Construindo Softwares com Qualidade e Rapidez Usando ICONIX

José Anízio Maia

Este artigo aborda as principais fases de construção de softwares de forma rápida e com qualidade através do processo de desenvolvimento de software ***ICONIX***.

Introdução

Fazer *software* hoje em dia não é fácil, cada vez mais os programas estão se tornando complexos, os problemas fáceis já foram resolvidos e é possível encontrá-los em vários portais gratuitamente, porem, os problemas que hoje tentamos resolver, através da implementação de um *software*, se tornaram complexos e sofisticados, por exemplo, antigamente a micro empresa do seu Joaquim tinha um sistema de folha de pagamento, um sistema de controle de estoque, um sistema de contas a pagar etc. Hoje, por questões de integridade dos dados, eficiência, rapidez nas informações etc., os sistemas estão integrados e conseqüentemente sua complexidade aumentou. Com isso, podemos perceber o quanto é importante planejar, projetar e avaliar o *software* que será construído. Isso pode ser feito através de um processo de desenvolvimento de *software*.

A quem se destina este Artigo?

Este artigo está destinado as pessoas que tem algum interesse em Engenharia de Software e desejam saber mais sobre os processos de desenvolvimento de sistemas. Somente é recomendável ter noção de UML. Isso facilitará bastante para assimilar a proposta do ICONIX.

O que é um processo de desenvolvimento de software?

São etapas cuidadosamente planejadas onde o sucesso de cada etapa é primordial para produtos com qualidade, baixo custo e rapidez na construção, ou seja, pode possibilitar bom resultado final no produto de *software*.

Nem sempre foi assim, com o surgimento do *software* como forma de agilizar o trabalho, controlar informações, armazenar dados, reduzir custos, etc. para as empresas e órgãos governamentais no mundo inteiro, criou-se um novo mercado de trabalho e oportunidade de negócio. Mas, o processo de desenvolvimento de *software* era verdadeiramente um caos, as organizações criavam *softwares* de qualquer jeito e os clientes aceitavam sem reclamar de nada, mesmo porque não sabiam como se concebia *softwares*. Agora a realidade é diferente, existe concorrência, altos custos e o cliente esta super exigente.

No inicio da década de 90, verificou-se o nascimento de uma série de metodologias de desenvolvimento de *software*. Isto devido ao envolvimento de vários engenheiros de *software*, que conseguiram visualizar as vantagens de usarem essas metodologias quando os compararam com os diversos processos existentes até então. A ordem agora é controlar, ter o domínio desde os primeiros rabiscos da idéia até o produto final, poder prever todos os

possíveis problemas que possam impedir o sucesso do projeto. Mas como conseguir isso? Claro que não é fácil, mas também não é impossível, basta ter força de vontade e selecionar o processo adequado de acordo com o tipo de projeto e segui-lo a risca.

Observações

Existem vários processos de desenvolvimento de *software*, cada um com suas características, mas, cabe ao gerente de projeto escolhê-lo. Essa escolha deve ser feita de acordo com as características do projeto e o que se espera para o resultado final.

Por exemplo: um sistema em que a vida de pessoas esteja em risco (sistema de trafego aéreo) não pode permitir falhas, nesse caso, ter o controle detalhado de todas as fases deve ser uma obsessão pela equipe, um *software* de grande porte, um *software* de médio ou pequeno porte, tempo, custo também são outros pontos que devem ser levados em consideração ao selecionar um processo de desenvolvimento de *software*, ou, simplesmente você pode usar um processo que a empresa onde você trabalha o obrigue a usar.

O que é ICONIX

ICONIX considerado uma metodologia pura, prática e simples, mas também poderosa e com um componente de análise e representação dos problemas sólido e eficaz, por isso, a metodologia ICONIX é caracterizada como um Processo de Desenvolvimento de Software desenvolvido pela ICONIX Software Engineering.

O ICONIX é um processo não tão burocrático como o RUP, ou seja, não gera tanta documentação. E apesar de ser um processo simples como o XP, não deixa a desejar na Análise de Design, e se destaca com um poderoso processo de análise de *software*. Isso poderá ser visto posteriormente nos tópicos seguintes.

Este processo também faz uso da linguagem de modelagem UML e possui uma característica exclusiva chamada “Rastreabilidade dos Requisitos” (*Traceability* of *Requirements*), mais precisamente, ele nos permite “obrigatoriamente” através de seus mecanismos, verificar em todas as fazes, se os requisitos estão sendo atendidos.

A abordagem do ICONIX é flexível e aberta, isto é, se for necessário usar outro recurso da UML para complementar os recursos usados nas fases do ICONIX, não há problema algum.

O ICONIX é composto pelas seguintes principais fazes das quais entraremos em detalhes nos próximos tópicos:

- Modelo de Domínio
- Modelo de Caso de Uso
- Análise Robusta
- Diagrama de Seqüência
- Diagrama de Classe

Este processo é dividido em dois grandes setores, que podem ser desenvolvidos em paralelo e de modo recursivo. Os modelos são: Modelo Estático e o Modelo Dinâmico como mostra a Figura 1:

