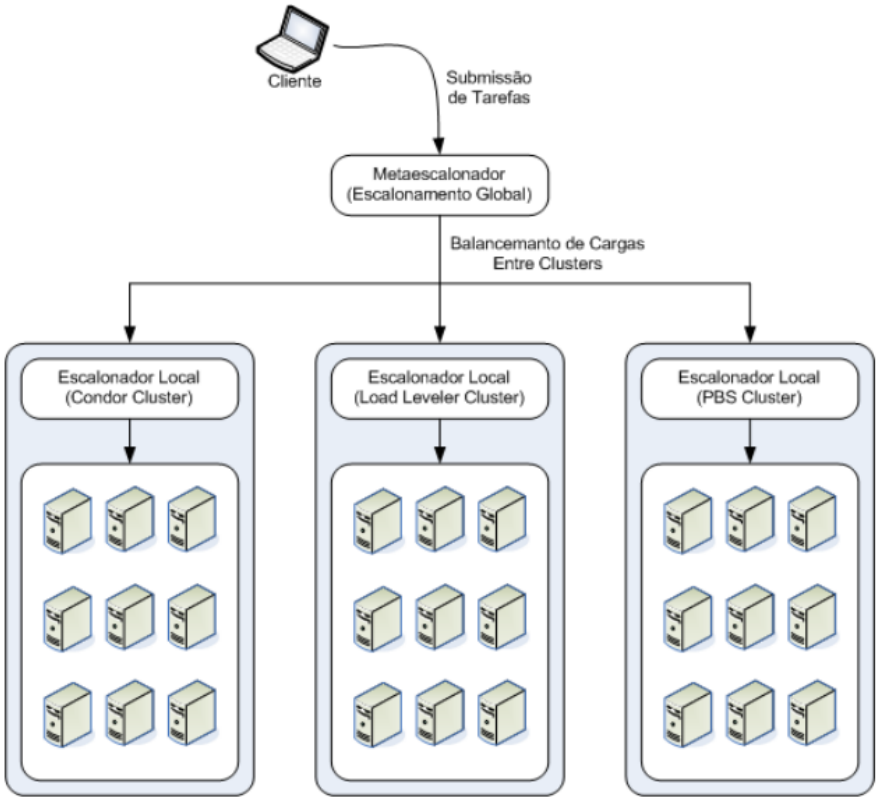

Figura 11 Visão geral de uma arquitetura com metaescalonador

1.3.10 Gerenciamento de Dados

O gerenciamento de dados ainda é um desafio para as aplicações em grades computacionais, pois o modelo de gerenciamento de dados é dependente da taxonomia ou finalidade do ambiente. Dentro de uma grade computacional podem ser encontrados diferentes tipos e formatos de dados que devem ser operados de forma comum. Alguns exemplos dessa variabilidade são: arquivos convencionais, base de dados relacionais, base de dados XML, objeto de dados, dados virtualizados, setores de dados e metadados.

A gerência e a movimentação de arquivos ou diretórios fica a cargo dos protocolos convencionais utilizados na Internet, que passaram por uma adaptação para se adequarem ao ambiente de grades computacionais, como é o caso do GridFTP.Nos demais tipos de dados se torna indispensável uma camada de software adicional (*middleware*) que possibilite a manipulação de tais informações com mais transparência. A função básica dessa cama de

software é compreender o provedor de informações (i.e. SGBD) e transformar essas informações num padrão comum, tal como os documentos em XML.

Na figura 12 é mostrado um modelo de acesso à base de dados relacionais em ambientes de grade computacionais.

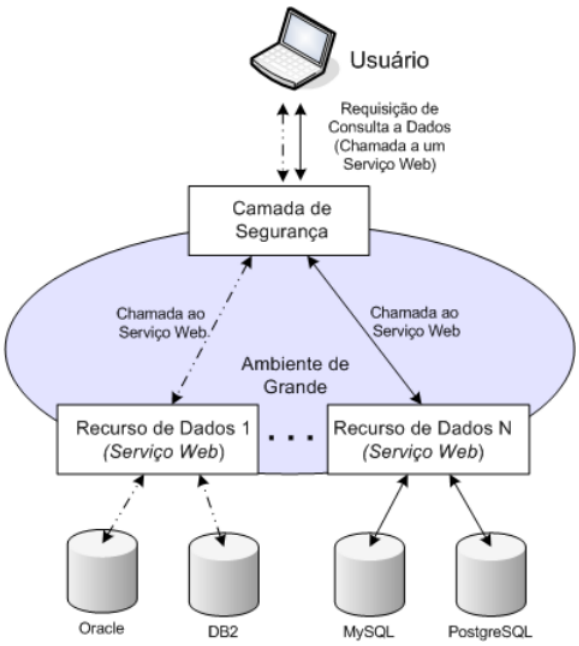

Figura 12 Gerenciador dos dados

1.3 Segurança em computação nas nuvens

Segurança em *cloud computing* não difere muito da segurança tradicional. No entanto, se deve ter mais atenção no software do que na infra-estrutura, visto que esta última, no melhor dos casos, quando se trata de um modelo de Infra-Estutura como Serviço (IaaS), estará sobre controle do desenvolvedor, com exceção do *hypervisor*.

Sendo assim, se faz necessário a proposta de uma metodologia que vise a garantia da segurança de um software que não cause grandes acréscimos de tempo e custo ao ciclo de desenvolvimento. Esse é um tema relevante e atual visto que a maioria das falhas reportadas nos sistemas estão nas aplicações e não mais na infra-estrutura, transformando a computação nas nuvens o local onde esse problema é potencializado.

No entanto, a existência de processos consolidados para garantir a segurança da infraestrutura não garante a segurança como um todo, visto que uma aplicação insegura em uma rede segura é inseguro da mesma forma.

Um sistema é considerado seguro quando a segurança está presente tanto na infraestrutura, quanto na aplicação. Porém, uma falha em qualquer um dos dois componentes e/ou em seus processos pode comprometer toda a segurança.

Considerando a evolução da Engenharia de Software, temos que qualidade de um software não é apenas garantida através da execução de um processo, mas sim a execução de testes e validação [30] no intuito de garantir que o produto entregue reflita a visão do cliente, ao menos em termos de especificações. Sendo assim, vemos práticas como TDD (*Test Driven Development*) e outras recomendações, em conjunto de processos e metodologias, sendo incorporados ao processo de desenvolvimento de software.

