ETAPAS DO DESENVOLVIMENTO DE SISTEMAS EM UML
MODELO DE PROCESSO DE SW

Artefatos de software

a) a informação susceptível à reutilização inclui a análise de requisitos, especificações do sistema, estruturas de desenho, e qualquer informação que seja necessária ao processo de desenvolvimento.
b) Estes produtos do desenvolvimento são chamados *artefatos de software*.
c) Um artefato é um sub-produto do desenvolvimento de software.
d) Podem ser manuais, arquivos executáveis, módulos etc.

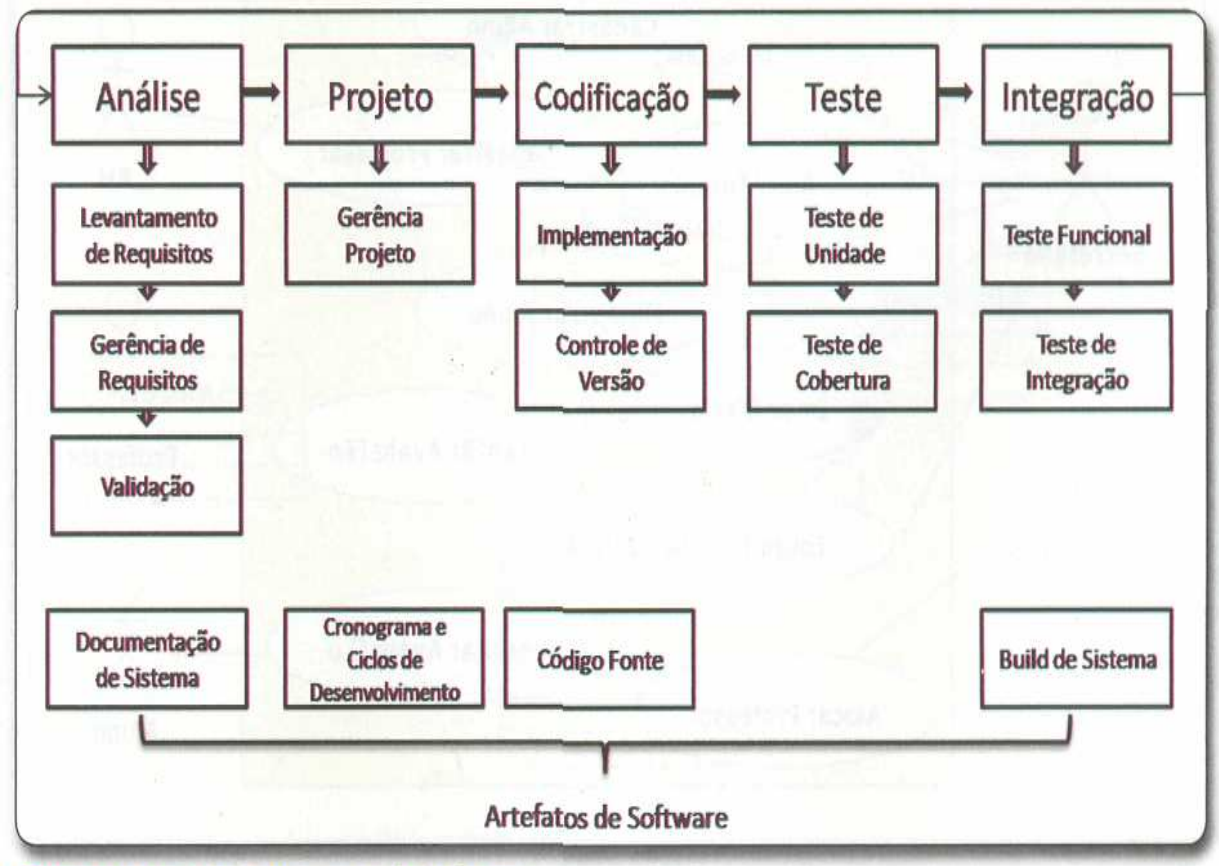

Figura 1. Modelo de Processo de Software

A figura 1 busca exemplificar de forma macro um processo de desenvolvimento de SW moderno e o POO.

FASE	DESCRIÇÃO
ANALISE:	✓ Inicialmente existe um processo de análise de SW onde os engenheiros de SW fazem o levantamento de requisitos da aplicação, fazem uso de técnicas de gerência de requisitos para organizar esses requisitos e, com o material coletado é feita então a validação destas informações, com uma participação efetiva do usuário. ✓ O principal **artefato** de SW gerado nesta etapa é a **documentação do sistema**.
PROJETO:	A próxima etapa é o projeto, onde o gerente de projetos irá propor um **cronograma e os ciclos de desenvolvimento (artefato** gerado neste etapa) de acordo com a documentação de sistema gerada na etapa anterior.

CODIFICAÇÃO:	Após isso, é feita a codificação, onde é realizada a implementação dos requisitos. Nesta etapa é de extrema importância a utilização de alguma ferramenta de controle de versão, como o CVS (Concurrent Version System) ou SVN (Subversion). O **artefato** de sw gerado nesta etapa é o **código-fonte**, que na próxima etapa é testado, utilizando diferentes técnicas de teste, como os de unidade, cobertura e funcional.
TESTE INTEGRAÇÃO:	onde o **artefato** de sw gerado nesta atividade é integrado em uma **nova versão do sistema**. Importante destacar que essas atividades ocorrem dentro de um ciclo de desenvolvimento incremental e evolutivo, onde essas atividades são continuamente repetidas para cada ciclo de desenvolvimento da aplicação. Desta forma, por exemplo, pode-se imaginar que a atividade de testes acontece ao longo de todo o processo de desenvolvimento de sw, não apenas ao sue final.

Esse é um exemplo macro de um processo de desenvolvimento de sw. Vários outros modelos existem e com várias outras possibilidades. É importante observar que o processo de desenvolvimento de sw deve contemplar o que a equipe de desenvolvimento busca para seus produtos e, por isso, o processo de sw deve ser modelado de acordo com a especificidade da empresa, produto, entre outras características e necessidades.

1ra Fase: Análise de Requisitos

Esta fase captura as intenções e necessidades dos usuários do sistema a ser desenvolvido através do uso de funções chamadas "use-cases" [Barros, 2001]. É a descrição das necessidades ou desejos de um determinado sistema. O princípio básico da análise de requisitos é identificar e documentar o que é realmente necessário, desta forma comunicando a todos os envolvidos no projeto da forma mais clara possível, de maneira não-ambígua de modo que os riscos sejam identificados sem correr riscos de imprevistos.