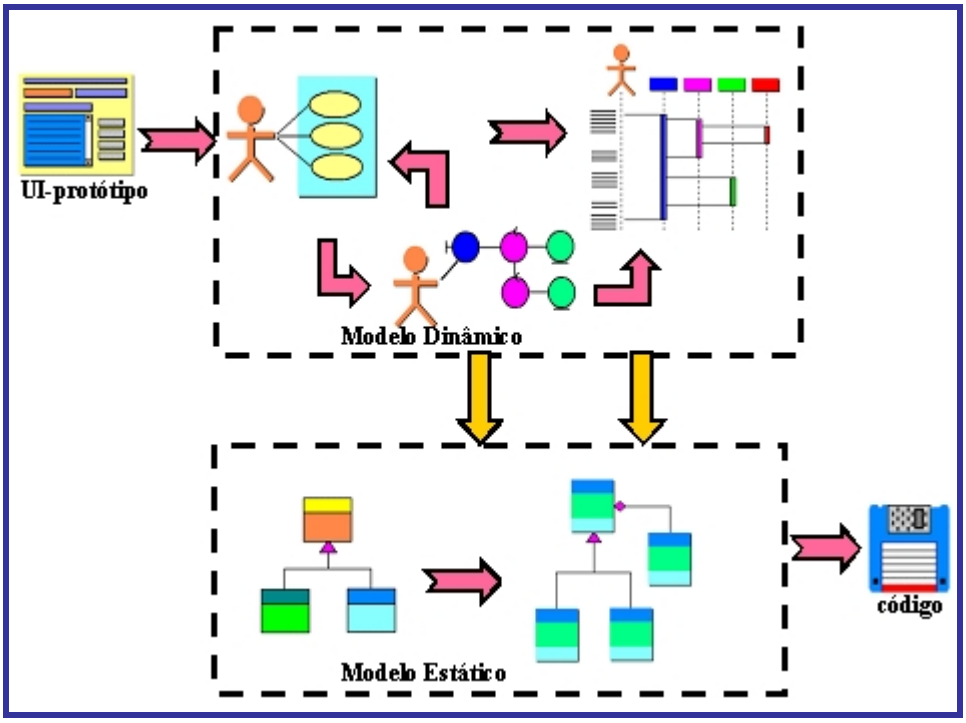

Figura 1: visão macro do ICONIX

O Modelo Estático é formado pelos Diagramas de Domínio e o Diagrama de Classe que pertencem à divisão estática por modelarem o funcionamento do sistema sem nenhum dinamismo e interação do usuário, ou seja, é o posto do Modelo Dinâmico que sempre mostra o usuário interagindo com o sistema através de ações onde o sistema apresenta alguma resposta ao usuário em tempo de execução. O modelo estático deverá ser refinado incrementalmente durante as iterações sucessivas do modelo dinâmico. Não exige marcos formais de projeto para o refinamento, isso é feito de forma natural durante o projeto.

A proposta do ICONIX permite um alto grau de Rastreabilidade. Em cada fase ao longo do caminho é necessário rever os requisitos em algum momento. Não existe um ponto em que o processo permite distanciar dos requisitos do usuário. Rastreabilidade também significa encontrar novos objetos em cada fase durante o projeto.

O processo ICONIX trabalha a partir de um protótipo de interface onde se desenvolvem os diagramas de caso de uso baseado nos requisitos do usuário. Com base nos diagramas de caso de uso se faz à análise robusta para cada caso de uso e com os resultados obtidos, é possível desenvolver o diagrama de seqüência e posteriormente, povoar o modelo de domínio já revisado com métodos e atributos descobertos.

O uso da interface/protótipo como ponto de partida

O modelo de caso de uso é criado a partir de um protótipo de interface validado pelo usuário de acordo com os requisitos do sistema. O objetivo da interface não é codificação do sistema e pode ser feito usando qualquer ferramenta ou tecnologia mesmo que esteja longe da realidade do sistema proposto pelo cliente. No ICONIX, os sistemas são concebidos a partir da visão do usuário para com o sistema, por isso o uso de um protótipo construído "na presença do cliente" de acordo com suas necessidades. O uso desta interface possibilita ter maior garantias que os requisitos estão sendo atendidos e uma visão maior do sistema e conseqüentemente maior controle.

O Modelo de Domínio

O Modelo de Domínio é uma parte essencial do processo de ICONIX. Ele constrói uma porção estática inicial de um modelo que é essencial para dirigir a fase de *design* a partir dos casos de uso.

Basicamente o modelo de domínio consiste em descobrir objetos de um problema do mudo real. Transferir os problemas do mundo real para um diagrama nem sempre é fácil, mas, com bastante treino e experiência, nossa capacidade de abstração tende a melhorar.

Para realizar o modelo de domínio, é preciso tentar descobrir o maior número possível de classes existentes no problema para qual se pretende desenvolver o *software*. Claro que modelo de domínio não retratará o cenário completo adequando para o problema que se pretende resolver, algumas classes serão excluídas e outras serão encontradas ou modificadas etc. Isso faz parte do processo de desenvolvimento de *softwares*.

Construindo o Modelo de Domínio:

A primeira coisa que temos a fazer ao construir um modelo estático do sistema é achar classes apropriadas que com precisão representam as reais abstrações do domínio de problema. Devemos executar bem está atividade, pois assim teremos uma base sólida para construir o sistema.

1. Destacar todos os substantivos e frases que contenham substantivos, analisar e refinar esta lista gradualmente, pois substantivos e frases de substantivo se tornarão objetos e atributos, enquanto que verbos e frases de verbo se tornarão associações. Casos possessivos como ("seu", "nosso" e "seus") tendem a indicar ser substantivos, mas, devem ser atributos no lugar de objetos.

2. Revisar a lista de classes em busca de ambigüidades, irrelevância, classes incorretas ou vagas e itens desnecessários, algumas vazes mesmo que sejam substantivos é possível que estejam fora do escopo do problema.

3. Fazer generalização quando necessário, devemos tomar cuidado com generalização, verificar se realmente existe uma relação de generalização e garantir que esteja de

acordo com o mundo real. Verifique se for o caso de existência de agregação ou composição e aplique no modelo.

4. Finalmente revise este diagrama, com atenção em associações e generalizações, pois, este diagrama será a base para o diagrama de classe.

Ex: suponhamos que queremos construir um *software* bancário onde um dos requisitos seria que os clientes pudessem abrir uma conta, os clientes podem ser pessoas físicas ou jurídicas. De posse destas informações, já poderíamos começar um modelo de domínio simples, então o diagrama poderia ficar assim: (ver Figura 2).