A Figura 13 apresenta o ciclo de segurança para um software em *cloud computing:*

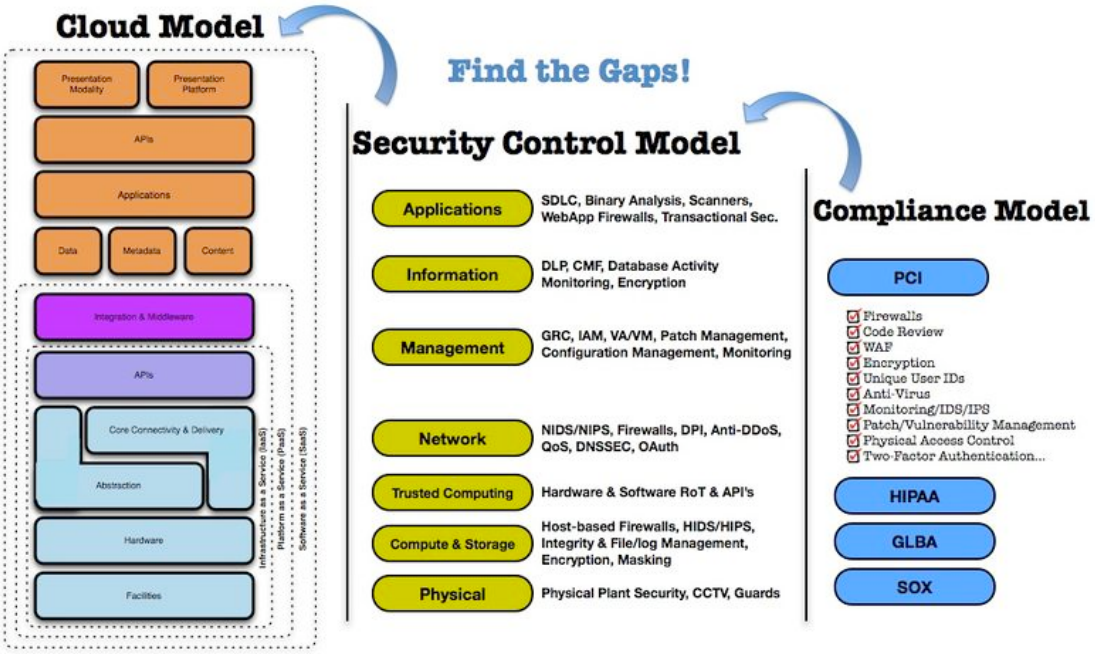

Figura 13 Segurança em Cloud Computing (Fonte: Cloud Security Aliance)

1.3.11 Processo para a garantia da qualidade

De posse das considerações feitas, a proposta de um processo de garantia da qualidade focado em segurança e que seja ágil e focado no reuso de software deve transpassar as atividades de:

a) Elaboração dos requisitos: a elaboração dos requisitos de segurança de softwares demanda profissionais mais experientes, pois exige conhecimento sobre segurança tanto para a utilização de estratégias de defesa como a execução de testes de validação da segurança. Como descrito, o conjunto de ataques mais frequentes à uma aplicação é conhecido, logo, pode-se basear neles para escrita de requisitos bem elaborados que visem a defesa das aplicações. Como os ataques seguem um padrão, é possível definir requisitos reutilizáveis, fazendo com que os mesmos venham de uma base de conhecimento sobre tecnologias, padrões de projeto, arquiteturas e ataques.

b) Teste e validação: a garantia da qualidade é dada através da execução de testes no sistema. A elaboração e automação dos testes de penetração da aplicação baseando-se nos ataques que a aplicação irá sofrer, se mostra uma prática válida para garantir a segurança. Adicionalmente, o reuso dos requisitos e dos testes é facilitado, bem como permite os desenvolvedores garantirem que o software estará protegido sobre o ponto de vista dos ataques que ele está sujeito. Não se pode garantir 100% de segurança em uma aplicação, mas com esta abordagem se pode garantir aos clientes que sobre um conjunto conhecido de testes a aplicação está segura.

c) Padrões de projeto: em relação aos requisitos de segurança sabe-se que os mesmos podem variar com a evolução do projeto, principalmente com a adoção de metodologias ágeis. O desenvolvimento de software seguro deve considerar que este

fato irá acontecer. A solução para que a segurança acompanhe essas alterações está na adoção de padrões de projeto que permitam a implementação tardia dos requisitos.

Nas seções a seguir será discutido, detalhadamente, cada uma dos itens acima que permitem a elaboração softwares seguros de forma ágil. A figura 4 abaixo apresenta como este processo pode ser aplicado para o desenvolvimento de software seguro.

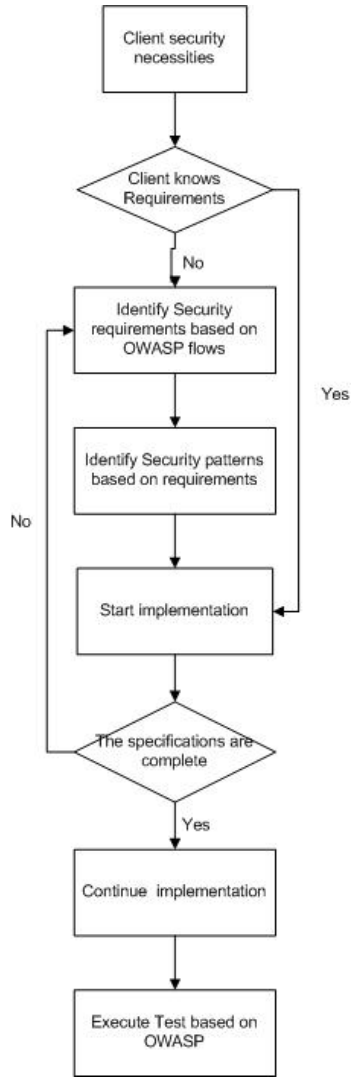

Figura 11 Processo para desenvolvimento seguro e ágil

1.3.12 Requisitos: Escrevendo requisitos de segurança

A solução proposta segue uma sequência de passos que se inicia na criação de um guia de segurança, passando para um *template* com a finalidade de facilitar a escrita dos requisitos de segurança, e como mapear, escrever e automatizar casos de testes de segurança, fechando o elo para a proposta de um processo para a garantia da segurança em projetos de software.