Através do desenvolvimento de casos de uso(em UML chamamos de “use-cases”), as entidades externas ao sistema (em UML chamamos de "atores externos") que interagem e possuem interesse no sistema são modelados entre as funções que eles requerem, funções estas chamadas de "use-cases". Os atores externos e os "use-cases" são modelados com
relacionamentos que possuem comunicação associativa entre eles ou são desmembrados em hierarquia. Descrevemos um “use-case” através de um texto especificando os requisitos do ator externo que utilizará este “use-case”.

Um “use-case” tanto pode depender de um ou mais “use-case” como pode ter seus dependentes. Através do diagrama de “use-cases” mostraremos aos atores externos(futuros usuários) o que estes podem esperar do sistema.

A análise de requisitos também pode ser desenvolvida baseada em sistemas de negócios, e não apenas para sistemas de software

2da Fase : Análise

A fase de análise preocupa-se com as primeiras abstrações(classes e objetos) e mecanismos presentes no contexto do problema[Larman, 2000].

Descrevemos as classes nos Diagramas de Classes e também para ajudar na descrição dos “use-cases”, podendo estas estar ligados uma nas outras através de relacionamentos.

Nesta fase modelamos somente as classes que pertencem ao domínio principal do problema, ou seja, classes técnicas mais detalhadas não estarão neste diagrama.

3ra Fase: Projeto

Projeto cria uma representação do domínio de problema do mundo real e leva-a a um domínio de solução que é o software [Pressman, 1995].

Nesta fase partimos para as soluções técnicas, através dos resultados obtidos na fase da análise. Serão adicionadas novas classes para oferecer uma infra-estrutura técnica tais como: interface do usuário e periféricos, banco de dados, interação com outros sistemas, e outras mais. É feita uma junção das classes da fase da análise com as classes técnicas da nova infra-estrutura, podendo assim alterar tanto o domínio principal do problema quanto a infra-estrutura.

Com a elaboração do projeto obtemos detalhadamente as especificações para dar inicio a fase de programação.

4ta Fase: Programação

Para que uma fase de programação possa ter um bom desempenho, necessitamos de um projeto bem elaborado, consequentemente converteremos as classes da fase do projeto para o código da linguagem orientada a objetos escolhida. Se o desenho foi elaborado corretamente e com detalhes suficientes, a tarefa de codificação é facilitada [Furlan, 1998].A complexidade dessa conversão vai depender da capacidade da linguagem escolhida, no entanto esta pode tornar-se fácil ou difícil de se realizar.

Em UML durante a elaboração dos modelos de análise e projeto, devemos evitar traduzi-los em códigos. Cabendo esta tarefa à fase de programação.

5ta Fase: Testes

Na fase de teste executamos um programa com a intenção de descobrir um erro[Pressman, 1995]. Testamos cada rotina ou processo detalhadamente, bem como a integração de todos os processos e a aceitação.

As rotinas devem ser testadas de duas formas, uma pelos programadores, pois estes tem um conhecimento das classes e grupo de classes que envolvem esta rotina, sabendo desta forma onde estão os pontos mais críticos que podem causar falhas. A outra forma de testes de uma rotina é feita por outro usuário, pois este irá fazer um teste do seu modo, tendo a visão da rotina como um todo.

Os testes de integração são feitos usando as classes e seus componentes de integração para verificar se estas classes estão realmente colaborando uma com as outras conforme especificado nos modelos. Nos testes de aceitação é verificado se o sistema está de acordo com o especificado no diagramas de “use-cases”.

Por fim, o sistema será testado pelo usuário final e este verificará se os resultados apresentados estão realmente de acordo com suas intenções expressas no início do projeto.

1) 1ra Fase: Análise de requisitos

Esta fase é a base do projeto, deverá ser bem elaborada para um bom desenvolvimento das demais fases. Serão abordados itens necessários para a concepção do projeto, definição de equipe envolvida e também alguns itens técnicos como: atores, casos de uso, diagrama de casos de uso e análise de riscos.

1.1) Reunião/Entrevista

Nas reuniões, das quais as pessoas convocadas a participar, não existe nenhuma hierarquia. Todos os participantes, sejam eles analistas, gerentes, diretores ou escriturários, devem se posicionar livremente, pois se foram convidados a participar é porque têm alguma contribuição a dar com o seu conhecimento específico no assunto objeto da reunião. Em uma situação normal, a hierarquia funciona como um agente inibidor e a opinião da pessoa de nível mais alto prevaleceria sobre as demais, sem nenhuma garantia de que fosse a mais indicada.

Para dar início a comunicação entre os participantes, Gause e Weinberg sugerem que o analista comece fazendo perguntas de livre contexto, concentrando-se no cliente, nas metas globais e nos benefícios.

a) Por exemplo, o analista poderia perguntar :

1) Quem está por trás do pedido deste trabalho?
2) Quem usará a solução?
3) Qual o benefício econômico de uma solução bem-sucedida?
4) Há outra fonte para a solução exigida?

b) Em seguida, o analista pode fazer perguntas que o ajudarão na compreensão do problema e ter uma idéia da solução:

5) Como você caracteriza um "bom" resultado(saída) que seria gerado por uma solução bem-sucedida?
6) Qual o(s) problema(s) essa solução resolverá?
7) Você poderia mostrar-me(ou descrever-me) o ambiente que a solução será utilizada?
8) Existem questões de desempenho ou restrições especiais que afetarão a maneira pela qual a solução é abordada?

c) Por fim, o analista fará perguntas decisivas, que concentram-se na efetividade da reunião:

9) Você é a pessoa certa para responder a essas perguntas? Suas respostas são "oficiais"?
10) Minhas perguntas são pertinentes ao problema que você tem?
11) Estou fazendo perguntas demais?
12) Há mais alguém que possa fornecer informações adicionais?
13) Existe algo mais que eu possa perguntar-lhe?

Uma abordagem à coleta de exigências orientadas a equipes aplicada durante as primeiras etapas de análise e especificação é a FAST(Facilitaded Application Specification Techniques). Essa abordagem estimula a criação de uma equipe conjunta de clientes e desenvolvedores que trabalhem juntos para identificar o problema, propor elementos de solução, negociar diferentes abordagens e especificar um conjunto preliminar de requisitos de solução. Hoje, a FAST é usada predominantemente pela comunidade de sistemas de informação, mas a técnica oferece potencial para uma melhor comunicação em aplicações de todos os tipos.

Outra abordagem especializada de entrevista que pode ser usada é o JAD(Joint Application Development). Consiste numa rápida entrevista e um processo acelerado de coleta de dados em que todos os principais usuários e o pessoal da análise de sistemas agrupam-se em uma única e intensiva reunião(que pode prolongar-se de um dia a uma semana) para documentar os requisitos do usuário. A reunião costuma ser supervisionada por um especialista treinado que atua como mediador para encorajar melhores comunicações entre os analistas de sistemas e os usuários.