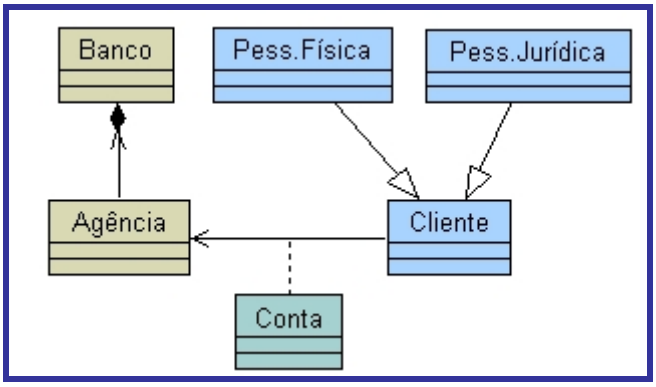

Figura 2: exemplo de modelo de domínio

Erros mais comuns na construção do Modelo de Domínio

O principal objetivo do Modelo de Domínio e descobrir objetos e associações, portanto, detalhes de cada objeto e sua multiplicidade deve ser omitida. Geralmente alguns erros são cometidos durante a construção do Modelo de Domínio:

1. Não se apressar em usar multiplicidades e associações, pois, ainda não temos informações suficientes e provavelmente isso causará paralisia de análise, ou seja, será perda de tempo tentando descobrir detalhes e se descobri-los provavelmente serão alterados.
2. Não fazer análise exaustiva analisando substantivos e verbos, pois é melhor ter um nível de abstração correto ao invés de um nível de detalhe alto e não garantido. Essa é uma boa técnica, mas devemos ter cuidado com exageros e não levar tão a sério.
3. Não tentar descobrir operações para as classes do modelo de domínio, pois, não temos informações suficientes para isso, é melhor esperar pela análise robusta e o diagrama de seqüência.
4. Não tentar construir o Modelo de Domínio pensando em reutilização de código, prefira primeiramente atender os requisitos do usuário.
5. Não perder tempo em dúvidas quanto agregação e composição para as partes da associação, prefira agregação que é mais simples, agregação contra composição são detalhes posteriores.
6. Não construir o modelo pensando em alguma estratégia de implementação ou tecnologia que representem algum compromisso tecnológico específico, exemplo:

servidores, bancos relacionais, assuntos de implementação ou de licença etc.
7. Não usar nomes difíceis para as classes. Procurar usar nomes óbvios e legíveis, isso será bom para toda equipe de projeto.
8. Se for modelar usando um legado, cuidado ao trazes as classes e seus relacionamentos para o modelo estático, apesar do banco de dados ser uma grande fonte de classes, existem muitos atributos e operações que não irão fazer parte do modelo.
9. Não usar padrões de projeto prematuramente, devemos nos preocupar em atender os requisitos do usuário, o uso de padrões deve ficar mais claro durante a análise robusta, portanto, agora não é hora para isso.

O Modelo de Caso de Uso

Este modelo é usado para representar as exigências do usuário seja um sistema novo (partindo do nada) ou baseado em um sistema já existente. Ele deve detalhar de forma clara e legível, todos os cenários que os usuários executarão para realizar alguma tarefa.

Construindo o Modelo de Caso de Uso:

1. Identificar tantos quantos casos de uso pudermos, descrever cada caso de uso detalhadamente com clareza e sem ambigüidades, manter um ciclo de refinamento de texto contínuo. É comum durante a descrição encontrar novos casos de uso.

2. Se já existe um sistema legado, o manual do usuário é uma grande fonte de casos de uso.

3. Para cada caso de uso, refinar o texto tendo certeza que orações estão claras e bem descritas, o formato básico do texto é substantivo-verbo-substantivo, e os potenciais atores e objetos de domínio são fáceis identificar.

4. Atualizar o modelo de domínio, pois é comum encontrarmos novos objetos e isso pode influenciar na compreensão do problema previamente modelado.

5. Durante a descrição dos casos de uso procurar definir claramente os cursos de ação básicos (realização da ação pelo usuário) e os cursos alternativos (algo que impeça o curso de ação básico).

6. Verificar se todos os casos de uso atendem as necessidades funcionais desejadas no sistema.

7. Verificar se para cada caso de uso foi descrito o curso básico de ação e o curso(s) alternativo(s).

Ex: esse é um exemplo de descrição de caso de uso e o diagrama para o exemplo proposto no modelo de domínio (Figura 3).

Curso Básico -
O usuário entra com as informações necessárias do cliente no sistema.
O sistema valida o cadastro do cliente.
Armazena os dados do cliente
Uma conta é criada em nome do cliente.
O sistema notifica o usuário sobre o cadastro.
Se o cadastro foi bem sucedido, o sistema exibe uma mensagem de confirmação.
Curso Alternativo - se o cliente já estiver cadastrado no sistema do banco, uma mensagem será exibida.
Curso Alternativo - se os dados estiverem incompletos o sistema deverá bloquear o cadastro
Exibir uma mensagem para que o usuário forneça as informações completas.

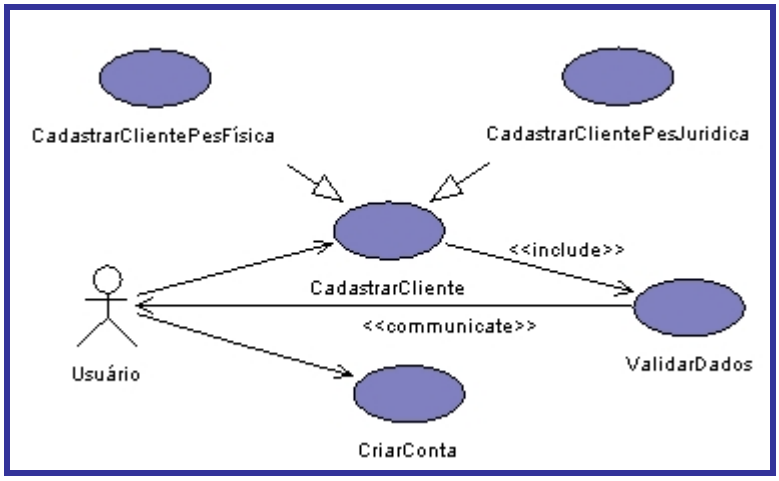

Figura 3: diagrama de caso de uso

Erros mais comuns na Descrição de Caso de Uso

O principal objetivo do Modelo de Caso de Uso e descobrir a maior quantidade possível de casos de uso e descrevê-los detalhadamente e verificar se os mesmos atendem aos requisitos do sistema. Geralmente alguns erros são cometidos durante a construção do Modelo de Caso de uso:

1. Não escrever requisitos funcionais em vez de escrever texto de argumento de uso. Os requisitos geralmente são declarados dentro de condições do que o sistema deverá fazer, enquanto que argumentos de uso descrevem uma ação do usuário e a resposta que o sistema gera. Eventualmente, o texto de descrição será usado como *run-time* comportamental e este texto servirá de apoio para o diagrama de seqüência. Nós precisamos ver como sistema executará o comportamento desejado como descrito no texto de caso de uso, para isso, precisamos diferenciar descrição de uso (comportamento) e requisitos do sistema.
2. Não descrever atributos e métodos em lugar de uso. O texto não deve incluir muitos detalhes de apresentação, mas também deve ser relativamente livre de detalhes sobre os campos de telas. Não deveriam ser nomeados métodos ou descrevê-los no texto de caso de uso porque eles representam como o sistema fará coisas, ao invés do que o sistema fará.
3. Não escrever os casos de uso muito concisamente. É preferível escrever texto para casos de uso expansivos. É preciso descrever todos os detalhes de ações de

usuário e respostas de sistema. Isso servirá para uma boa análise robusta. Também é bom lembrar que os casos de uso servirão para construir o manual de usuário. É melhor errar por excesso de detalhe na documentação de usuário.

