Aluno: Moises R. Souza.
Curso: Engenharia em Computação
Bloco: Análise, Projeto e Construção Disciplinada de Softwares I [20E3_1]

Introdução

VenturaHR

Documento de Visão

O RH 2.0 ganhou protagonismo e está sendo promovido a área estratégica, essencial para a tomada de decisão e crescimento do negócio.

Problemas antigos como a dificuldade em medir o retorno sobre o investimento (ROI) das ações, ineficiência do recrutamento e baixo desempenho em treinamentos já têm suas soluções tecnológicas sob medida.

As HRTechs são startups que desenvolvem soluções tecnológicas para a área de Recursos Humanos (Human Resources), agregando inteligência aos processos. A missão dessas empresas é levar a inovação ao RH, mostrando que a tecnologia pode reduzir custos, aumentar a eficiência e agilizar o crescimento do setor.

Cenário Atual

VenturaSoft é uma HRTech que atua no segmento de recolocação de profissionais de TI. Devido às peculiaridades desse mercado, os requisitos para contratação têm um dinamismo vertiginoso, pois as tecnologias e as "stacks" adotadas pelas empresas estão em constante evolução.

A empresa necessita que seja construída uma solução de software, chamado de VenturaHR, que tenha abrangência de todos os fluxos operacionais da sua atividade-fim.

A VenturaSoft tem como clientes empresas que precisam fazer processos seletivos para vagas em aberto.

Solução Desejada

O software VenturaHR precisa ter como diferencial o fato de não manter um banco de Curriculum e sim um banco de Vagas X Critérios X Candidatos.

Publicação da Vaga

O fluxo de negócio é iniciado com a publicação de uma vaga de trabalho pelos responsáveis na empresa cliente. Cada empresa cliente pode ter várias contas que permitem o cadastro de vagas.

Uma vaga tem dados específicos e é descrita por uma lista de critérios de seleção que podem ser:

- Conhecimento em tecnologias especificas.
- Conhecimento em idiomas.
- Graduação, Pós-Graduação, Mestrado, Doutorado.
- Tempo de experiência em funções específicas.

Os critérios são variáveis e são informados durante a publicação da vaga. Cada vaga tem o seu conjunto de critérios exclusivo a fim de evitar que critérios semelhantes sejam reconhecidos como diferentes em função de erros de digitação / semântica.

O Perfil Mínimo Desejado - PMD do candidato é uma graduação de 1 (desejável), 2, 3, 4 e 5 (obrigatório) a ser conjugado com o peso que cada critério tem na seleção. Esse perfil indica a adequação dos critérios à vaga publicada.

O perfil da seleção é um índice calculado a partir da média ponderada dos valores definidos para cada critério. Esse índice serve como linha-base para a contratação.

Por exemplo, uma vaga para Analista de Requisitos poderia ter 4 critérios: UML, Língua Inglesa, Análise de Pontos de Função e Experiência Profissional. Observe que nenhum critério foi definido como obrigatório, aceitando candidatos que estejam próximos do que se deseja.

Dessa forma, uma publicação pode ficar assim:

Critério	Descrição	PMD*	Peso
UML	O candidato deverá conhecer os principais diagramas da UML: casos de uso, classes e sequência.	4	5
Inglês	Conversação e leitura de documentos técnicos.	4	3
Análise de Pontos de Função	Desejáveis conhecimentos de dimensionamento de sistemas.	1	1
Experiência Profissional	2 anos de experiência em levantamento de requisitos e análise.	4	2

*Perfil Mínimo Desejado

Nesse exemplo, o perfil da oportunidade é calculado pela média ponderada:

(4*5) +(4*3) + (1*1) + (4*2) / (5 + 3 + 1 + 2) = 20 + 12 + 1 + 8 / 11 = 41 / 11 = **3,72**

A tabela acima mostra como o contato da empresa que publica a oportunidade deve especificar uma vaga.

Cada vaga terá uma data/hora limite para receber as informações dos interessados.

Resposta Para a Vaga

O candidato que desejar responder a essa oportunidade deve poder pesquisar por vários critérios, mas a pesquisa pelo cargo desejado é mandatória. Ao selecionar uma vaga, uma página deverá ser carregada e o candidato deve responder através de valores 1

(nenhum/pouco), 2, 3, 4 e 5 (todo) qual é o seu conhecimento / experiência em cada critério, como por exemplo:

Critério	Descrição	Conhecimento/ Experiência
UML	O candidato deverá conhecer os principais diagramas da UML: casos de uso, classes e sequência.	5 todo
Inglês	Conversação e leitura de documentos técnicos.	3 médio
Análise de Pontos de Função	Desejáveis conhecimentos de dimensionamento de sistemas.	1 nenhum/pouco
Experiência Profissional	2 anos de experiência em levantamento de requisitos e análise.	5 todo (tem 3 anos)

O candidato que respondeu à oferta teve o seu perfil calculado da seguinte forma:

(5*5) +(3*3) + (1*1) + (5*2) / (5 + 3 + 1 + 2) = 25 + 9 + 1 + 10 / 11 = 45 / 11 = **4,09**

Regras Gerais

As vagas de emprego têm um período dentro do qual candidatos podem responder. Na data limite da vaga o sistema processará as respostas e mandará um e-mail para o contato da empresa que publicou a oportunidade. Esse e-mail contém um link para o resultado da coleta de dados e os valores de perfil de cada candidato (ranking).