A figura abaixo mostra a solução proposta nesse trabalho para esta atividade, onde cada etapa mostrada será melhor descrita nas seções seguintes:

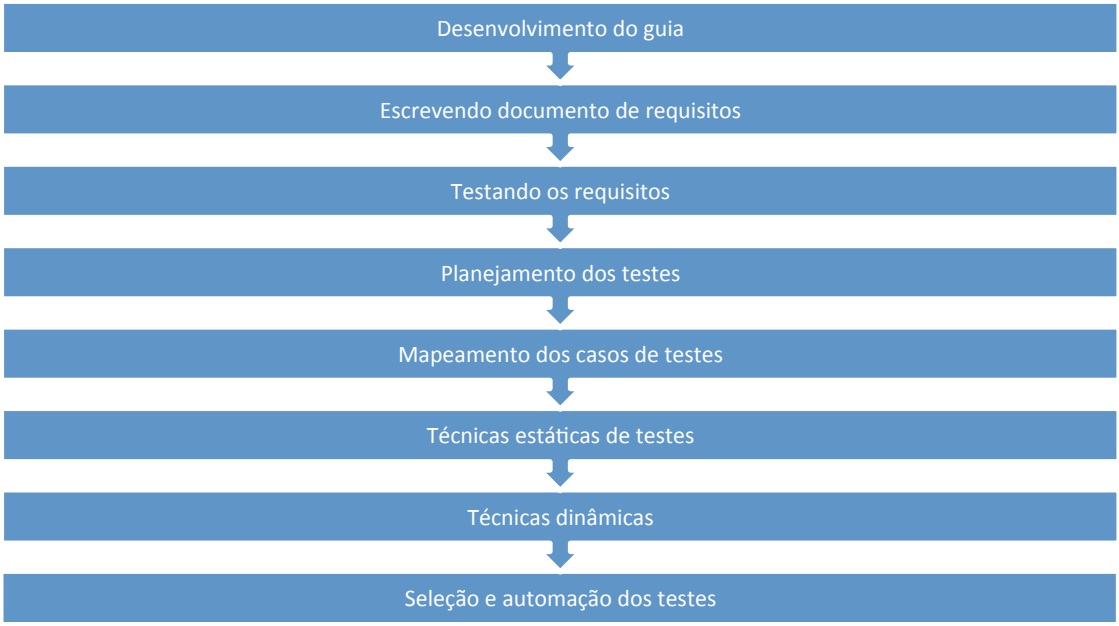

Figura 15 Solução proposta

A escolha da elaboração de um guia que dê suporte à escrita de requisitos ao invés de ir diretamente à escrita de requisitos através dos padrões para a escrita do mesmo. O guia é escrito baseado nos conceitos básicos de requisitos para web levando em consideração as necessidades de toda a equipe. Não só a análise de requisitos, mas qual a solução proposta para uma arquitetura mais segura, quais as opções de tecnologias existentes e quais são as vulnerabilidades associadas a cada requisito, sendo assim um guia prático para utilização de toda a equipe. O guia ajuda na divulgação e compartilhamento do conhecimento para toda equipe podendo ser utilizado para um passo inicial da gestão do conhecimento.

1.4.1 Guia

A criação de um guia de segurança para aplicações *web* se propõe não só a ajudar os engenheiros de requisitos a entender e detalhar as necessidades de segurança de forma mais profunda e clara, mas também para servir como referência para as outras fases do desenvolvimento do sistema. O guia foi formulado baseado nos padrões de requisitos proposto por [44] e no *template* especificação de documentação dos casos de usos desviados [31], bem como nas listas de falhas mais comuns e como testá-las em uma aplicação *web* proposto pelo OWASP [*32*].

O guia deve ser desenvolvido baseado no reuso das informações para diferentes tipos de sistemas *web*. Cada ponto do guia foi pensando para dar apoio a uma ou várias fases do desenvolvimento específico do ciclo de vida do desenvolvimento do sistema. Para isso foram formuladas algumas perguntas divididas por disciplinas, como mostra a tabela abaixo:

Requisitos
Quais são os requisitos comuns de segurança para aplicações web?

Aqui será listada uma serie de requisitos de segurança comuns às aplicações web e suas variações, de forma a ajudar o analista de sistema entender quais são as necessidades de segurança do cliente e como atingi-las.

Testes

Quais as vulnerabilidades encontradas em cada requisito?

Cada requisito ou objetivo de segurança está associado a várias vulnerabilidades e riscos que precisam ser mitigados, para isso primeiro precisamos entender quais são estas vulnerabilidades e qual o impacto da mesma no sistema.

Desenvolvimento

Quais são os métodos de soluções disponíveis?

Quais os padrões e soluções encontradas hoje no mercado e na academia para atender os requisitos levantados, e como esses se comportam e quais seus pré-requisitos.

Qual a ação para mitigar?

Agora que sabemos exatamente quais os possíveis ataques que podem ocorrer em nosso sistema, vamos analisar o risco e as possíveis formas de mitigar estes.

Qual a linguagem a ser utilizada?

Toda linguagem tem suas particularidades, desvantagens e vantagens, é importante saber quais são elas do ponto de vista de segurança das aplicações web e como podemos tratá-las e evitá-las.

Arquitetura e Design

Impactos da escolha arquitetural, frameworks e componentes?

Da mesma forma que a linguagem tem suas vantagens e desvantagens em relação à segurança, as decisões que tomamos na fase de análise e projeto, impactam de maneira significativa na segurança da nossa aplicação. Hoje existem padrões arquiteturais específicos para segurança [34]

O guia pode ser escrito e atualizado com a ajuda de toda a equipe, ou seja, a idéia é que cada perfil fique responsável pela atualização de cada ponto do guia:

- Analista de requisitos – ajudar na escrita dos requisitos de forma mensurável e atuar como facilitador da equipe;
- Engenheiro de Testes – levantamento das vulnerabilidades e como testar cada uma destas;
- Arquiteto – propor as soluções e padrões de arquitetura para as vulnerabilidades e problemas encontrados;
- Desenvolvedores – propor soluções de tecnologias e linguagens que podem ser utilizadas para mitigar os riscos levantados pela análise de impacto das vulnerabilidades.

Com o desenvolvimento do guia têm-se uma base de conhecimento para levantamento dos requisitos, além de ajudar a definir o que deve ser implementado no sistema e quais cuidados devem ter na hora de arquitetar e desenvolver o sistema baseando-se nas falhas e padrões de ataques associados a cada requisito, tendo ainda soluções de mitigação propostas para evitar ou radicalizar as vulnerabilidades e com isso reduzindo de forma significante os riscos levantados.