Esses métodos citados possuem suas diferenças, mas todos tem alguns princípios básicos :

a) Um encontro é levado a efeito num local neutro e participam desenvolvedores e clientes;
b) Usam-se regras para preparação e participação;
c) Uma agenda formal para captar os pontos importantes, agindo de forma livre o suficiente para encorajar o fluxo de idéias e sugestões;
d) Um moderador (Cliente, Desenvolvedor ou alguém de fora) para controlar a reunião;
e) Utilização de um mecanismo de definição que pode ser uma planilha, um cavalete, adesivos de parede ou cartazes;
f) Identificação do problema, propor elementos de solução, negociar diferentes abordagens e especificar um conjunto preliminar de requisitos de solução num clima que facilite a realização da atividade.

Preenchimento da ata da reunião

Nesta primeira reunião aparecerão muitas idéias sobre como se deve comportar o sistema ou o que ele deve atingir. Todas as idéias devem ser incluídas na ata pois serão a partir delas que será dado o start no projeto.

A ata da reunião deverá ser compostas de alguns elementos essenciais .

Definições de responsabilidades
Aqui definiremos a responsabilidade de cada um dos envolvidos no projeto. Coordenador da equipe: Responsável pelo andamento do Projeto.

Resumo do Projeto
Um texto geral sobre o sistema que será desenvolvido. Neste texto será descrito as principais características e funcionalidades do sistemas.
Também será descrita a abrangência de todo o Projeto, com quem interage e por quem é interagido(isto pode ser representado por diagramas de casos de uso) .

Discriminar alguns requisitos mínimos para um bom funcionamento do projeto também faz parte deste resumo, tais como equipamentos, fontes de dados, ambiente de trabalho, etc.
Cria-se um arquivo com o nome do sistema cujo o resumo faz referência.

Atores
Atores são usuários e/ou outros meios externos que desenvolvem algum papel em relação ao sistema. Os meios externos são hardwares e/ou softwares que, assim como os usuários, geram informações para o sistema ou necessitam de informações geradas a partir do sistema. Existem atores que podem desempenhar mais de um papel no sistema, quando se pensar em atores é sempre bom pensar neles como papéis em vez de pensar como pessoas, cargos, máquinas. No sistema podem ter usuários com diferentes permissões, para isto é necessário criar um ator para cada diferente tipo de permissões. Os atores são quem desempenham os casos de uso, um mesmo
ator pode estar em um ou mais casos de uso.

Cada ator deve possuir um nome cujo terá relação direta com a sua função, possuirá uma descrição que definirá o que ele faz e com quem ele interage.

Casos de Uso
Antes da definição de casos de uso, devemos ter definidos os atores e ter a idéia da funcionalidade do sistema, a partir disto, começaremos a definir os casos de uso.

Casos de uso especificam o comportamento do sistema ou parte(s) dele e descrevem a funcionalidade do sistema desempenhada pelos atores.
Você pode imaginar um caso de uso como um conjunto de cenários, onde cada cenário é uma seqüência de passos a qual descreve uma interação entre um usuário e o sistema. Os casos de uso são representados em forma de elipse.
Não deve-se detalhar muito um determinado caso de uso, o seu detalhamento vai depender do risco que o mesmo corre, quanto maior o risco, mais detalhes terá o caso de uso.

Ao definir os casos de uso a serem desenvolvidos, trate apenas dos casos de usos mais críticos para o sistema, os casos de uso que são tarefas rotineiras não precisam ser desenvolvidos. Dessa forma sua documentação não se tornará monótona. Numericamente falando, em modo geral trate apenas de 10 à 20(%) dos casos de uso mais críticos de seu sistema, estes números citados é claro que podem variar dependendo do sistema a ser desenvolvido.

Os nomes dos casos de usos devemos sempre usar verbos, pois assim facilitará no entendimento dos mesmos. Devemos possuir uma lista com todos os nomes dos casos de usos para facilitar na identificação dos mesmos.
Preencher todos os requisitos de um caso de uso é de extrema importância.

Escopo
Uma visão gráfica do sistema, mostrando a idéia geral do sistema como ele irá interagir com o mundo externo. Esta visão é transformação de funcionalidades, interação e abrangência descritos no resumo do projeto para uma representação gráfica. Deve-se usar os atores e casos de uso.

Diagrama de caso de uso

O diagrama de caso de uso é um ponto importante de comunicação entre a equipe de desenvolvimento e o usuário, portanto deverá ser de fácil entendimento. Sua definição deve deixar bem transparente a idéia do sistema.
Nos diagramas de casos de uso ilustramos conjuntos dos casos de uso do sistema, os atores e a relação entre os atores e os casos de uso. Devemos definir o nome do diagrama de caso de uso sempre relacionado-o com o seu propósito.

Então relacionamos os casos de uso de uma determinada parte ou de todo o sistema e os relacionamos com seus respectivos atores. Deve-se documentar cada diagrama desenvolvido
observando alguns itens importantes como: nome do diagrama, descrição, atores e casos de uso relacionados.

Análise de riscos

Em todo projeto surgem várias incertezas, devemos considerar algumas mais críticas, estas deverão ser tratadas com mais importância, pois
poderão comprometer todo o desenvolvimento do sistema. Dentre elas podemos citar as mais críticas e que geralmente ocorrem na maioria dos projetos:

a) • Está realmente claro a importância deste projeto?
b) • Baseado nos casos de uso, tem-se condições de implementar as funções dentro prazo previsto?
c) • Analisar os problemas técnicos que poderão ocorrer ao longo do projeto.
d) • Caso encontre problemas, estes serão resolvidos de forma que não altere o cronograma do projeto?
e) • Os requisitos foram bem definidos de modo que se ocorrerem eventuais mudanças, estas não gerarão alterações consideráveis no tempo e nem nos custos de desenvolvimento?
f) • Empenho dos envolvidos no projeto, todos que se comprometeram realmente irão participar?

A análise de riscos é uma tarefa de grande importância no gerenciamento de um projeto de software, embora, em muitos casos, esta atividade nem seja considerada.

O seu objetivo é determinar um conjunto de passos a serem seguidos para determinar os riscos envolvidos no projeto: identificação, avaliação, classificação, definição de estratégias para administrar os riscos, resolução dos riscos, etc...

Cronograma

Os cronogramas são metas que devem ser atingidas, definindo prazos nos quais os envolvidos deverão empenhar-se na execução de suas atividades para o cumprimento dos mesmos.
Deverá ser criada uma representação (gráfico ou tabela) que exprima o tempo dedicado a cada tarefa, determinando assim o prazo final do projeto.