Na página com o resultado o contato da empresa que publicou a oferta pode renovar ou finalizar a publicação.

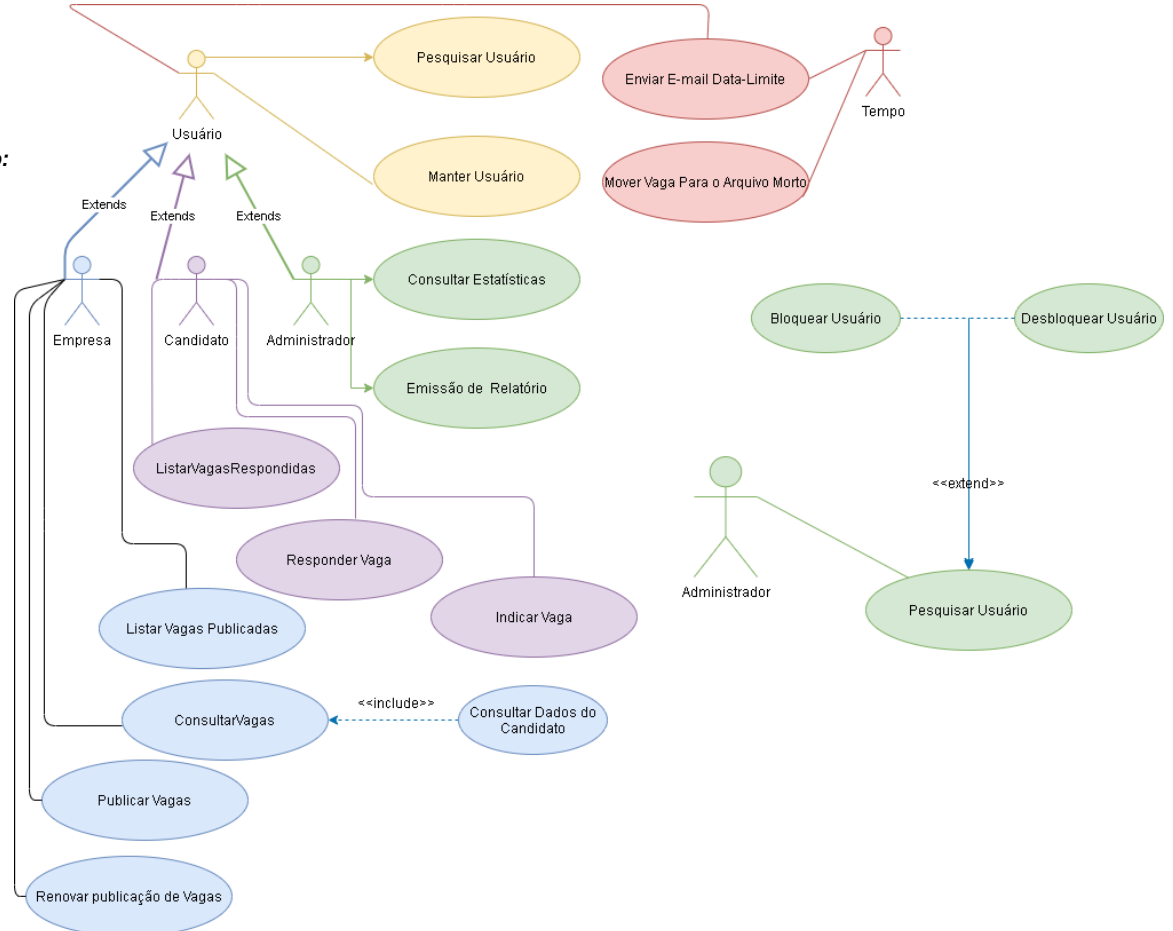
A página que exibe o resultado de uma publicação de oferta fica disponível por até um mês. A renovação ou encerramento de uma publicação poderá ser feita em até dois dias. Caso nenhuma ação seja tomada pela empresa que publicou a oportunidade, a oferta será encerrada automaticamente pelo sistema. Nessa mesma página, o contato pode obter as informações dos candidatos através de uma consulta que lista todos os que responderam à oferta, os que tiveram pontuação igual ou acima do perfil da oferta ou os x primeiros.

O sistema precisa controlar cadastro e acesso de usuários e empresas e fazer a manutenção automática das ofertas vencidas.

O sistema precisa tornar disponível para o administrador relatórios básicos de acesso por usuário, empresa e número de ofertas.