Para a escrita do guia, foi desenvolvido um passo a passo a fim de atender e responder as perguntas acima levantadas. Desta forma, é usado uma forma estruturada de montar e atualizar o guia.

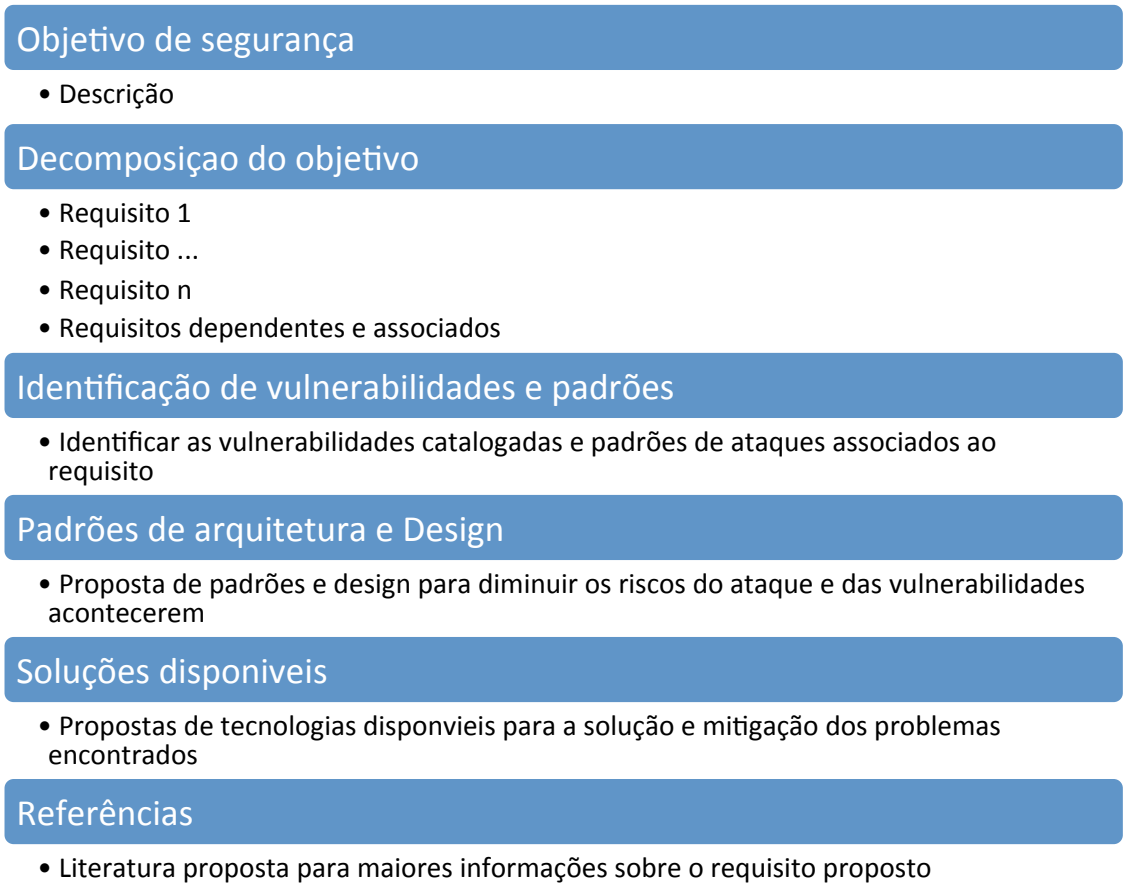

Figura 16 Estrutura do Guia de Segurança

Descrevendo a figura acima, encontram-se os seguintes pontos:

1. Objetivo – definir o objetivo do requisito de segurança. Quem vai ser decomposto em várias partes, ou seja, requisitos associados e requisitos dependentes que juntos vão suprir o objetivo de segurança;
2. Descrição – uma breve descrição do objetivo de forma geral e uma especificação do que significa para aplicações web;
3. Requisitos – decomposição do objetivo em requisitos funcionais e não funcionais, para que seja possível atingir o objetivo estabelecido no primeiro passo. Requisitos que estejam associados de forma indireta.
4. Vulnerabilidades – as vulnerabilidades relacionadas ao requisito e quais são os possíveis percas para o sistema, ocasionadas pela ameaça;
5. Padrões de arquitetura e design – padrões já catalogados na literatura;
6. Soluções disponíveis – tecnologias desenvolvidas e testadas;
7. Referências – literatura associada ao requisito.

1.4.2 Montando o documento de requisitos

Para montar o documento de requisitos vamos selecionar os objetivos de segurança e seus requisitos associados, que estão descritos no guia, e que são importantes para o sistema a ser desenvolvido. Neste caso montamos um *checklist* de acordo com o guia para que possa ser mais

fácil e usual de entender as necessidades do cliente. Neste *checklist* deve conter não só as informações contidas no guia. Algumas perguntas foram elaboradas de forma geral para ajudar no levantamento dos requisitos de segurança.

1. Existe uma política de segurança da empresa?
2. Existe um ambiente montado para a execução da aplicação?
3. Existe funcionalidades que são restritas?
4. Existe um controle de acesso centralizado?
5. Existe uma política de acesso aos dados?
6. Foi feita a classificação dos recursos e o de segurança atribuídas a ela?
7. Qual linguagem vai ser utilizada?

São perguntas que precisam ser respondidas. Caso não tenham sido pelo cliente, a equipe precisa definir quais as tecnologias serão utilizadas, como vai ser o acesso aos dados, entre outras coisas para poder desenvolver os requisitos de segurança do sistema.

Depois de respondidas essas perguntas, é preciso entender o domínio do sistema para selecionar, dentro desse domínio, quais são as necessidades de segurança do cliente. Para selecionar as necessidades é utilizado o guia de acordo com os objetivos de segurança. Por exemplo, se o sistema vai ter dados de acesso restrito, temos como objetivos a integridade dos dados, controle de acesso, e dentro deles os requisitos decompostos. O *checklist* é utilizado para selecionar os que são adequados e possíveis ao sistema a ser desenvolvido.