Aprovação

Cria-se um formulário onde responsáveis deverão aprovar a fase.
.

2) 2da FASE : Análise

Esta fase preocupa-se com as primeiras abstrações e mecanismos que estarão presentes no domínio do problema. Traremos dos diagramas de classes em conjunto com diagramas de objetos, os diagramas de interação os dividiremos em seqüência e colaboração, trataremos também dos diagramas de estado e atividade.

2.1) Diagrama de Classes

Nos diagramas de classes modelamos a visão estática do sistema, o conjunto de classes, interfaces, colaborações e seus relacionamentos. Nestes relacionamentos dar-se-á mais atenção aos quatro principais tipos que são:

Generalização/especificação, Agregação, Associação e Dependência.
Em sistema de grande complexidade podemos dividir o diagrama de classes em diagramas menores, logicamente observando sempre a integração de todas as partes.

Deve-se considerar alguns procedimentos no desenvolvimento de diagramas de classes:

a) Praticamente, ter em mãos todas as informações sobre um modelo de objetos é pouco provável. Deve-se ter conhecimento dos comportamentos iniciais destes objetos, defini-los, classificá-los e identificar os relacionamentos entre eles. Devemos nos direcionar mais as abstrações comportamentais, reduzindo o número de objetos.

b) Não devemos nos preocupar muito em usar toda a teoria envolvida nos diagramas de classes, mas sim as mais simples: classes, associações, atributos e generalização usando logicamente outras abstrações quando forem necessárias.

c) Ter conhecimento e compreensão da idéia do projeto é fundamental.

d) Cria-se modelos apenas das áreas chaves do sistema.

e) Atribuir-lhe um nome que indique sua finalidade.

f) •Na definição das classes podemos usar para definir os nomes substantivos, para os métodos usamos verbos e para os atributos usamos propriedades da classe.

A atenção em todos os detalhes desenvolvidos neste diagrama será de grande influência na construção de um bom sistema.

2.2) Diagrama de Objetos

Os diagramas de objetos são utilizados para visualizar, especificar, construir e documentar a estrutura de um conjunto de objetos, onde cada um está num estado específico e em um determinado relacionamento com demais objetos.
Os diagramas de objetos ajudam principalmente na modelagem de estruturas complexas. Quando construímos diagramas de classes, de componentes ou de implementação o que captamos realmente é um conjunto de abstrações que nos interessa como um conjunto e, deste ponto de vista,
mostra-se sua semântica e seus relacionamentos com outras abstrações existentes neste conjunto. Se imaginarmos um determinado momento num sistema modelado, encontraremos um conjunto de objetos, cada um em um estado específico e em um determinado relacionamento com os demais
objetos.

Um diagrama de objetos mostra um único conjunto de objetos relacionados uns com os outros em um momento específico.

Segundo Booch, citamos algumas considerações para desenvolvimento de diagramas de objetos [1995]:

a) Identifique alguma função ou comportamento de parte do sistema cuja modelagem você está fazendo e que é resultante da interação de uma sociedade de classes, interfaces e outros itens.

b) Para cada função ou comportamento, identifique as classes, interfaces e outros elementos e seus relacionamentos entre si que participem nessa colaboração.

c) Considere apenas um único cenário capaz de percorrer tudo isto, congele o cenário em determinado momento e represente cada objeto participante.

d) Exponha o estado e os valores dos atributos de cada um desses objetos, conforme seja necessário, para compreensão do cenário.

e) De forma semelhante, exponha os vínculos existentes entre esses objetos, representando instâncias de associações entre eles.

2.3) Diagramas de Interação

O diagrama de interação enfatiza a interação de objetos, uma especificação comportamental que inclui uma seqüência de trocas de mensagens entre objetos dentro de um contexto a fim de realizar um determinado propósito.

Devemos utilizar diagramas de interação quando necessitamos ver o comportamento de vários objetos dentro de um único caso de uso, a partir de mensagens que são trocadas entre eles. Estes diagramas modelam os **aspectos dinâmicos** do sistema.

Os diagramas de interação podem ser apresentados de duas formas:

a) Diagrama de Seqüência e
b) Diagrama de Colaboração.

2.3.a) Diagrama de Seqüência

O diagrama de seqüência expõe o aspecto do modelo que enfatiza o comportamento dos objetos em um sistema, incluindo suas operações, interações, colaborações e histórias de estado em seqüência temporal de mensagem e representação explícita de ativação de operações [Furlan, 1998].

Em um diagrama de seqüência, um objeto é representado por um retângulo no topo de uma linha vertical tracejada projetada para baixo. Esta linha representa o ciclo de vida de um objeto durante uma interação. As mensagens são representadas por linhas horizontais entre as linhas verticais de dois objetos, e a ordem de como as mensagens acontecem também é mostrada de cima para baixo.
O diagrama de seqüência mostra uma seqüência explícita de mensagens e devemos usá-lo para especificações de tempo real e para cenários complexos.

2.3.b) Diagrama de Colaboração

Um diagrama de colaboração é um contexto, um gráfico de objetos e vínculos com fluxos de mensagens anexados [Furlan, 1998].
Num diagrama de colaboração, os objetos são desenhados como ícones, e as setas indicam as mensagens enviadas entre objetos para realizar um caso de uso.

O diagrama de colaboração auxilia no entendimento de um comportamento de um conjunto de objetos que trocam mensagens entre si para realizar um propósito em uma interação global, através deste diagrama podemos ver somente os objetos e as mensagens envolvidas. Uma colaboração é anexada a um tipo, uma operação ou um caso de uso para descrever seus efeitos externos.

Em diagramas complexos devemos separá-los em diagramas menores para facilitar seu entendimento. Utilizamos a descrição do caso de uso como ponto de partida para a confecção do diagrama.
Como podemos observar os diagramas de seqüência e colaboração expressam informações semelhantes, porém de forma diferente. Quando tivermos que decidir entre qual diagrama devemos adotar para estudar uma interação, podemos escolher primeiramente o diagrama de colaboração quando o objeto e seus vínculos facilitam a compreensão da interação, e somente escolher o diagrama de seqüência se a seqüência precisar ser evidenciada.

2.4) Diagrama de Estado

O diagrama de estado exibe os eventos e os estados interessantes de um objeto e o comportamento de um objeto em resposta a um evento. O diagrama de estado implica na importância da ordem de execução dos eventos. Este diagrama faz a modelagem dos aspectos **dinâmicos de sistemas**.