4. Não ignorar o protótipo de interface. Na orientação a casos de uso é fundamental o uso de telas para construir o sistema a partir da visão do usuário para com o sistema. Não descrever casos de uso sem ser específicos sobre como as ações do usuário executarão em suas telas, não precisa descrever nome de campos nem aparência das telas.
5. Não evitar nomes explícitos para os objetos de interface. Objetos de interface são os objetos com que os atores irão interagir. Estes freqüentemente incluem janelas, telas, diálogos e menus. Devemos manter amplo detalhe e sendo explícito sobre navegação do usuário, é necessário nomear os objetos interface no contesto explicitamente. Também é importante fazer isto porque estaremos explorando o comportamento destes objetos durante análise robusta, e poderemos reduzir ambigüidade e confusão ao nomeá-los.
6. Não escrever os textos na voz passiva, usando uma perspectiva diferente do usuário. Um caso de uso é mais eficaz escrito da perspectiva do usuário como um jogo de frases e verbos no presente em voz ativa. A tendência dos engenheiros em usarem voz passiva é muito grande, mas casos de uso deveriam declarar as ações que o usuário executa, e as respostas do sistema para essas ações. Este tipo de texto só é efetivo quando é expresso na voz ativa.
7. Não escrever somente iterações do usuário, não ignorar as respostas do sistema para as ações. A narrativa de um caso de uso deveria ser evento - resposta, como em: "O sistema faz isto quando o usuário fizer aquilo". O caso de uso deveria capturar uma transação boa do que acontece a respeito ao que o ator está fazendo, se o sistema cria objetos novos, valida usuário, gera mensagens de erro ou tudo que for possível. Lembrando que o texto de caso de uso descreve ambos os lados do diálogo entre o usuário e o sistema.
8. Não omitir texto para cursos alternativos de ação. Cursos básicos de ação são geralmente mais fáceis identificar e descrever. Porém, isso não significa que devemos ignorar detalhes dos cursos alternativos. Longe disto. Na realidade, é muito importante quando cursos alternativos e de ação são descobertos, pois do contrário, quando forem codificar e depurar, o programador responsável por escrever o código tende a tratá-los de modo conveniente para ele. Isto não é saudável para um projeto.
9. Não devemos focalizar o texto em algo diferente de uma descrição de caso de uso. Não escrever texto com informações desnecessárias somente porque em um livro ou artigo foi publicado.
10. Não perder muito tempo pra decidir em usar estereótipos “include” e “extends”.

O Modelo de Análise Robusta

Está fase tem como objetivo, conectar a fase de análise com a fase de projeto (ver Figura 4) assegurando que a descrição dos casos de uso estão corretas, além de descobrir novos objetos através do fluxo de ação. A Análise Robusta focaliza construir um modelo analisando as narrativas de texto de caso de uso, identificando um conjunto de objetos que participarão de cada caso de uso. Estes objetos podem ser classificados em três tipos como mostra a Figura 5.

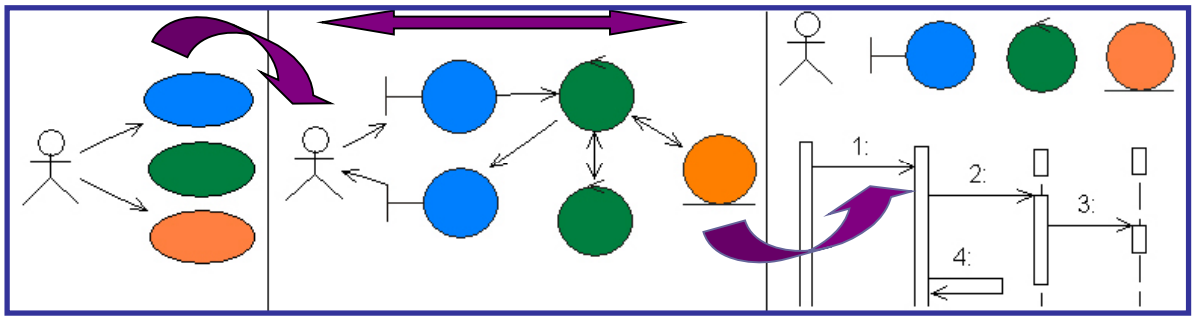

Figura 4: conecta a fase de análise com a fase de design.

Tipos de objetos da Análise Robusta:

1. Objetos Limite ou interface (*Boundary Objects*) – é usado pelos atores (por exemplo, os usuários) para se comunicarem com o sistema. A maioria dos objetos interface podem ser vistos no protótipo de interface.
2. Objetos Controladores (*Control Objects*) - são objetos que controlam a lógica de negócio, eles fazem a conexão entre os objetos interface e objetos entidade.
3. Objetos entidade (*Entite Objects*) são responsáveis para realizar algum tipo de persistência. Geralmente eles vêm do modelo de domínio.

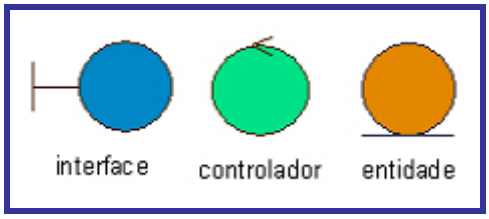

Figura 5: representação dos objetos limite/interface, controlador e entidade.

A análise robusta é um modelo simples, mas, é uma técnica extremamente útil para análise, é uma fase primordial e tem vários papéis dentro do ICONIX.