Tabela 2 Exemplo de *checklist* de requisitos de segurança

Objetivo: RFN001 Integridade dos dados

Requisito	Sim	NA	Comentários
RFN001_1 Transferência de dados			
RFN001_2 Classificar informações			
RFN001_3 Acessar informações			
RFN001_4 Armazenamento de dados			
RFN001_5 Validação dos dados			
RFN001_6 Encode de dados			

1.4.3 Calculando o risco

Para que seja possível a utilização de metodologias ágeis em um processo de desenvolvimento de software considerando segurança, é necessária uma lista de requisitos reutilizáveis. Caso seja necessário que os mesmos sejam considerados para implementação, que o gerente do projeto possa calcular o risco que esta nova funcionalidade que tenha, são necessários dois fatores:

- A probabilidade de um invasor explorar determinada falha depende das decisões de arquitetura do sistema, bem como de algumas questões relacionadas à implementação.
- O cálculo do impacto que uma determinada falha pode causar a um sistema e que pode ser previamente definida através do uso de metodologias.

1.4.3.1 Cálculo do impacto

Para calcular o impacto é utilizado o método disponibilizado pelo NIST: o CVSS – *Common Vulnerability Scoring System* [35]. Este consiste em três grupos: Básico, Temporal e Ambiental. Cada grupo produz um resultado numérico variando de 0,0 a 10,0, e um vetor, uma representação textual que reflete os valores utilizados para obter a pontuação. O grupo Base representa as qualidades intrínsecas de uma vulnerabilidade. O grupo temporal reflete as características de uma vulnerabilidade que mudam ao longo do tempo. O grupo Ambiental representa as características ambientais de uma vulnerabilidade que são únicas para qualquer usuário do ambiente. CVSS permite que gestores, provedores de boletim de vulnerabilidade, fornecedores de segurança, vendedores de aplicações e pesquisadores se beneficiem de uma linguagem comum de pontuação das vulnerabilidades de TI. Um exemplo de como calcular o impacto é apresentado abaixo:

Para calcular o impacto utilizamos a vulnerabilidade: V1 - Vulnerabilidade associada: Transporte de credenciais através de canal criptografado (OWASP-AT-001)

Fazendo o cálculo do grupo de métricas base:

Métricas de exploração:

1. Vetor de acesso: *Network* (rede)
2. Complexidade de acesso: *Medium* (Média)
3. Autenticação: *None* (nenhuma)

Métricas de Impacto:

1. Impacto de confiabilidade: *Partial* (parcial)
2. Impacto de integridade: *None* (nenhum)
3. Impacto de disponibilidade: *None* (nenhum)

Resultado obtido:

CVSS Base Score	7.8
Impact Subscore	7.8
Exploitability Subscore	8.6
CVSS Temporal Score	Undefined
CVSS Environmental Score	Undefined
Overall CVSS Score	**7.8**

Figura 17 Exemplo de cálculo de impacto do grupo base

Atualizando o impacto utilizando as métricas de ambiente:

Modificadores gerais:

1. Potencial danos colaterais: *Low-Medium* (Baixo moderado)
2. Alvo de distribuição: *Medium* (médio)

Sub pontuação do impacto dos modificadores:

1. Requisito de confiabilidade: *Medium* (médio)
2. Requisito de integridade: *Low* (baixa)
3. Requisito de disponibilidade: *Low* (baixa)

Resultado obtido:

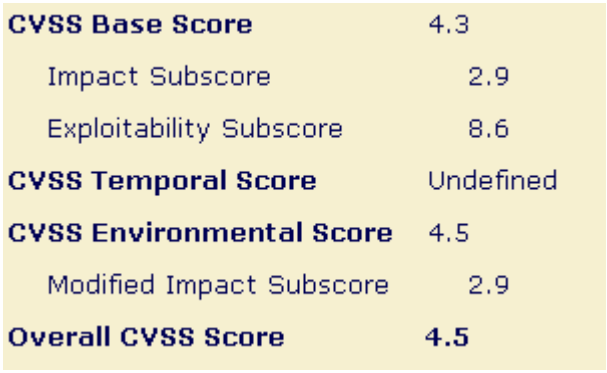

Figura 18 Exemplo de cálculo de impacto do grupo ambiental

Atualizando o impacto utilizando as métricas de tempo:

1. Exploração: *High* (alta)
2. Nível de remedição: *Oficial Fix* (Consertada oficialmente)
3. Confidencia de reporte: *Confirmed* (confirmada)

1.4.3.2 Calculando a probabilidade

O cálculo da probabilidade normalmente é feito baseado em um banco de dados ou na experiência do usuário. De acordo com o PMBOK [36] é necessário estipular uma escala de probabilidade para que o risco possa ser calculado. Nossa escala vai ser baseada na escala do impacto que vai de 1 a 10, sendo 10 a maior probabilidade e 1 a menor probabilidade.

Nosso cálculo se baseia no banco de dados de testes, todas as vulnerabilidades serão verificadas através de testes, neste caso seria P=F/E. Onde P é a probabilidade, F é número de vezes que o teste falhou e E é numero de vezes que esse teste foi executado.

Para calcular a probabilidade é necessário o uso de um histórico dos casos de testes. Dois fatores podem ser utilizados: a experiência da equipe ou listas de vulnerabilidade que classificam as vulnerabilidades mais ocorridas na *web*, como por exemplo, o Top 10 da OWASP [37].

1.4.3.3 Análise de risco

Para analisar o risco do requisito pode ser utilizado a análise qualitativa proposta pelo PMBOK [36]. Para tal é verificado o risco baseado no impacto e probabilidade da vulnerabilidade acontecer. A tabela apresentada abaixo é mostrado a classificação do risco de acordo com a escala apresentada:

Tabela 3: Tabela de calculo de risco

Tabela de Riscos					
	Impacto				
	1 – 2	3 - 4	5 – 6	7 - 8	9 – 10

Probabilidade					
9 – 10	5	9	18	36	72
7 – 8	4	7	14	28	56
5 – 6	3	5	10	20	40
3 – 4	2	3	6	12	24
1 – 2	1	1	2	4	8

Legenda:

	Baixo
	Moderado
	Alto

Para os requisitos que tem mais de uma vulnerabilidade associada deve ser considerada a vulnerabilidade que tem o maior risco.