Para melhor definir estes diagramas, deve-se desenvolver de forma individualizada um diagrama para cada classe existente, desta forma detalhando todos os estados da classe.

Um estado é representado por um retângulo com cantos arredondados e pode ter de um à três compartimentos que são: compartimento do nome do estado, compartimento de variáveis e compartimento de atividade interna.
Deve-se considerar alguns procedimentos no desenvolvimento de diagramas de estados:

a) • Atribuir-lhe um nome capaz de comunicar seu propósito
b) • Comece modelando os estados estáveis do objeto e, em seguida faça a modelagem das transições legais de um estado para outro.

c) • No gráfico deste diagrama, procure organizar seus componentes de forma que reduza o cruzamento de ligações de componentes.

2.5) Diagrama de Atividade

Podemos fazer uma analogia com um **gráfico de fluxo(fluxograma**), pois os diagramas de atividades controlam o fluxo de controle de uma atividade para outra. Focalizando as atividades que ocorrem entre objetos, modelando aspectos dinâmicos do sistema.

Nos Diagramas de Atividades modelamos a ordem pela qual as coisas devem ser feitas, em processos seqüências ou paralelos, o que os diferencia de um fluxograma que são limitados a processos seqüênciais.
Usamos os Diagrama de Atividades para diferentes propósitos como:

a) • Análise de Caso de Uso e compreensão do fluxo de trabalho entre vários casos de uso;
b) • Capturar o funcionamento interno de um objeto;
c) • Entender as ações quando executamos uma operação;
d) • Mostrar o funcionamento de um processo de negócio em termos de atores, fluxos de trabalho, organizações e objetos;
e) • Mostrar como uma instância de caso de uso pode ser realizada em termos de ação e mudanças de estado de objetos;
f) • Mostrar como um conjunto de ações relacionadas pode ser executado e como afetará objetos ao seu redor.

3) Projeto

Baseado na análise, trataremos nesta fase de soluções técnicas. Para essas soluções serão usados os diagramas de implementação, os quais estão divididos em diagramas de componentes e implantação.

3.1) Diagrama de implementação

Os diagramas de implementação modelam os aspectos físicos.
Apresentam aspectos de implementação, inclusive da estrutura de código fonte e de implementação de *runtime.*

Estão divididos em

a) diagramas de Componentes e
b) diagramas de implantação.

3.1.a) Diagrama de componente

O diagrama de implantação mostra a configuração dos nós de processamento em tempo de execução e os componentes que nele existem.
Os diagramas de componentes mostram a organização e as dependências existentes entre um conjunto de componentes, transformam seu projeto lógico em bits. Modelam os itens físicos que residem em um nó como **executáveis, bibliotecas, tabelas, arquivos e documentos**.

Devemos atribuir-lhe um nome que demonstre seu propósito, seu conteúdo costuma conter: componentes, interfaces, relacionamentos de dependência, generalização, associação e realização. Podem conter notas e restrições.

São usados para a modelagem da **visão estática** de implementação de um sistema. A reunião das partes modeladas ajudarão na formação do sistema executável. Usamos os Diagramas de Componentes nas seguintes formas:

a) ***Para modelagem de código-fonte***:

- Identificar um conjunto de arquivos de código-fonte de interesse e transformá-lo em componentes;
  - Em sistemas grandes, devemos usar pacotes para mostrar grupos de arquivos de código-fonte;
  - No diagrama de componentes a exposição da versão do arquivo de código-fonte ajuda no entendimento das dependências de compilação;

b)***Para modelagem de versões executáveis:***

- Identificar o conjunto de componentes, isso envolverá alguns ou todos que vivem em um único nó ou a distribuição desse conjunto de componentes por todos os nós existentes no sistema;
- Considere o estereótipo de cada componente desse conjunto;
- Para cada componente existente no conjunto, faça seus relacionamentos.

c) ***Para modelagem de bancos de dados físicos****:*

- Identificar as classes existentes no modelo que representam o esquema de seu banco de dados lógico;
- Faça o mapeamento de suas classes em tabelas;
- Estereótipe suas tabelas para um diagrama de componentes;
- Transforme seu projeto lógico em físico

d) ***Para modelagem de sistemas adaptáveis:***

- Identificar a distribuição física dos componentes que poderão mover de um nó para outro. Especifique a localização de uma instância do componente, atribua a ela o valor *location*.
- Para modelagem das causas de mudanças de um componente use um diagrama de interação para auxiliá-lo.

Para uma melhor compreensão dos diagramas de componentes podemos usar notas e cores como indicações visuais com a finalidade de chamar a atenção a pontos importantes do diagrama. E também devemos usar elementos estereotipados com cuidado, escolhendo um conjunto pequeno de ícones e utilizando-o de maneira consistente.

3.1.b) Diagrama de implantação

Diagrama de Implantação é usada para mostrar a organização do hardware e a ligação do software aos dispositivos físicos. Este diagrama denota vários dispositivos de hardware e interfaces físicas determinadas por seus estereótipos, como processador, impressora, memória, disco; suficientes para que o engenheiro de software especifique a plataforma em que o sistema é executado.

O diagrama de implantação modela a visão estática da implantação de um sistema entre seus nós físicos e seus relacionamentos e para especificar seus detalhes referente a construção.

Usamos os Diagramas de Implantação em uma das três formas :

a) *Para modelar Sistemas Embutidos :*

- Identificar os dispositivos e os nós que são únicos para o sistema;
- Crie indicações visuais para dispositivos de hardware e outros tipos dispositivos;
- Modele os relacionamentos entre os dispositivos;
- Se necessário crie diagramas de implantação mais detalhado em dispositivos inteligentes.

b) *Para modelar sistema Cliente/Servidor :*

- Identificar os nós que representem os processadores do cliente e do servidor do sistema;
- Junte os dispositivos importantes para o comportamento do sistema.
- Crie indicações visuais para processadores e dispositivos usando estereótipos
- Modele a topologia dos nós, especificando seus relacionamentos.

c) *Para modelar sistema totalmente distribuído :*

- Identificar os dispositivos e processadores para sistemas cliente/servidor mais simples;
- Caso precise analisar o desempenho da rede do sistema ou modificações na rede, modele esses dispositivos em um nível de detalhe que previna essas avaliações;
- Analise os agrupamentos lógicos de nós, especificando-os em pacotes;
- Identifique a topologia do seu sistema;
- Se for preciso focalizar a dinâmica do sistema, introduza diagramas de casos de uso para especificar o comportamento em foco e crie diagrama de interação para expandir esses casos de uso.

Os diagramas de implantação não são muito utilizados em sistema que residem em uma máquina e que interagem somente com dispositivos padrão(teclado, monitor, mouse, modem pessoal) dessa máquina, que já são gerenciados pelo S.O. host.