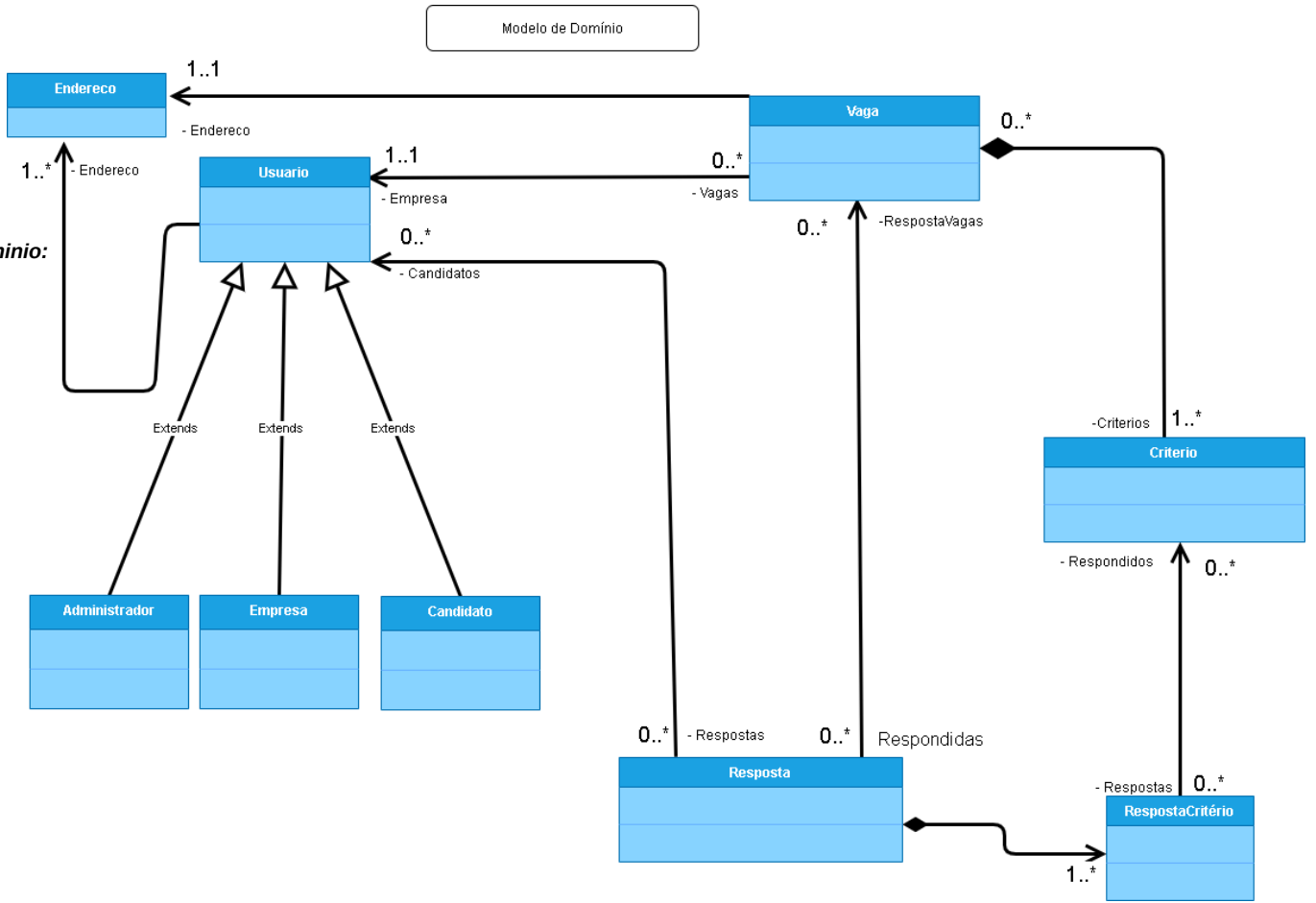
Diagramas

Diagrama de Caso de Uso:



Diagramas

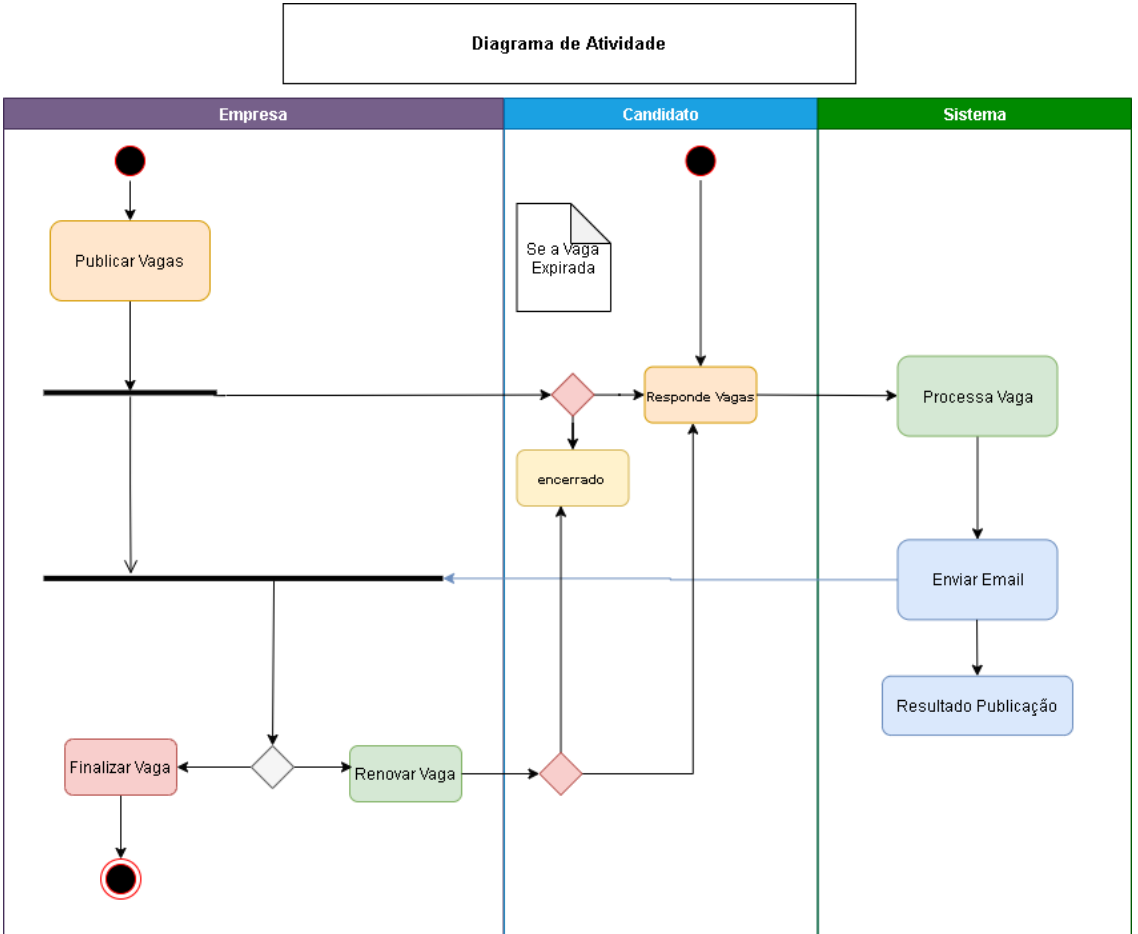
Diagrama de Dominio:

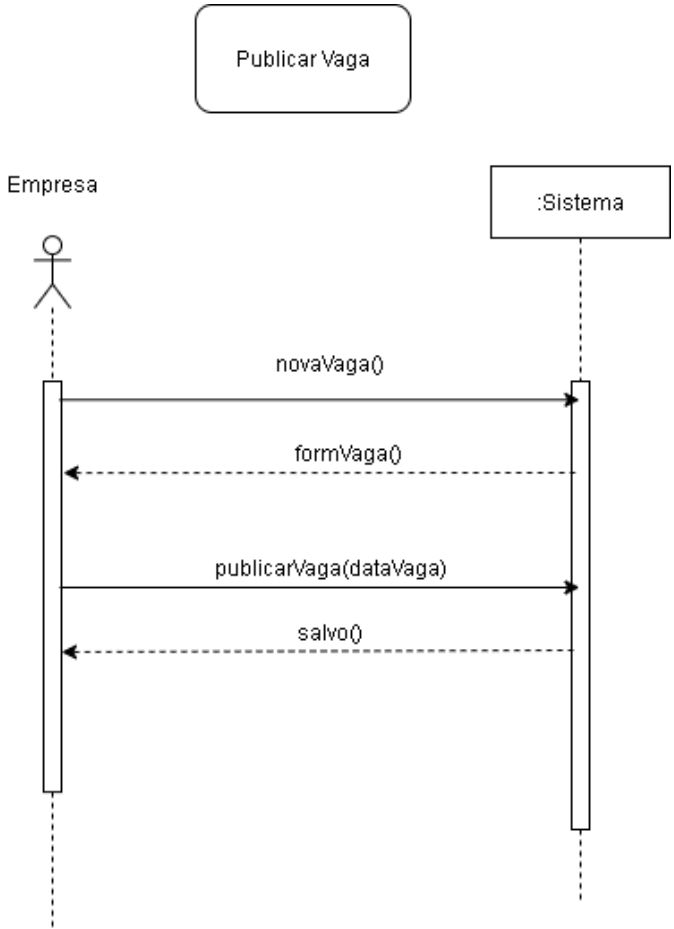


Diagramas

Diagrama de Atividade:

Fluxo Geral

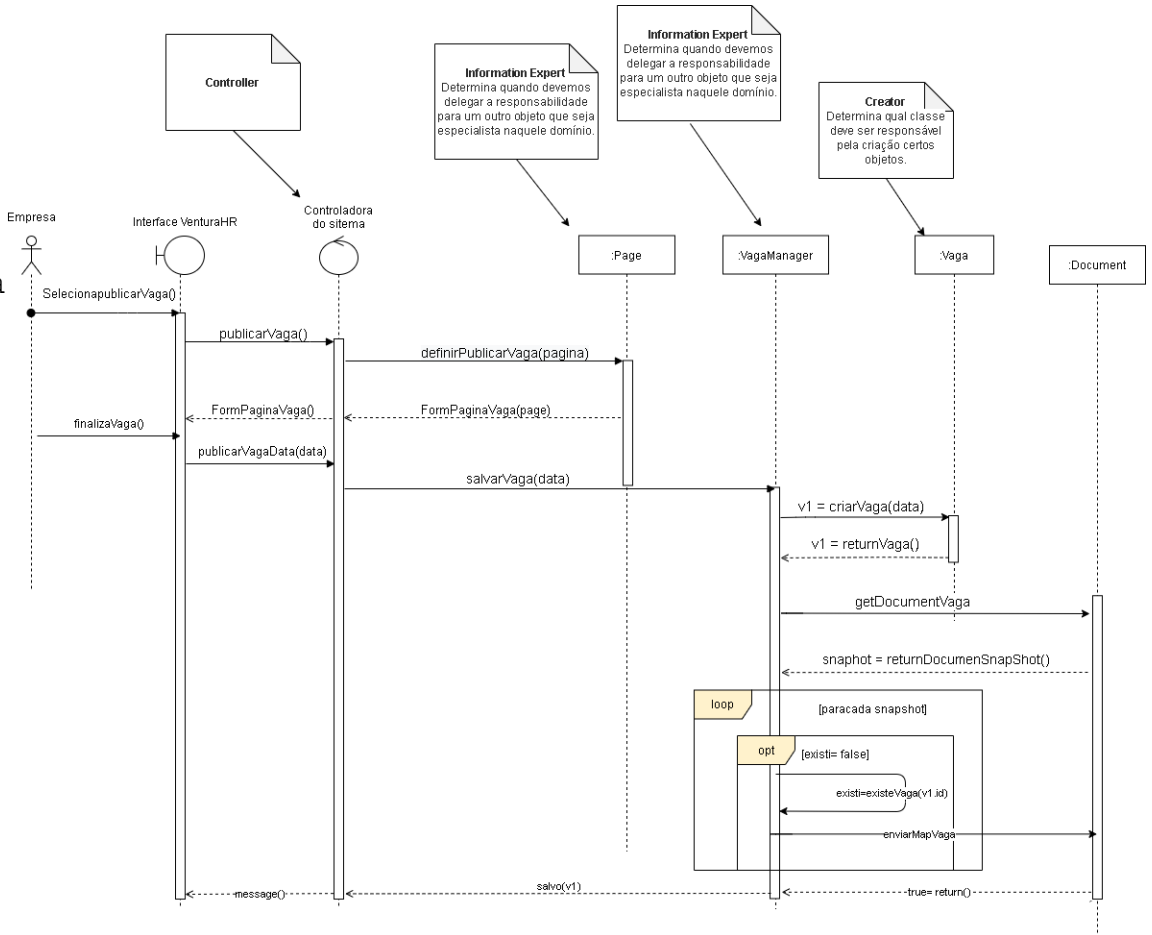



Publicar Vaga
Empresa
:Sistema
novaVaga()
formVaga()
publicarVaga(dataVaga)
salvo()

Diagramas

Diagrama de Sequência:

Publicar Vaga



Algoritimo “PublicarVaga”

```
Var         nomeVaga,cliterioVaga: literal;
var         cont:inteiro;
Inicio
            publicarVaga();
            escreva(“Insira os nomeVaga da Vaga”);
            leia(nomeVaga);
            escreva(“Insira os cliterioVaga da Vaga”);
            leia(cliterioVaga);
            objVaga <— vaga(nomeVaga, cliterioVaga);
            snapshotDocument <— buscaDocumentos();
            para cont de 1 ate snapshotDocument.cumprimento() faca
                  se (objVaga == snapshotDocument[cont])
                  escreva(“Vaga Ja existe”);
                  fimse
                  fimpara
            salvarVaga(objVaga);
```

Usaremos Processo Unificado Ágil

Porque ele é Ágil?

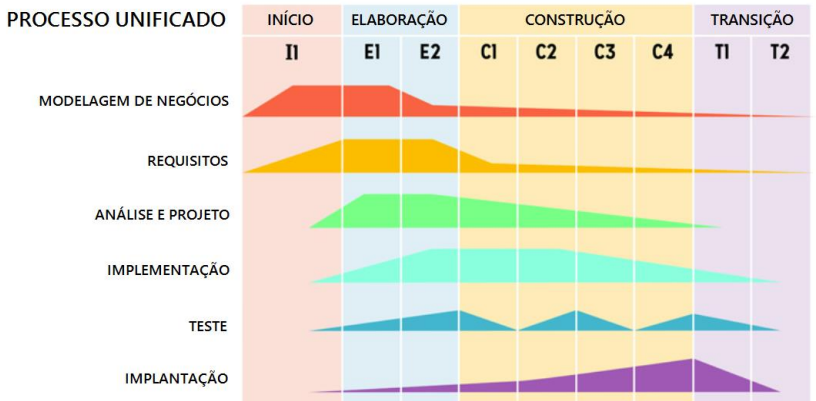
Porque Contem:

- Conjunto **pequeno** de atividades => Priorize à programação precoce e não à documentação precoce
- Requisitos e projetos emergem adaptativamente por meio de uma série de iterações, com base em **realimentação**
- **Modelagem ágil** usando UML
- Não há um plano detalhado: **Plano de Fases** e **Plano de Iteração**

Processo iterativo popular extremamente flexível

Fases:

Concepção - visão aproximada, casos de negócio, escopo e estimativas vagas
Elaboração - visão refinada, implementação iterativa da arquitetura central, resolução dos altos riscos, identificação da maioria dos requisitos e do escopo
Construção - implementação iterativa do elementos restantes de menor risco e mais fáceis
Transição - testes beta e implantação


PROCESSO UNIFICADO
INÍCIO
ELABORAÇÃO
CONSTRUÇÃO
TRANSIÇÃO
I1
E1
E2
C1
C2
C3
C4
T1
T2
MODELAGEM DE NEGÓCIOS
REQUISITOS
ANÁLISE E PROJETO
IMPLEMENTAÇÃO
TESTE
IMPLANTAÇÃO

Quais são os principais artefatos utilizados em um projeto ágil PU:

Artefato	Comentário
Visão e Caso de Negócio	Descrevem os objetivos e as restrições de alto nível, o caso de negócio, além de um resumo para executivos
Modelo de Casos de Uso	Descreve os requisitos funcionais [10 % !!!]
Especificações Suplementares	Descrevem outros requisitos não funcionais
Glossário	Contém a terminologia-chave do domínio e dicionário de dados
Lista de Riscos e Plano de Gestão de Riscos	Descrevem os riscos (de negócios, técnicos, de recursos e de cronograma)
Protótipos e Provas de conceitos	Visam a esclarecer a visão e validar as ideais técnicas
Plano da Iteração	Descreve o que fazer na prieira iteração da elaboraçãoo
Plano de Fase e Plano de Desenvolvimento de Software	Estimativa de baixa precisão para a duração e esforço da fase de elaboração. Ferramentas, pessoal, treinamento e outros recursos
Pasta de Desenvolvimento	Descrição dos passos e artefatos do PU personalizados para esse projeto.

Padrão de projeto de software

O que são padrões?

Em Engenharia de Software,um padrão de desenho (pt) ou padrão de projeto (pt-BR) (do inglês design pattern) é uma solução geral para um problema que ocorre com frequência dentro de um determinado contexto no projeto de software. Um padrão de projeto não é um projeto finalizado que pode ser diretamente transformado em código fonte ou de máquina, ele é uma descrição ou modelo (template) de como resolver um problema que pode ser usado em muitas situações diferentes. Padrões são melhores práticas formalizadas que o programador pode usar para resolver problemas comuns quando projetar uma aplicação ou sistema. Padrões de projeto orientados a objeto normalmente mostram relacionamentos e interações entre classes ou objetos, sem especificar as classes ou objetos da aplicação final que estão envolvidas. Padrões que implicam orientação a objetos ou estado mutável mais geral, não são tão aplicáveis em linguagens de programação funcional.

Padrões de projeto residem no domínio de módulos e interconexões. Em um nível mais alto há padrões arquiteturais que são maiores em escopo, usualmente descrevendo um padrão global seguido por um sistema inteiro.

As características obrigatórias que devem ser atendidas por um padrão de projeto é composto basicamente por 4 (quatro) elementos que são:

Nome do padrão;
Problema a ser resolvido;
Solução dada pelo padrão; e
Consequências.
Os padrões de projeto:

visam facilitar a reutilização de soluções de desenho - isto é, soluções na fase de projeto do software - e estabelecem um vocabulário comum de desenho, facilitando comunicação, documentação e aprendizado dos sistemas de software.

Padrões GoF ('Gang of Four')

O Que é o GOF?