1.4.4 Testes de requisitos

Nesta seção é apresentado como validar os requisitos escritos que podem ser feitos através de duas técnicas:

1. Revisão dos requisitos;
2. Teste de requisitos.

A primeira técnica se baseia no processo formal de revisão de documentos, como mostrado na figura abaixo:

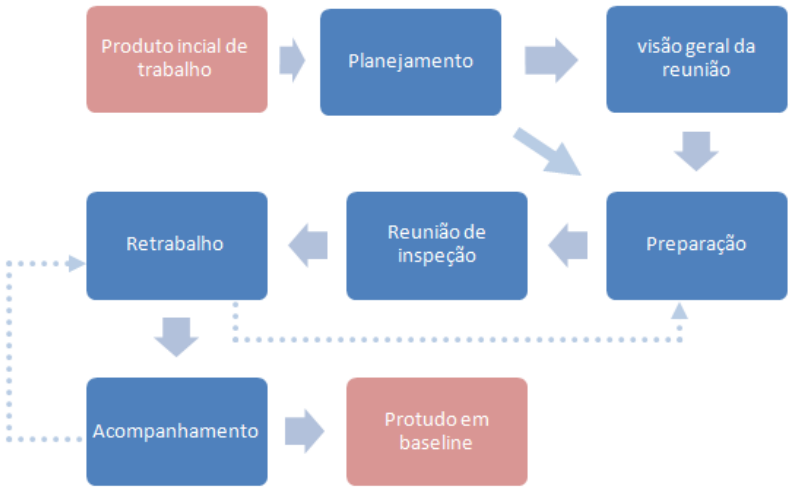

Figura 19 Processo de inspeção de requisitos [38]

Para ajudar na inspeção, um *checklist* foi proposto por [38] dividido em categorias.

Nos requisitos de segurança, é incluído mais uma categoria:

- Vulnerabilidades
  I. As vulnerabilidades associadas aos requisitos foram listadas?
  II. O impacto de cada vulnerabilidade foi calculado?
  III. As ações de mitigação foram propriamente descritas?

A outra técnica de testes de requisitos é escrever pelo menos um caso de teste para todos os requisitos, verificando assim se o mesmo é testável. Pontos a serem considerados na escrita do caso de teste:

1. O identificador do requisitos – selecionar o requisito que vai ser testado por seu identificador, pelo menos um requisito tem que estar representado;
2. Requisitos associados - requisitos que podem ser relevantes para a escrita do caso de teste.
3. Descrição do teste – escrever qual o objetivo do teste e onde ele pode ser aplicado;
4. Problemas dos requisitos – problemas encontrados nos requisitos que fazem com que a escrita do teste seja difícil ou impossível.
5. Comentários e recomendações - para a solução dos problemas de requisitos encontrados.

É importante que os requisitos de segurança sejam escritos em forma de testes para cada vulnerabilidade associada ao requisito.

1.4.5 Testes de segurança

Nesta seção é mostrado como mapear, escrever, executar e automatizar testes de segurança. De acordo com [39], a técnica mais utilizada de testes de segurança são testes de penetração e a utilização de *patch's*. Os problemas com essas técnicas são que a primeira necessita de profissionais com alto nível de conhecimento em segurança e os *patch's* podem introduzir novas vulnerabilidades.

1.4.6 Planejando os testes

Os testes devem começar o mais cedo possível. Por isso, o planejamento dos testes não deve considerar sua execução só no final. Para tal a estratégia a seguir é utilizada:

1. Revisão do documento de requisitos – os requisitos devem ser mensuráveis, claros e testáveis.
2. Revisão do código – levando em consideração os pontos que precisam ser checados descritos no guia de segurança. As ferramentas utilizadas para esta técnica será escolhida de acordo com a tecnologia escolhida para escrever a aplicação. Para isso é utilizado um *checklist* de boas práticas da linguagem e codificação do sistema, ferramentas estáticas de análise de código e ferramenta de análise de vulnerabilidades.
3. Execução manual dos casos de testes;
4. Execução automática dos casos de testes.

Existem técnicas de testes estáticas e dinâmicas. As técnicas estáticas utilizadas nesse trabalho são as revisões de documento:

1. Revisão de documento de requisitos

2. Revisão de código utilizando ferramentas estáticas

Para as técnicas dinâmicas de testes vamos utilizar:

1. Automatização de testes funcionais;
2. Testes de penetração.

1.4.7 Projeto de testes: especificando casos de testes

Após o mapeamento das vulnerabilidades nos cenários, é feita uma seleção dos casos de testes. Todos os cenários e vulnerabilidades devem ser testados, mas não tendo tempo para tal, a seleção dos casos de testes é feita baseado no nível do impacto e na probabilidade de acontecimento da falha.

A escrita dos casos de testes segue um *template* definido de maneira similar considerando os seguintes pontos:

Tabela 4: *Template* de caso de testes formal

Campos	Descrição
ID	<<Identificador do caso de teste>>
Objetivo	<<Descrever qual o objetivo do caso de teste>>
Dados de entrada	<<Quais são os dados de entrada necessários para o teste>>
Passo	<<Quais são os passo para a execução do teste>>
Resultados esperados	<<Qual o resultado esperado para cada passo executado>>
Pré-Condições	<<Em que estado o sistema deve estar para execução do teste>>

Para escrita dos casos de testes de segurança outros pontos devem ser considerados:

1. No item de Objetivos deve ser definido qual o ataque será executado;
2. Inserir o item de vulnerabilidades que fazem parte do ataque
3. Determinar onde a segurança pode ser vulnerável
4. Qual o ambiente proposto do ataque, deve ser descrito nas pré-condições;
5. Quais ferramentas podem ou devem ser utilizadas para o ataque;
6. Quais *scripts* e massa de dados devem ser utilizados para o ataque.

O *template* do caso de testes de segurança, incluindo os campos acima apresentados, ficam da seguinte forma:

Tabela 5: *Template* de casos de testes de segurança

<<Identificador do caso de teste>> – <<Descrever qual o ataque que vai ser utilizado do caso de teste>>	
Severidade: <<Alta / Média / Baixa>>	Tipo de Execução: << Manual / Semi-Automática / Automática>>
Requisito:<< Identificador do requisite>>	Vulnerabilidade: << Nome da Vulnerabilidade a ser testada>>
Pré-Condições: <<Em que estado o sistema deve estar para execução do teste>>	
Dados de entrada: <<Quais são os dados de entrada necessários para o teste>>	
Passos:	Resultados esperados:
<<Quais são os passo para a execução do teste>>	<<Qual o resultado esperado para cada passo executado>>
Ferramenta: <<Qual a ferramenta utilizada>>	Script: <<Qual Script deve ser utilizado nesse teste>>