Em contrapartida se estiver desenvolvendo software que interaja com dispositivos que o sistema operacional host tipicamente não gerencia, ou por processadores distribuídos fisicamente no mesmo host, o diagrama de implantação ajudará a analisar o mapeamento de software para hardware de seu sistema.

Para uma melhor compreensão dos diagramas de implantação devemos atribuir-lhe um nome que identifique seu propósito, podemos usar notas e cores como indicações visuais com a finalidade de chamar a atenção a pontos importantes do diagrama. E também devemos usar elementos estereotipados com cuidado, escolhendo um conjunto pequeno de ícones e utilizando-o de maneira consistente.

4) Implementação

A fase da implementação é, na prática, a construção física do sistema proposto [Silva,1998]. É onde todos os modelos desenvolvidos no projeto são traduzidos para código que a máquina possa reconhecer.
É muito importante que as fases anteriores já estejam concluídas, pois a codificação é uma conseqüência natural do projeto.

4.1) Tradução

O processo de tradução acontece quando a compilador aceita o código-fonte como entrada e gera um código-objeto como saída.
Para que o código-fonte não seja gerado de maneira diferente do projeto, deve-se ter todos os detalhes do projeto bem elaborados e estudados.

4.2) Escolha da Linguagem

Linguagem de codificação complexa ou pouco conhecida pode levar a códigos mal elaborado e de difícil legibilidade. Devemos nos concentrar em escolher linguagens de codificação que tenha suporte ao projeto.

A escolha de uma linguagem de programação para um projeto específico deve-se considerar as suas facilidades e aonde ela irá ajudar o melhor desenvolvimento do projeto.
PRESSMAN sugere alguns critérios a serem avaliados durante a escolha de uma linguagem de programação [1995]:

a) • A área de aplicação geral;
b) • A complexidade computacional e algorítmica;
c) • O ambiente em que o software será executado;
d) • Considerações de desempenho;
e) • A complexidade da estrutura de dados;
f) • O conhecimento da equipe de desenvolvimento do software;
g) • A disponibilidade de um bom compilador.

Geralmente considera-se a área de aplicação geral como um dos itens mais estudados na escolha da linguagem.
O lançamento de novas linguagens ou versões é muito constante, portanto muitas vezes necessitamos escolher linguagens do conhecimento da equipe, que geralmente não são os últimos lançamentos. Porém novas linguagens devem ser cuidadosamente estudadas e a mudança da linguagem atual para nova tem que ocorrer.
Na escolha de uma linguagem orientada a objetos deve-se considerar suporte para definições de classes, herança, encapsulação e transmissão de mensagens ou ainda características adicionais como herança múltipla e polimorfismo.

As novas gerações de ferramentas CASE possuem uma variedade de linguagens, com isto elas podem gerar códigos-fonte baseadas no modelo desenvolvido. Porém devem ser bastante estudadas pois uma falha no modelo pode comprometer todo o produto final.

4.3) Documentação do Código

Um código-fonte bem estruturado e documentado será de grande importância para futuras manutenções no sistema.
A estruturação e documentação começa com a escolha de nomes identificadores(variáveis e rótulos), prosseguindo com a colocação e composição do comentário e por fim com a organização visual do código.
Na escolha de nomes de identificadores devemos considerar alguns itens a fim de facilitar a interpretação :

a) • Expressar diretamente o seu propósito.
b) • Identificação do tipo e de sua abrangência(global, local) no sistema. Podem ser feitas algumas definições de nomes antes do início da programação.
c) • Não usar nomes muito extensos, mas que expresse o seu propósito. Neste caso deverá prevalecer o bom censo e experiência por parte do programador.

Na colocação e composição de comentários, tomaremos alguns cuidados para que estes realmente ajudem outros leitores de código na compreensão do código e até mesmo para que o desenvolvedor possa localizar-se com mais facilidade e rapidez em futuras manutenções. Isto é de extrema importância no desenvolvimento de sistemas e/ou funções complexas.

Segundo PRESSMAN, comentários em forma de prólogo são bastante usados, eles aparecem no início de cada módulo, e seguem o formato abaixo [1995]:

1) • Uma declaração de propósitos que indique a função do módulo.
2) • Uma descrição da interface("seqüência de chamada", descrição de todos os argumentos e uma lista de todos os módulos subordinados).
3) • Uma discussão de dados pertinente, tais como variáveis importantes e suas restrições e limitações de uso.
4) • Um histórico de desenvolvimento(autor do módulo, auditor e data, datas e descrição das modificações).

Para uma melhor documentação em módulos e/ou funções muito complexas devemos utilizar também comentários descritivos embutidos no decorrer do código-fonte. Este tipo de comentário será de grande utilidade para documentarmos linhas ou conjunto de linhas específicas em determinado local dentro de um módulo e/ou função. Este comentário deverá sempre anteceder o
bloco de comandos que se deseja documentar, faça apenas a identificação da funcionalidade de tal linha ou conjunto de linhas, não é necessário documentar outros dados como datas, autores, etc.
Ao escrevermos código-fontes e também seus comentários devemos sempre nos preocupar com sua estética, procuramos sempre elaborar códigos identados e seus comentários devem sempre ter uma boa visualização.

Algumas dicas de PRESSMAN para simplificar a codificação [1995]:
• Evitar o uso de testes condicionais complicados;
• Evitar os testes em condições negativas;
• Evitar um intenso alinhamento de laços ou condições;
• Usar parênteses para esclarecer expressões lógicas ou aritméticas;
• Usar símbolos de espaçamento e/ou legibilidade para esclarecer o conteúdo da instrução;
• Usar somente características com padrão ANSI;
• Auto questionar-se: Será que entenderia este código se não fosse eu o autor deste?

5) Testes

A atividade de teste visa executar um programa ou parte dele com a intenção de encontrar erros.
Um teste bem executado é aquele que encontra um maior número de erros com o menor número de tentativas, em um tempo e esforço mínimo.

Certamente que os projetistas codificam o software para que não haja erros. Uma estratégia de teste de software deve estar dividida nas seguintes fases : planejamento de teste, projeto de casos de teste, execução de teste e a resultante coleta e avaliação de dados [Pressman, 1995].
Para tornar esta estratégia mais flexível e customizada sem perder sua validade e regidez, objetivando a redução de custos e tempo adotamos a seguinte estratégia : planejamento, execução e avaliação dos resultados.

5.1) Planejamento de Testes

Antes do início dos teste devemos definir a equipe responsável pelos testes. Devemos formar um grupo de testes independentes (GTI), envolvendo também os desenvolvedores principalmente na etapa de teste de unidade da fase de execução que será abordado a seguir.