Construindo o Modelo de Análise Robusta:

1. Durante a análise robusta, será possível encontrar problemas na descrição dos casos de uso e, portanto, nós seremos obrigados a corrigi-los. Ao final desta fase, teremos o modelo de caso de uso refinado. O mesmo acontecerá para o modelo estático.

2. Devemos ter certeza que os casos de uso foram descritos corretamente e estão de acordo com as funcionalidades do sistema, caso contrário, a análise robusta nos ajudará a corrigir estes problemas.

3. Durante a análise robusta será necessário que todos os cursos básicos e alternativos tenham sido identificados, caso contrário, devemos voltar aos casos de uso de identificá-los. A análise robusta também ajudara nesse processo.

4. Se for necessário, devemos perder bastante tempo neste modelo, pois, ele servirá para construção do diagrama de seqüência. Caso este modelo não esteja de acordo com a realidade ou mal feito, não será possível prosseguir para o diagrama de seqüência.

5. Durante a construção do modelo robusto, será possível encontrar objetos novos não identificados anteriormente no modelo de domínio, e possíveis problemas de discrepância ou conflito entre objetos antes que eles causem maiores problemas.

6. A análise robusta é a ponte entre a fase de análise e a fase de projeto, portanto, ao concluir este modelo, nós estaremos partindo para os detalhes de projeto.

7. Qualquer pessoa deve ser capas de rastrear um diagrama robusto de acordo com a descrição do texto de caso de uso.

8. A maioria das pessoas não escreve casos de uso corretamente na primeira vez, então se tiver excesso ou falta de informação, o diagrama robusto irá ajudar a identificar essas falhas.

9. Refinar o modelo estático continuamente com os novos objetos encontrados e se alguns atributos importantes foram descobertos é hora de ir povoando nossas classes mais importantes com eles.

10. Se for o caso de usar algum padrão de projeto é hora de aplicar no diagrama robusto.

Regras para uso dos objetos na análise Robusta:

1. Atores somente podem se comunicar com objetos interface.
2. Objetos interface somente podem se comunicar com atores e controladores.
3. Objetos entidade somente podem se comunicar com objetos controladores.
4. Objetos controladores somente podem se comunicar com objetos entidade e interface, também com outros controladores, mas não com atores.

Ex: elaborando o modelo de análise robusta para o exemplo proposto no modelo de domínio no qual já foi feita a descrição de caso de uso e o diagrama de caso de uso. Esse diagrama poderia ser de acordo com a Figura 6:

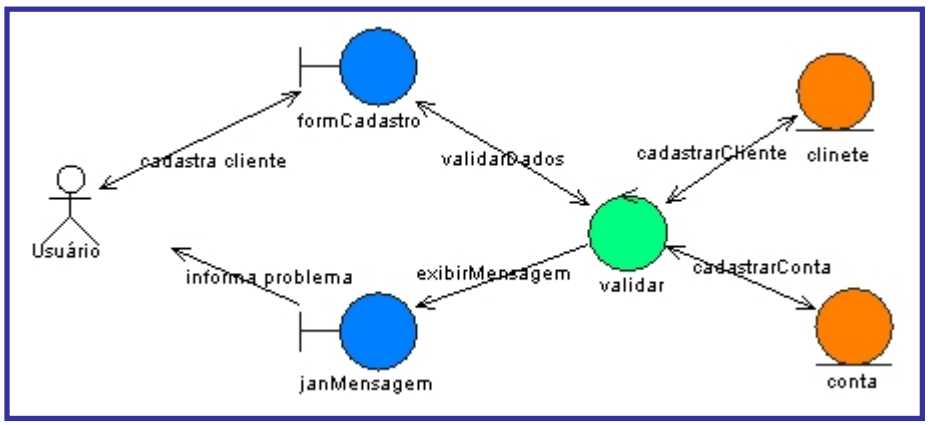

Figura 6: exemplo de diagrama robusto

Erros mais comuns na construção do Modelo de Análise Robusta

O Modelo de Análise Robusta tem vários papéis dentro do ICONIX como, por exemplo, criar um diagrama Robusto de acordo com a descrição do caso de uso, refinar o modelo estático, refinar a descrição dos casos de uso, identificar novos objetos. Geralmente alguns erros são cometidos durante a construção do Modelo Robusto:

1. Não violar as quatro regras do Diagrama de Análise Robusta listados acima. Estas regras garantem coerência com os testos de casos de uso e que os objetos não irão receber responsabilidades erradas.
2. Usar a análise robusta para ajudar a dar um formato consistente para os textos de casos de uso, melhorando a redigibilidade e a manutenibilidade.
3. Incluir cursos alternativos no diagrama de análise robusta. A maioria dos comportamentos interessante acontece nos curso alternativo, assim é bom modelar esses comportamentos. É comum encontrar cursos alternativos novos, especialmente quando os objetos controladores são explorados em verificar e validar informações.
4. Usar análise robusta para assegurar consistência entre o nome das classes e do texto de descrição dos casos de uso. Isso também será útil ao construir bons diagramas de seqüência.
5. Não colocar comportamento de classes no diagrama robusto.Não devemos nomear métodos a classes em um diagrama robusto, porque é provável que não tenha bastante informação. Devemos tomar decisões sobre comportamento seqüencial para cada fluxo.
6. Não incluir poucos controladores. É bom ter entre dois e cinco controladores em um diagrama robusto, se tiver apenas um controlador em um diagrama robusto, provavelmente não foi bem descrito o comportamento do caso de uso. Por outro lado, se tiver mais de dez controladores em um caso de uso, é bom quebrar este caso de uso para melhor manejo.
7. Não perder tempo para detalhar diagrama robusto e torná-lo perfeito, pois seu objetivo não é detalhamento de projeto e sim ter uma visão geral do problema, descobrir novos objetos, alocar atributos refinar o texto de caso de uso. Se detalharmos demasiadamente o diagrama robusto, poderemos perder seus benefícios. O detalhamento deve ficar para o diagrama de seqüência.