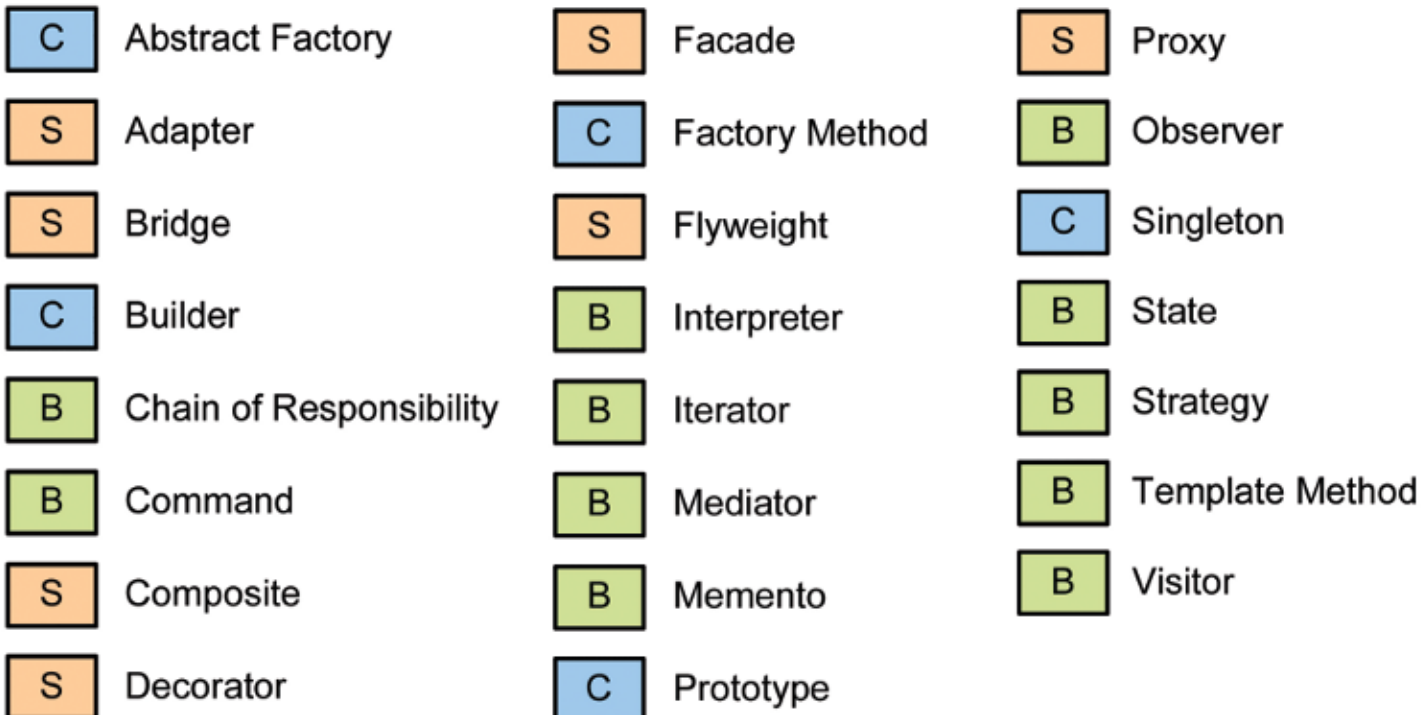
De acordo com o livro: "Padrões de Projeto: soluções reutilizáveis de software orientado a objetos", os padrões "GoF" são divididos em 24 tipos. Em função dessa grande quantidade de padrões, foi necessário classificá-los de acordo com as suas finalidades.

São 3 as classificações/famílias:

Padrões de criação: Os padrões de criação são aqueles que abstraem e/ou adiam o processo criação dos objetos. Eles ajudam a tornar um sistema independente de como seus objetos são criados, compostos e representados. Um padrão de criação de classe usa a herança para variar a classe que é instanciada, enquanto que um padrão de criação de objeto delegará a instanciação para outro objeto. Os padrões de criação tornam-se importantes à medida que os sistemas evoluem no sentido de dependerem mais da composição de objetos do que a herança de classes. O desenvolvimento baseado na composição de objetos possibilita que os objetos sejam compostos sem a necessidade de expor o seu interior como acontece na herança de classe, o que possibilita a definição do comportamento dinamicamente e a ênfase desloca-se da codificação de maneira rígida de um conjunto fixo de comportamentos, para a definição de um conjunto menor de comportamentos que podem ser compostos em qualquer número para definir comportamentos mais complexos. Há dois temas recorrentes nesses padrões.

Primeiro todos encapsulam conhecimento sobre quais classes concretas são usadas pelo sistema. Segundo ocultam o modo como essas classes são criadas e montadas. Tudo que o sistema sabe no geral sobre os objetos é que suas classes são definidas por classes abstratas. Conseqüentemente, os padrões de criação dão muita flexibilidade no que é criado, quem cria, como e quando é criado. Eles permitem configurar um sistema com objetos “produto” que variam amplamente em estrutura e funcionalidade. A configuração pode ser estática (isto é, especificada em tempo de compilação) ou dinâmica (em tempo de execução).
Padrões estruturais: Os padrões estruturais se preocupam com a forma como classes e objetos são compostos para formar estruturas maiores. Os de classes utilizam a herança para compor interfaces ou implementações, e os de objeto descrevem maneiras de compor objetos para obter novas funcionalidades. A flexibilidade obtida pela composição de objetos provém da capacidade de mudar a composição em tempo de execução o que não é possível com a composição estática (herança de classes).
Padrões comportamentais: Os padrões de comportamento se concentram nos algoritmos e atribuições de responsabilidades entre os objetos. Eles não descrevem apenas padrões de objetos ou de classes, mas também os padrões de comunicação entre os objetos. Os padrões comportamentais de classes utilizam a herança para distribuir o comportamento entre classes, e os padrões de comportamento de objeto utilizam a composição de objetos em contrapartida a herança. Alguns descrevem como grupos de objetos que cooperam para a execução de uma tarefa que não poderia ser executada por um objeto sozinho.