Devemos tomar como base para testes os casos de usos desenvolvido na fase de análise de requisitos, verificando se seu propósito está contemplado, observando seu fluxo normal e quando este fluxo não é seguido se os fluxos alternativos estão atendendo tal situação.

5.2) Execução de Testes

Nesta etapa são realizados os testes na integra, abordaremos os

a) testes de Unidade,
b) Integração,
c) Validação e
d) Sistema.

Durante a execução dos testes, deve-se fazer um relatório de acompanhamento para registro de todos as ocorrências encontradas.

5.2.a) Teste de Unidade

Nestes testes cada unidade de software(classe, procedimento, função, arquivo, etc) é testado individualmente garantindo que ele funcione adequadamente. Aqui a participação dos desenvolvedores também é recomendada, pois estes tem um melhor conhecimento interno de cada unidade.
Alguns aspectos a serem testados: são a interface, as estruturas de dados, as condições limite, os caminhos independentes e os caminhos de tratamento de erros.

a) A ***interface***, em busca de erros de passagem de dados para dentro e para fora do módulo. Deve ser realizada em primeiro lugar, durante a realização deste teste os seguintes aspectos devem ser observados :

1) Consistência entre o número e o tipo de parâmetros de entrada com o número e tipo de argumentos;
2) Consistência entre o número e o tipo de argumentos transmitidos aos módulos chamados com os parâmetros;
3) Consistência de variáveis globais ao longo dos módulos;
4) Existência de referências a parâmetros não associados ao ponto de entrada atual;
5) Modificação de argumentos de entrada/saída.

b) As ***estruturas de dados*** locais, para se prover a garantia de que todos os dados armazenados localmente mantêm sua integridade ao longo da execução do módulo. Neste teste devemos nos preocupar em :

1) Digitação inconsistente ou imprópria;
2) Iniciação ou valores default errôneos;
3) Tipos de dados errados;
4) Underflow e Overflow.

c) As ***condições limite***, que permitem verificar que o módulo executa respeitando os valores máximos e mínimos estabelecidos para seu processamento.

d) Os ***caminhos independentes*** (ou caminhos básicos) da estrutura de controle são analisados para ter garantia de que todas as instruções do módulo foram executadas pelo menos uma vez. Os erros mais comuns são :

1) Precedência aritmética incorreta;
2) Operações em modo misto;
3) Inicialização incorreta;
4) Erros de precisão;
5) Representação simbólica de uma expressão incorreta;
6) Laços mal definidos;
7) Comparação de diferentes tipos de dados;
8) Operadores lógicos utilizados incorretamente.

e) Os ***caminhos de tratamento de erros*** (se existirem), serão testados para observar a robustez do módulo. Devemos observar os seguintes itens :

1) Descrição incompreensível do erro;
2) O erro exibido não é o erro encontrado;
3) Intervenção da condição de erro no sistema antes do seu tratamento;
4) O processamento das condições de exceções é incorreto;
5) A descrição do erro não ajuda na localização da causa do erro.

5.3) Teste de Integração

O teste de integração é uma técnica sistemática para a construção da estrutura de programa, realizando-se em paralelo, testes com o objetivo de descobrir erros associados a interfaces. Seu objetivo é montar, a partir dos módulos testados ao nível de unidade construir a estrutura de programa que foi determinado pelo projeto.

Este teste está divido em incremental e não incremental. Incremental aborda o teste de integração top-down o programa é construído e testado em pequenos seguimentos, onde os erros são mais fáceis de ser isolados e corrigidos. O teste não incremental aborda o teste de integração bottom-up o programa é testado como um todo, um conjunto de erros é encontrado e sua correção é difícil porque o isolamento das causas é complicado pela vasta amplitude do programa inteiro [Pressman,1995].

1. **Integração top-down :** Os módulos são integrados movimentando-se de cima para baixo através da hierarquia de controle, iniciado-se do módulo principal. Os módulos subordinados ao módulo principal de uma maneira depth-first(pela profundidade) ou breadth-first(pela largura).

2. **Integração bottom-up :** Inicia-se a construção e testes com módulos localizados nos níveis mais baixos da estrutura do programa dirigindo-se ao módulo principal.

5.4) Teste de Validação

Ao final do teste de integração, o software é finalmente estruturado na forma de um pacote ou sistema. O teste de validação é a validação e verificação de que o software como um todo cumpre os requisitos do sistema.
Na fase de análise de requisitos são definidos critérios que servirão de validação para aprovação ou não do software. O teste de validação aborda as seguintes estratégias : revisão de configuração e teste alfa e beta.

1. **Revisão de configuração:** é onde são verificados os elementos de configuração do software se estão adequadamente desenvolvidos e detalhados para futuras manutenções.
2. **Teste alfa**: é feito por um cliente nas instalações do desenvolvedor. O software é usado num ambiente controlado com o desenvolvedor "olhando sobre os ombros" do usuário e registrando erros e problemas de uso.
3. **Teste beta**: é feito nas instalações do cliente pelo usuário final do sistema. O desenvolvedor não está presente. O cliente registra os problemas(reais ou imagináveis) que são encontrados e relata-os ao desenvolvedor em intervalos regulares.

5.5) Teste de Sistemas

O software é apenas um elemento de um sistema baseado em computador mais amplo. Este teste envolve um grande número de diferentes testes, cujo propósito é pôr completamente à prova o sistema baseado em computador.
Entre os tipos de testes de sistemas destacamos: De recuperação, de segurança, de estresse e de desempenho.

a) • O ***teste de recuperação*** é um teste de sistema que força o software a falhar de diversas maneiras e verifica se a recuperação é executada.

b) • O ***teste de segurança*** verifica se todos os mecanismos de proteção embutidos no sistema o protegerão.

c) • O ***teste de estresse*** força o sistema a usar recursos em quantidade, freqüência ou volumes anormais.

d) • O ***teste de desempenho*** é feito para testar o desempenho do software em tempo real.

5.5) Avaliação dos Resultados
Completadas a fase de execução dos testes, passaremos para a avaliação dos mesmos. Isto será feito analisando os relatórios elaborados durante a fase anterior.

Esta fase é de grande valia para vários aspectos do sistema. Nesta poderemos ter uma idéia por completo de todo o desenvolvimento do projeto.
Podemos apontar falhas em determinadas fases do projeto baseando-se nesta avaliação, bem como certificar-se da qualidade e/ou bom desempenho do projeto.

Ocorrendo pequenos erros ou observações que não interfiram no objetivo do projeto, estes deverão ser tratados diretamente com o responsável pela fase onde ocorreu o problema.