8. Fazer um rastro visual do diagrama robusto de acordo com o texto de caso de uso. O esquema deve estar de acordo com o texto e o texto de acordo com os requisitos do sistema. Não devemos considerar os casos de uso concluídos até que esteja finalizado a análise robusta e que ele passe no teste de rastreabilidade visual.
9. Atualizar o modelo estático. Temos que atualizar o modelo de domínio antes de considerar concluído a análise robusta e passar então para o diagrama de seqüência. Pois não podemos alocar comportamento a classes que não aparecem no modelo estático.

O Diagrama de Seqüência

O Diagrama de Seqüência tem como objetivo construir um modelo dinâmico entre o usuário e o sistema. Para tal, devemos utilizar os objetos e suas interações identificadas na análise robusta, só que agora, temos por obrigação o detalhamento de cada fluxo de ação. Teoricamente isso já é possível, pois, uma vez concluído o modelo de domínio (diagrama de classe de alto nível) e análise robusta, nós, teremos descoberto a maioria dos objetos no contexto do problema e definido alguns atributos e relações estáticas entre objetos no diagrama de classe de alto nível e algumas relações dinâmicas na análise robusta. Estas conquistas representam passos largos para um bom resultado.

Construindo o Diagrama de Seqüência:

Agora é hora de projetar como o *software* realmente irá funcionar. O modelo de interação é fase onde construímos as linhas de ação entre objetos. Agora começamos a ver como o sistema executará comportamentos úteis.

Durante a construção do modelo de interação, nós devemos alcançar as seguintes metas primárias:

1. Alocar comportamentos entre os objetos interface, controlador e entidade identificados durante a análise robusta. Isso permitirá realizar o comportamento desejados para os casos de uso. Se tiver difícil identificar os objetos e seus comportamentos, nesse caso é melhor retornar a análise robusta e ter certeza que os fluxos estão corretos.

2. Mostrar as interações detalhadas que ocorrem na linha de tempo entre os objetos associados com o texto de caso de uso. Os objetos interagem mandando mensagens (chamadas) uns aos outros. Uma mensagem estimula um objeto a executar alguma ação desejada. Para cada comportamento de um caso de uso, temos que identificar as mensagens necessárias e os métodos.

3. Finalmente devemos povoar o diagrama de classe distribuindo as operações entre os objetos. Quando se termina a análise robusta é possível ter identificado a maioria dos atributos dentro do modelo estático.

Uma vez concluído o modelo de interação, nós teremos bastantes informações e poderemos dispor de comportamento detalhado do esquema de seqüência entre objetos no contexto de seus respectivos casos de uso. Podemos finalizar procurando e alocando apropriadamente atributos e operações.

À medida que vamos progredindo no modelo de interação, devemos atualizar o modelo estático. Isso também ajuda a compreender melhor como o sistema deverá se comportar.

Dentro do ICONIX, o diagrama de seqüência representa o artefato principal do produto de *design*, devemos construir o diagrama de seqüência de acordo com o curso básico e alternativo de para cada caso de uso.

O resultado que o modelo de interação deve apresentar é a essência do modelo dinâmico no qual exibe o comportamento do sistema em tempo de execução de forma detalhada.

Um Diagrama de seqüência é composto por quatro componentes:

1. O texto que descreve os fluxos de ação para os caso de uso – o texto deve estar do lado esquerdo das linhas de ação dos objetos, é bom separar o texto com uma linha em branco entre os fluxos de ação, assim, fica fácil ver cada linha de ação e sua descrição.

2. Os objetos que interagem entre si – os objetos são extraídos da análise robusta e são formados por dois componentes: o nome do objeto (opcional) e a classe a qual ele pertence, todos ficam no topo do diagrama dentro de uma caixa da qual desce uma linha pontilhada.

3. Mensagens - mensagens são setas entre objetos. Uma seta de mensagem pode ir diretamente entre duas linhas pontilhadas, entre uma linha e um método de um retângulo, ou entre dois métodos de um retângulo.

4. Métodos ou operações – são mostrados em cima das linhas a qual pertencem os objetos.

Quatro passos para montar o Diagrama de Seqüência:

1. Copiar o texto de especificação do caso de uso no lado direito do diagrama. O texto serve como uma lembrança continua do que é necessário fazer. O resultado é que quando se está fazendo o *design*, o comportamento requerido sempre está diante de nossos olhos e se não foi definido todos os cursos alternativos pertinentes às ações do curso básico para cada caso de uso, então é melhor não continuar até que eles estejam identificados, caso contrário os diagramas não cobrirão todos os casos especiais e não será possível descobrir tudo sobre o comportamento dos casos de uso. Isto significa que não será possível descobrir todos os métodos necessários para os objetos.

2. Adicionar os objetos entidade do diagrama robusto. Cada um destes objetos é um exemplo de classe que deve estar no diagrama de classe que representa o modelo estático. Lembrando que já era para estar atualizado o diagrama de classe com os novos objetos encontrados na análise robusta, se isso não foi feito, podemos fazer agora. Estes objetos estão com a maioria dos atributos encontrados, mas é possível encontrar novos atributos perdidos durante a construção do diagrama de seqüência. Devemos ser meticulosos ao acrescentar algo no diagrama de seqüência, pois, este é o ultimo passo antes da codificação.

3. Adicionar os objetos interface do diagrama de análise robusta.

4. Alocar os métodos em suas respectivas classes adequadamente. Isso envolve converter os controladores do diagrama robusto, um de cada vez, aos conjuntos dos métodos e mensagens que envolvem o comportamento desejado. Ocasionalmente um controlador pode se tornar um objeto real de controle.

Ex: este é um exemplo de como ficaria o diagrama de seqüência pra o exemplo dos tópicos anteriores (Figura 7).