Para que os casos de testes sejam reusáveis é proposto que os mesmos sejam escritos em alto nível, e com objetivos amplos. Para uma rastreabilidade mais fácil do testes, eles são catalogados da seguinte forma:

Tabela 6: *Template* de rastreabilidade dos requisitos x vulnerabilidades

Requisito	**Vulnerabilidade**	**Ferramenta**	**Caso de teste**	**Dados**
RF<<xx>>	V1 - <<nome da vulnerabilidade>>	F1 –nome da ferramenta necessária	CT 001<<Id do caso de teste>>	Script o1<<Dados ou arquivos ou script necessários para execução >>
			CT 002	Dados.txt
		Teste manual	CT 004	NA
			CT 005	NA

	V2 - <<nome da vulnerabilidade>>			
RF XX	V3 <<nome da vulnerabilidade>>	Ferramenta	Casos de testes	Scripts, massa de dados ou arquivos

Assim fica fácil a localização dos casos de testes reusáveis para requisitos que possuem vulnerabilidades iguais ou reutilização dos casos de testes para outros sistemas a serem testados. Os *scripts* de testes serão escritos de forma parametrizada para que sua adaptação para outros sistemas seja mínima.

1.4.8 Reportagem dos resultados e análise dos testes

A reportagem de erros é uma fase extremamente importante para o processo de testes, pois neste ponto são identificados os problemas do sistema e baseado neles, pode-se fazer uma análise da qualidade do sistema, que é um dos fatores para a liberação ou não da *release* do mesmo.

1.4.8.1 Reportando Falhas

A reportagem das falhas precisa ser precisa e fornecer a maior quantidade de informações possíveis para o que o desenvolvedor possa reproduzir a falha reportada e ter dados suficientes para entender que pontos precisam ser corrigidos. Antes de reportar um erro é preciso seguir as instruções abaixo [33]:

- Verifique se este é realmente uma falha;
- Verifique se não é uma falha já conhecida;
- Determinar um conjunto de passos confiáveis para reproduzir a falha;
- Documente quaisquer informações adicionais que o ajudarão a reproduzir a falha

Após recolher todas as informações necessárias para reportar um erro de segurança, alguns pontos, além do normal, precisam ser reportados.

1.4.9 Template

Para reportar um erro de segurança encontrado é importante que os seguintes campos sejam preenchidos [105]:

Tabela 7 *Template* do relatório de falhas

Campos	Descrição
Repórter:	<<Quem reportou o erro>>
Severidade:	<< Qual a severidade do erro>>

Descrição	<<Uma breve descrição do erro>>
Ambiente:	<<Qual o ambiente foi utilizado no teste. Por exemplo: Browser IE 7.0.5730.13>>
Build:	<<Qual a baseline do sistema foi testado>>
Caso de Teste:	<< Qual o caso de teste achou o erro>>
Pré-condição:	<< Qual a pré-condição para executar o teste. Por exemplo: é necessário está logado como usuário xy no sistema>>
Passos:	<< Quais passos foram executados para atingir o erro>>
Resultado esperado:	<< Qual o resultado esperado se não houvesse o erro>>
Resultado atual:	<< Qual o resultado atual>>
Ferramenta:	<< Qual a ferramenta utilizada>>
Impacto:	<< Qual o impacto do erro>>
Ação de mitigação ou solução:	<<Qual a ação de mitigação prevista ou solução>>
Informações adicionais:	<< toda informação que possa ajudar o desenvolvedor entender e reproduzir o erro, como: print screen da tela, logs, scripts utilizados, dados de entrada entre outros >>

Para reportagem do erro não é necessária uma ferramenta especial para reportar os erros de segurança.

1.4.10 Análise dos erros reportados

É necessário que após os erros serem reportados, um análise seja feita para determinar o quanto urgente é o conserto. Para esta análise alguns pontos devem ser considerados:

Risco - O ativo do sistema está em risco devido a esta falha de segurança?

Custo do conserto - Qual é o custo de tempo para corrigir e fazer qualquer re-testes do código se essa falha for consertada?

Custo da exploração - Quais são os custos monetários e não monetárias para a empresa, se a falha for utilizada em uma exploração de segurança? Quais são os custos para os clientes?

Efeito para Sistemas Dependentes - Quais são os efeitos sobre os sistemas dependentes caso a falha não seja consertada? Quais são os efeitos sobre os sistemas se a falha for consertada?

Após responder essas perguntas é mais fácil entender qual a urgência de consertar ou não uma falha reportada. Para responder as perguntas acima apresentadas, é utilizado como referência o documento de requisitos e o guia para saber qual o impacto, risco e perda que o sistema vai sofrer se a falha não for resolvida.

1.5 Conclusão

Computação nas nuvens (Cloud Computing), consiste de um conjunto de tecnologias que agrupadas trazem vantagens significativas para a gerenciamento dos ambientes de Tecnologia da Informação, consequentemente a diminuição dos custos.

No entanto, a idéia de que qualquer aplicação pode ser migrada para a nuvem sem a necessidade de alterações, consiste de uma visão simplista do problema. Esta afirmativa, é verdadeira desde que os gestores de tecnologia abdiquem das questões relacionadas a performance (no caso de computação nas nuvens o crescimento elástico) e segurança, caso estejam em execução em ambientes públicos.

Este capítulo de livro apresentou como se deve ser tratado tais questões em relação a aplicações existentes ou em desenvolvimento de forma que as mesmas possam atender os novos desafios colocados para as aplicações que devam estar em execução neste ambiente.

1.6 Bibliografia

[1] Batten, C., Barr, K., Saraf, A., and Trepetin, S. pStore: A secure peer-to-peer backup system. Unpublished report, MIT Laboratory for Computer Science, (2001), 130-139.

[2] Cox, L., Murray, C., and Noble, B. Pastiche: Making backup cheap and easy. ACM SIGOPS Operating Systems Review 36, (2002), 285-298.

[3] Kubiatowicz, J., Bindel, D., Chen, Y., et al. Oceanstore: An architecture for global-scale persistent storage. ACM SIGARCH Computer Architecture News 28, 5 (2000), 190-201.

[4] Landers, M., Zhang, H., and Tan, K. PeerStore: better performance by relaxing in peer-to-peer backup. Proceedings of the Fourth International Conference on Peer-to-Peer Computing, (2004), 72-79.

[5] Lillibridge, M., Elnikety, S., Birrell, A., Burrows, M., and Isard, M. A cooperative internet backup scheme. Proceedings of the USENIX Annual Technical Conference, (2003), 29-42.