Quando na avaliação ocorrem problemas do tipo que tenha que ser refeito alguma unidade por completo, isto geralmente estará comprometendo os objetivos do projeto e, consequentemente deverá ser de conhecimento de toda a equipe envolvida no projeto.

Após feitas as correções dos problemas encontrados, estando de acordo com as especificações definidas na análise de requisitos e aceitas pelo cliente, o software estará pronto para uso.

VISÕES DE SISTEMAS
Visão USE-CASE
Visão Lógica
Visão de Componentes
Visão de concorrência
Visão de Organização

VISÕES DE SISTEMAS
Conceito
O desenvolvimento de um sistema complexo não é uma tarefa fácil. O ideal seria que o sistema inteiro pudesse ser descrito em um único gráfico e que este representasse por completo as reais intenções do sistema sem ambigüidades, sendo facilmente interpretável [Barros, 2001].
Um sistema é composto por diversos aspectos: funcional (que é sua estrutura estática e suas interações dinâmicas), não funcional (requisitos de tempo, confiabilidade, desenvolvimento, etc.) e aspectos organizacionais (organização do trabalho, mapeamento dos módulos de código, etc.).

Descrevemos um sistema em várias visões, cada visão mostra alguma particularidade do sistema e estão em alguns de diagramas.
Algumas visões estão interligadas através de diagramas, podendo uma visão fazer parte de uma ou mais visões. Os diagramas que compõem asvisões contém os modelos de elementos do sistema. A Figura 6.1 apresenta as cinco visões de sistemas.

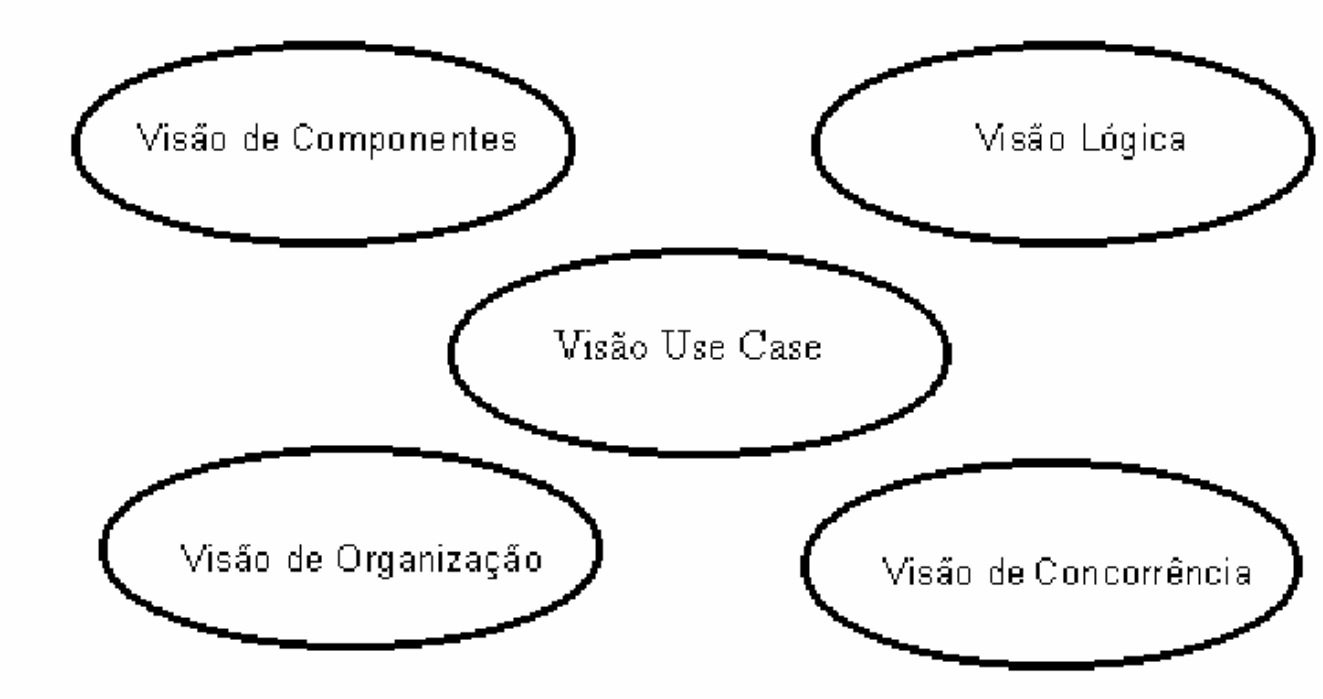

Figura: visões de sistemas.

Visão USE-CASE
Descreve a funcionalidade do sistema desempenhada pelos atores externos do sistema (usuários). A visão use-case é central, já que seu conteúdo é base do desenvolvimento das outras visões do sistema. Essa visão é montada sobre os diagramas de use-case e eventualmente diagramas de atividade.

Visão Lógica
Descreve como a funcionalidade do sistema será implementada. É feita principalmente pelos analistas e desenvolvedores. Em contraste com a visão use-case, a visão lógica observa e estuda o sistema internamente. Ela descreve e especifica a estrutura estática do sistema (classes, objetos, e relacionamentos) e as colaborações dinâmicas quando os objetos enviarem mensagens uns para os outros para realizarem as funções do sistema.
Propriedades como persistência e concorrência são definidas nesta fase, bem como as interfaces e as estruturas de classes. A estrutura estática é descrita pelos diagramas de classes e objetos. O modelamento dinâmico é descrito pelos diagramas de estado, seqüência, colaboração e atividade.

Visão de Componentes
É uma descrição da implementação dos módulos e suas dependências. É principalmente executado por desenvolvedores, e consiste nos componentes dos diagramas.

Visão de concorrência
Trata a divisão do sistema em processos e processadores. Este aspecto, que é uma propriedade não funcional do sistema, permite uma melhor utilização do ambiente onde o sistema se encontrará, se o mesmo possui execuções paralelas, e se existe dentro do sistema um gerenciamento de eventos assíncronos. Uma vez dividido o sistema em linhas de execução de processos concorrentes (threads), esta visão de concorrência deverá mostrar como se dá a comunicação e a concorrência destas threads. A visão de concorrência é suportada pelos diagramas dinâmicos, que são os diagramas de estado, seqüência, colaboração e atividade, e pelos diagramas de implementação, que são os diagramas de componente e execução.

Visão de Organização
Finalmente, a visão de organização mostra a organização física do sistema, os computadores, os periféricos e como eles se conectam entre si.

Esta visão será executada pelos desenvolvedores, integradores e testadores, e será representada pelo diagrama de execução.