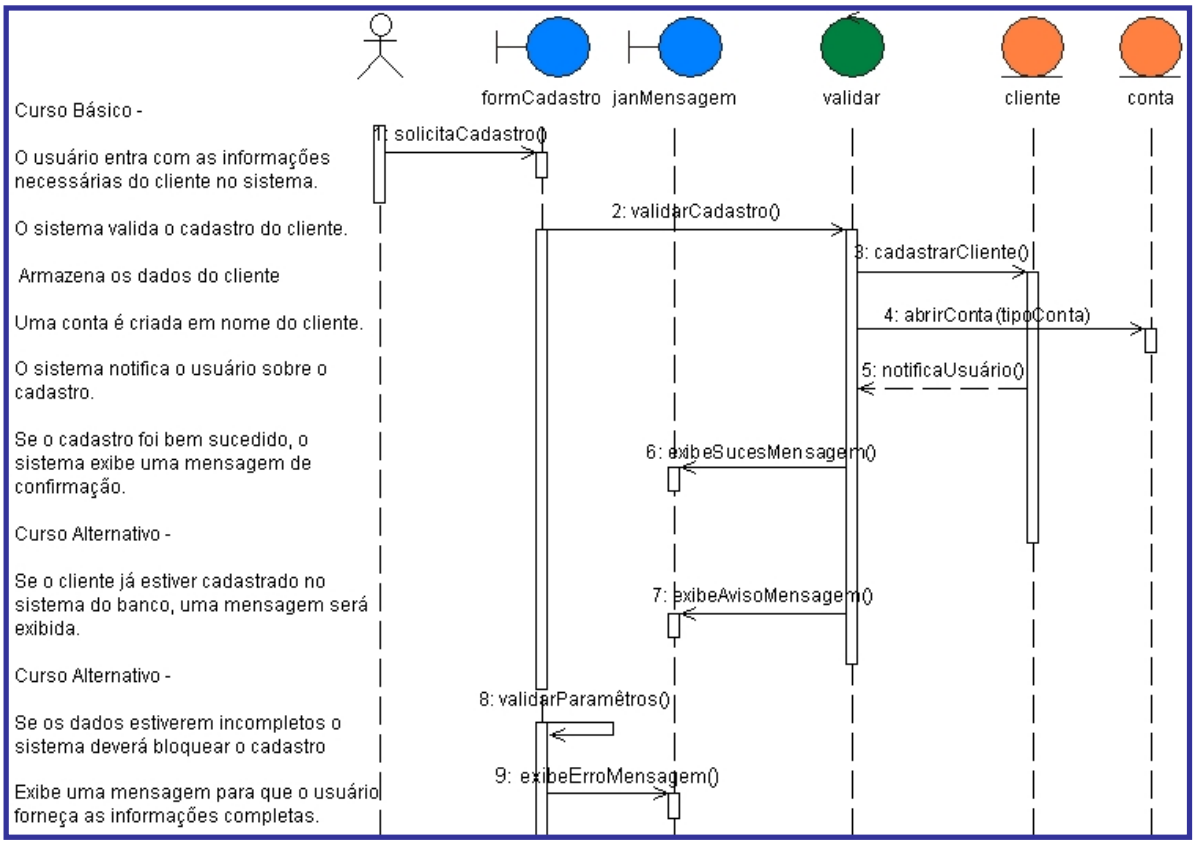

Figura 7: diagrama de seqüência

Erros mais comuns na construção do Diagrama de Seqüência

O Diagrama de Seqüência tem como papel principal mostrar o funcionamento do sistema em tempo de execução, para tal, é preciso realizar a troca de mensagens entre os objetos seguindo a descrição do caso de uso que deve estar no diagrama de seqüência. Geralmente alguns erros são cometidos durante a construção do diagrama de seqüência:

1. Não fazer um diagrama de seqüência para cada caso de uso. É preciso mostrar os objetos e a responsabilidade de cada um.
2. Não colocar o texto de caso de uso no diagrama de seqüência. O texto permite a Rastreabilidade dos requisitos do sistema, de modo que não tem como não seguir as necessidades do sistema visto que sua descrição está face a face com o projetista e cada mensagem deve corresponder a uma ação.
3. Não identificar todos os objetos necessários no diagrama robusto. Se tiver ocorrendo paralisia de análise (não está conseguindo fazer o diagrama de seqüência) provavelmente há problemas com diagrama robusto, então devemos voltar e se certificar que ele esteja seguindo corretamente a descrição de caso de uso.
4. Não prover um rastro visual entre as setas de mensagens e o texto de caso de uso. Cada seta de mensagem deve esta alinhada com a descrição de sua ação, isso torna mais fácil a compreensão para qualquer pessoa que tentar ler o diagrama e garante a rastreabilidade dos requisitos e identificar possíveis problemas.
5. Não mostrar em detalhes a realização do caso de uso. Um diagrama de seqüência com nível de abstração muito elevado pode prejudicar na codificação do sistema, o diagrama de seqüência é a ultima etapa antes da codificação, portanto devemos abusar dos detalhes.
6. Transformar o diagrama de seqüência em um fluxograma ao invés de alocar comportamentos entre objetos. O diagrama de seqüência é a fase onde se deve tomar decisões de projeto muito importantes, alocar comportamento nas classes corretamente, isso pode tornar o projeto bom ou ruim.
7. Não focalizar os métodos mais importantes (mostrem o real funcionamento do sistema) e gastar tempo em definir métodos get() e set(). Devemos explorar o comportamento dinâmico do sistema, alocar atributos é métodos as classes.
8. Não pensar cuidadosamente na origem das setas de mensagens, em outras palavras qual objeto está no controle em determinado momento. Mensagens entre objetos são operações entre classes, por isso, elas devem estar bem definidas no diagrama de seqüência. O fluxo entre as mensagens deve ser óbvio e mostrar qual objeto está no controle.
9. Não seguir os princípios básicos da orientação a objeto. Um objeto através da extensão de uma classe deve ter uma única responsabilidade. Isso quer dizer que uma classe deve ser focalizada em um comportamento relacionado a sua existência. Isso também será útil para Reusabilidade. Portanto, antes de alocar um método a um objeto, devemos nos perguntar se este comportamento esta realmente de acordo com as funções daquele objeto.
10. Atualizar o modelo estático transformando-o em diagrama de classe agrupados em pacotes de caso de uso. Nesse momento estaremos trocando o espaço do problema pelo espaço da solução do problema.

O Diagrama de Classe

O diagrama de classe é o modelo de domínio que foi atualizado ao longo das fazes do ICONIX e representa as funcionalidades do sistema de modo estático sem a interação do usuário com o sistema. Os objetos banco e agência (ver Figura 8) não foram incluídos no diagrama de seqüência e na análise robusta para não complicar os exemplos, mas somente para nível de abstração, devemos saber que uma agência deve ter um relacionamento de composição com a classe banco, ou seja, a classe agência somente existe se a classe banco existir.