Padrões "GoF" organizados nas suas 3 classificações/famílias:



Padrões de criação

- Abstract Factory
- Object pool
- Builder
- Factory Method
- Prototype
- Singleton

Padrões estruturais

- Private class data
- Adapter
- Bridge
- Composite
- Decorator
- Façade (ou Facade)
- Business Delegate
- Flyweight
- Proxy

Padrões comportamentais

- Chain of Responsibility
- Command
- Interpreter
- Iterator
- Mediator
- Memento
- Observer
- State
- Strategy
- Template Method

- Visitor

Assim como os padrões SOLID, o GRASP foi criado com o intuito de tornar o código mais flexível, facilitando a manutenção e a extensibilidade.

Os padrões GRASP (General Responsibility Assignment Software Patterns) consistem em uma série de princípios baseados em conceitos para atribuição de responsabilidades a classes e objetos na construção de bons softwares usando programação orientada a objetos.

Criação

Este conjunto de princípios foram publicados originalmente no livro “Applying UML and Patterns — An Introduction to Object-Oriented Analysis and Design and the Unified Process” (obra traduzida em português com o título “Utilizando UML e Padrões — Uma introdução à análise e ao projeto orientados a objetos e ao desenvolvimento

iterativo”) escrito pelo Craig Larman em 1997, contribuindo para a codificação de princípios de design de software, esse é um livro muito popular que contribuiu para a adoção generalizada do desenvolvimento orientado a objetos

Objetivo

O conceito de responsabilidade deve ser compreendido como as obrigações que um objeto possui quando se leva em conta o seu papel dentro de um determinado contexto. Além disso, é preciso considerar ainda as prováveis colaborações entre diferentes objetos.

A implementação de soluções OO considerando os conceitos de responsabilidade, papéis e colaborações fazem parte de uma abordagem conhecida como Responsibility-Driven Design ou simplesmente RDD.

Responsibility-Driven Design é uma técnica de desenvolvimento OO proposta no início dos anos 1990, como resultado do trabalho dos pesquisadores Rebecca Wirfs-Brock e Brian Wilkerson. Este paradigma está fundamentado nas ideias de responsabilidades, papéis e colaborações.

Partindo de práticas consagradas no desenvolvimento de aplicações orientados a objeto, procuram definir quais as obrigações dos diferentes tipos de objetos em uma aplicação, o GRASP disponibiliza uma catalogo de recomendações que procuram favorecer um conjunto de práticas visando um software bem estruturado.

A responsabilidade é definida como um contrato ou obrigação de uma classe e está relacionada ao comportamento. Existem 2 tipos de responsabilidades:

Saber:

· O conhecimento dos dados privados que o objeto em questão encapsula;

· Saber a respeito de outros objetos relacionados;

· Saber sobre as coisas que pode derivar ou calcular;

Fazer:

· A execução de ações que condizem com o papel desempenhado por tal objeto

· A criação de outros objetos dos quais a instância inicial depende;

· Controlando e coordenando atividades em outros objetos

Como os outros patterns conhecidos, os diferentes padrões GRASP não devem ser encarados como soluções pré-definidas para problemas específicos. Estes patterns devem ser compreendidos como princípios que auxiliam os desenvolvedores a projetar de uma forma bem estrutura aplicações orientadas a objetos.

Catálogo

Ao todo são nove os padrões GRASP, os quais serão discutidos em maiores detalhes nas próximas seções:

Information Expert (Especialista na Informação) — Determina quando devemos delegar a responsabilidade para um outro objeto que seja especialista naquele domínio.

Creator (Criador) — Determina qual classe deve ser responsável pela criação certos objetos.

Controller (Controlador) — Atribui a responsabilidade de lidar com os eventos do sistema para uma classe que representa a um cenário de caso de uso do sistema global.

Low Coupling (Baixo Acoplamento) — Determina que as classes não devem depender de objetos concretos e sim de abstrações, para permitir que haja mudanças sem impacto.

High Cohesion (Alta Coesão) — este princípio determina que as classes devem se focar apenas na sua responsabilidade.

Polymorphism (Polimorfismo) — As responsabilidades devem ser atribuídas a abstrações e não a objetos concretos, permitindo que eles possam variar conforme a necessidade.

Pure Fabrication (Pura Fabricação) — é uma classe que não representa nenhum conceito no domínio do problema, ela apenas funciona como uma classe prestadora de serviços, e é projetada para que possamos ter um baixo acoplamento e alta coesão no sistema.

Indirection (indireção) — este princípio ajuda a manter o baixo acoplamento, através de delegação de responsabilidades através de uma classe mediadora.

Protected Variations (Proteção contra Variações) **—** Protege o sistema com a variação de componentes, encapsulando o comportamento que realmente importa.

Conclusão

Todos esses padrões servem para a resolução de problemas comuns típicos de desenvolvimento de software orientado a objeto. Tais técnicas apenas documentam e normatizam as práticas já consolidadas, testadas e conhecidas no mercado.

Práticas de engenharia de software são relativamente desconhecidas ou pouco aplicadas pelos desenvolvedores e há uma suposição de que é tudo complicado ou muito bonito no papel, mas se você realmente entender como eles funcionam e quando aplicá-los, pode-se perceber que é fácil implementar e obter seus benefícios.

Font: Applying UML and Patterns