[6] Manber, U. and others. Finding similar files in a large file system. Proceedings of the USENIX Winter 1994 Technical Conference, October (1994), 1-10.

[7] Muthitacharoen, A., Morris, R., Gil, T., and Chen, B. Ivy: A read/write peer-to-peer file system. usenix.org, .

[8] Plank, J. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. Software Practice and Experience 27, 9 (1997), 995-1012.

[9] Rodrigues, R. and Liskov, B. High availability in DHTs: Erasure coding vs. replication. Lecture Notes in Computer Science 3640, 2 (2005), 226-239.

[10] Rowstron, A. and Druschel, P. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. Lecture Notes in Computer Science, November 2001 (2001), 329-350.

[11] Stoica, I., Morris, R., Karger, D., Kaashoek, M., and Balakrishnan, H. Chord: A scalable peer-to-peer lookup service for internet applications. Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, (2001), 160.

[12] Balakrishnan, H., Kaashoek, M.F., Karger, D., Morris, R., and Stoica, I. Looking up data in P2P systems. Communications of the ACM 46, 2 (2003), 43.

[13] Rabin, M. Fingerprinting by random polynomials. Technical Report, Center for Research, TR-15-18, Havard Aiken Computer Laboratory, 1981.

[14] Muthitacharoen, A., Chen, B., and Mazieres, D. A low-bandwidth network file system. Proceedings of the eighteenth ACM symposium on Operating systems principles, (2001), 174-187.

[15] W. Lin, D. Chiu, and Y. Lee, "Erasure code replication revisited," Proceedings of the Fourth International Conference on Peer-to-Peer Computing, Citeseer, 2004, 90-97.

[16] Oliveira, M., Cirne, W., Brasileiro, F., and Guerrero, D. On the impact of the data redundancy strategy on the recoverability of friend-to-friend backup systems. Proceedings of the 26th Brazilian Symposium on Computer Networks and Distributed Systems (SBRC'2008), Rio de Janeiro, Brazil, (2008).

[17] Karger, D., Lehman, E., Leighton, F., Levine, M., Lewin, D., and Panigrahy, R. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. 29th Annual ACM Symposium on Theory of Computing, (1997), 654-663.

[18] M. Oliveira. OurBackup: A P2P backup solution based on social networks, MSc Thesis, Universidade Federal de Campina Grande, Brazil, 2007.

[19] FIPS 180-1. Secure Hash Standard. U.S. Department of Commerce/NIST, National Technical Information Service, Springfield, VA, Apr. 1995.

[20] L. Cox and B. Noble, "Samsara: Honor among thieves in peer-to-peer storage," Proceedings of the nineteenth ACM symposium on Operating systems principles, 2003, p. 120–132.

[21] F. Aidouni, M. Latapy, and C. Magnien, "Ten weeks in the life of an eDonkey server," Proceedings of HotP2P'09, 2009, pp. 1-5.

[22] R. Buyya and S. Venugopal, "A Gentle Introduction to Grid Computing and Technologies," CSI Communications, vol. 29, 2005, pp. 9-19.

[23] I. Foster and C. Kesselman, The Grid 2, Second Edition: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, 2003.

[24] K. Krauter, R. Buyya, and M. Maheswaran, "A taxonomy and survey of grid resource management systems for distributed computing," Software: Practice and Experience, vol. 32, 2002, pp. 135-164.

[25] F. BERMAN, A. HEY, and G. FOX, Grid Computing: Making the Global Infrastructure a Reality, Wiley, 2003.

[26] T.a. DeFanti, I. Foster, M.E. Papka, R. Stevens, and T. Kuhfuss, "Overview of the I-Way: Wide-Area Visual Supercomputing," International Journal of High Performance Computing Applications, vol. 10, 1996, pp. 123-131.

[27] A. ABBAS, Grid Computing: Practical Guide To Technology & Applications, Hingham, Massachusetts: Charles River Media, 2003.

[28] "Open Grid Forum," 2009.

[29] M. Chetty and R. Buyya, "Weaving Computational Grids: How Analogous Are They with Electrical Grids?," Computing in Science and Engineering, vol. 4, 2002, pp. 61-71.

[30] I. Foster, "What is the Grid? A Three Point Checklist," Argonne National Laboratory & University of Chicago, 2002.

[31] Assad, Rodrigo; Katter, Tarciana; Ferraz Felipe; Meira, Silvio; Security Quality Assurance on Web Application. ICSEA 2011

[32] TALUKDER, Asoke K.; CHAITANYA, Manish. ARCHITECTING SECURE SOFTWARE SYSTEMS. Auerbach Publications, 2008.ISBN-13: 978-1-4200-8784-0

[33] OWASP, 2008, OWASP TESTING GUIDE 2008 V3.0. Disponível em: http://www.owasp.org/index.php/Category:OWASP_Testing_Project , ultimo acesso em: 06/05/2011

[34] SCHUMACHER, Markus et all. SECURITY PATTERNS - INTEGRATING SECURITY AND SYSTEMS ENGINEERING. New York: John Wiley & Sons, 2006. ISBN-13 978-0-470-85884-4

[35] Peter Mell, Karen Scarfone, Sasha Romanosky. The Common Vulnerability Scoring System (CVSS). NIST, 2007. Disponível em: http://www.first.org/cvss/ , ultimo acesso em: 06/12/2008

[36] PHILLIPS, Joseph. PMP – PROJECT MANAGEMENT PROFESSIONAL – GUIA DE ESTUDO. Rio de Janeiro: Editora Campus, 2004. ISBN:85-352-1410-0

[37] As 10 vulnerabilidades de segurança mais críticas em aplicações WEB. Disponivel em: http://www.owasp.org/images/4/42/OWASP_TOP_10_2007_PT-BR.pdf . Ultimo acesso em: 17/08/2008

[38] WIEGERS, Karl E. MORE ABOUT SOFTWARE REQUIREMENTS: THORNY ISSUES AND PRACTICAL ADVICE. Redmond: Microsoft Press, 2006. ISBN:0735622671

[39] GHOSH, Anup K., MCGRAW, Gary, 1998, An Approach for Certifying Security in Software Components, published 21st National Information Systems Security Conference, National Institute of Standards and Technology (NIST). Disponível em: http://csrc.nist.gov/nissc/1998/proceedings/paperA4.pdf. Ultimo acesso em: 22/10/2008