Se tivéssemos que modelar o requisito usado como exemplo neste artigo em um caso real, teríamos que fazer algumas alterações em todos os diagramas que foram usados como exemplo, essas mudanças seriam em relação a detalhes que foram omitidos também para não complicar os exemplos. Por exemplo, o diagrama robusto usado como exemplo neste artigo não é uma boa prática de modelagem, nele poderíamos incluir pelo menos mais um objeto controlador e um objeto interface. Devemos observar que o sexto item dos erros mais comuns na elaboração do diagrama robusto acusa erro em caso e se usar apenas um controlador por diagrama.

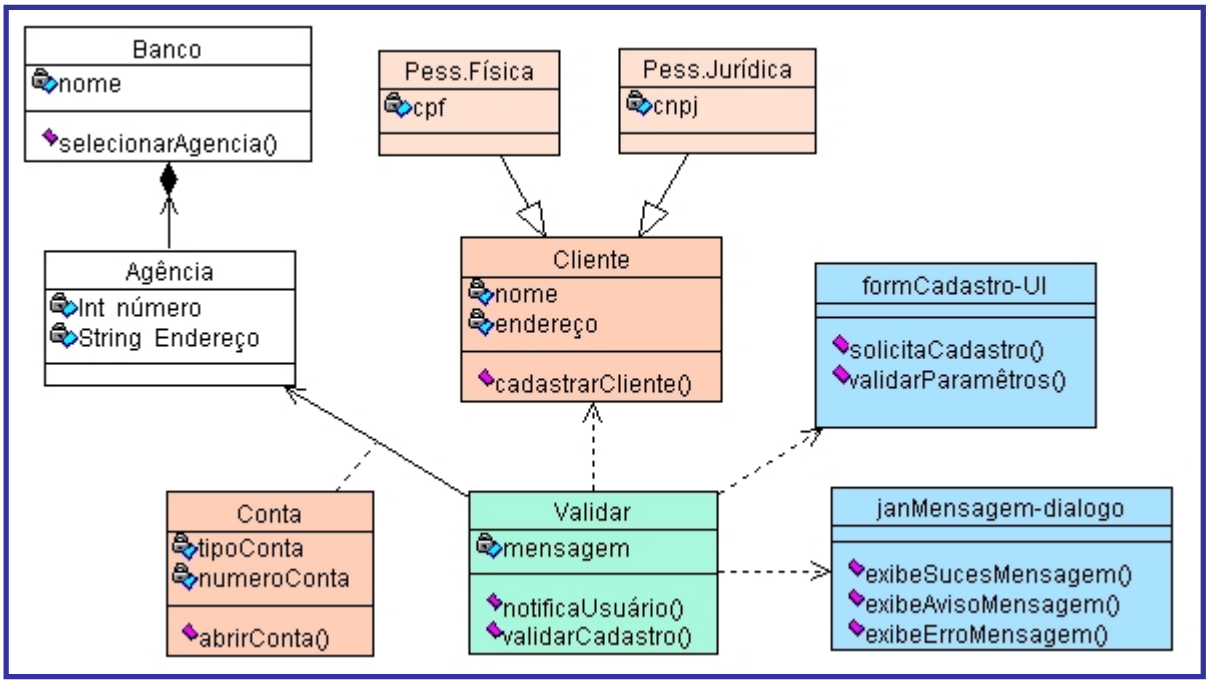

Figura 8: diagrama de classe

Codificação

O ICONIX considera a fase de codificação fora de sua área de análise, por considerar subjetiva para o desenvolvimento correto do sistema que se deseja construir. A fase de codificação na atividade de produção do sistema final é responsável pela correta tradução dos artefatos produzidos para um produto de *software* correto e estruturado. O produto deve refletir as funcionalidades requeridas identificadas durante a construção dos diagramas.

Obviamente para qualquer tarefa de codificação, será necessário especificar que após desenvolver as utilidades necessárias para estabelecer a ligação entre os diversos objetos, uma bateria de testes devera ser aplicada no sistema para verificar se o mesmo está de acordo com as operações desejadas e resultados esperados.

Conclusão

Podemos observar durantes os tópicos deste artigo, como é possível ter controle no desenvolvimento de um *software*. Ter controle total de um sistema a ser construído pode ser determinante para o resultado final de um projeto, portanto, é sempre bom que as organizações adotem algum processo de desenvolvimento de *software*.

O ICONIX é uma sugestão de processo de desenvolvimento de *software*. Este artigo não é uma referência completa, mas podemos mostrar a essência do ICONIX, suas principais características como rastreabilidade de requisitos que é um processo natural que ocorre durante todas as fases. O seu poderoso mecanismo de análise também é um bom ponto a ser elogiado e levado em consideração.

Como na vida nem tudo são flores, há também os espinhos, existe alguns problemas nessa metodologia como, por exemplo, o fato do ICONIX se afastar bastante da fase de codificação. Faz sentido a filosofia empregada por esse sistema (a codificação deve retratar o que foi produzido durante as fases de análise), mas, não devemos esquecer que um processo sozinho não faz nada, é preciso que pessoas o executem, e é bom lembrar que nenhum ser humano esta livre de falhas.

Logicamente que se todos os artefatos produzidos durante as fases que compõe o ICONIX estiverem retratando clara e detalhadamente o comportamento do sistema, não há mais nada a fazer a não ser codificar e testar. Isso levanta a hipótese que os engenheiros de sistemas que elaboraram essa metodologia estão certos de que ao final do projeto de análise, os artefatos produzidos estão retratando perfeitamente o que o sistema deverá fazer.

Autor

José Anízio Pantoja Maia (gorran4@hotmail.com / anizio_maia@yahoo.com.br) graduado em Sistemas de Informação, participou a implantação/operação do SIPAM (Sistema de Proteção da Amazônia), certificado em Java atualmente trabalha com desenvolvimento de dispositivos móveis usando a Plataforma Java Micro Edition e é um dos fundadores do JugManaus (http://www.jugmanaus.com).

Referências

http://www.iconixsw.com/Spec_Sheets/Uml2.html
http://pst.web.cern.ch/PST/HandBookWorkBook/Handbook/SoftwareEngineering/UCDOM_summary.html
http://www.dca.fee.unicamp.br/cursos/EA976/2s2002/referencia.html#iconix