

RESUMO ESII - Sophia Carazza

FASE CONCEITUAL → PRELIMINAR → DETAILED

Projeto Preliminar:

- Histórias de usuários e levantamento de dados
- diagramas de caso de uso
- diagramas de classes e pacotes
- protótipos de telas
- diagrama de robustez

transforma ideias iniciais e + conceituais em arquiteturas de software iniciais do sistema.

★ sistema de interface

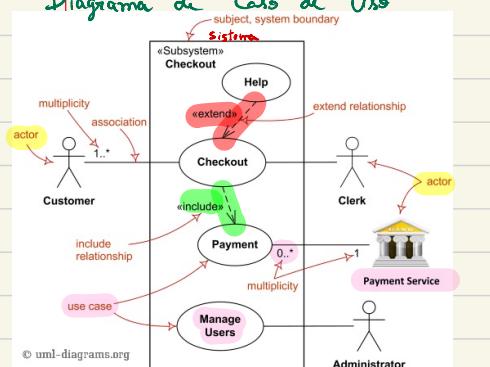
ESI

como medir

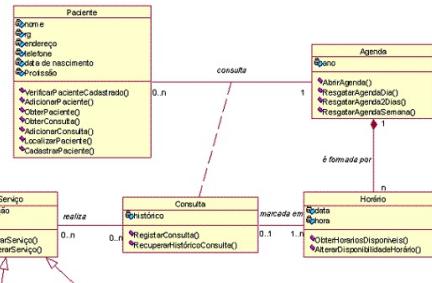
- satisfação do usuário
- tempo p/ realizar tarefas
- taxas de erro e sucesso
- uso de recursos
- frequência do uso da agenda

ESII

Diagrama de Caso de Uso



Classe e Pacotes



relacionamentos:

- **inclusão**: quando um caso de uso base requer um caso de uso para concluirlo *extenso*
- **extensão**: quando um caso de uso extende *outro* caso a partir de um base
- **generalização**: herança - não casos de uso explicitados.

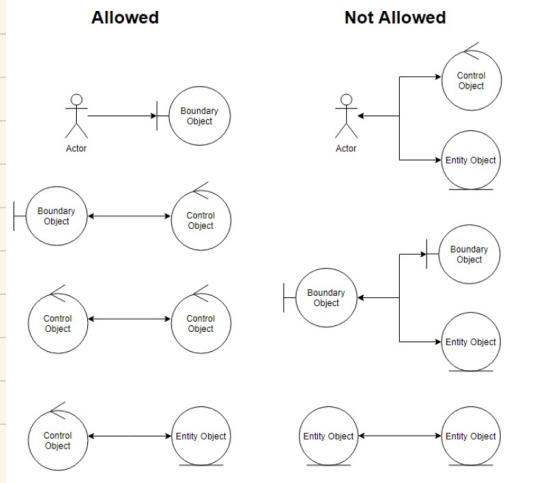
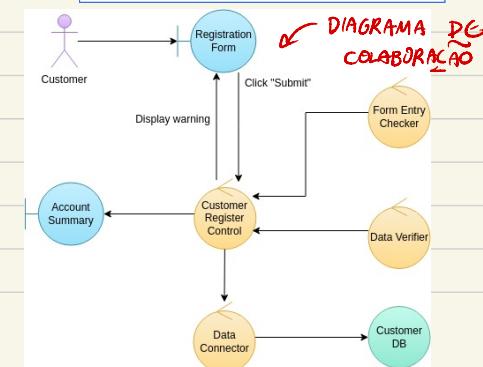
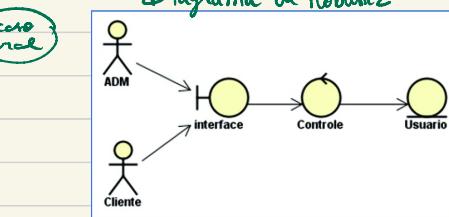
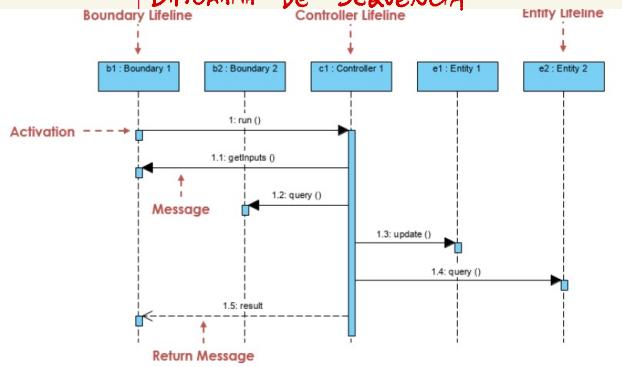


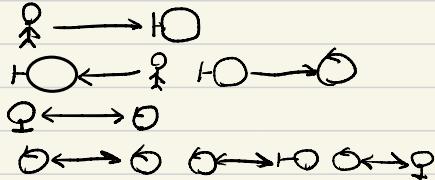
Diagrama de Robustez





- = boundary (fronteira) = interfaces entre atores e sistema (ex: telas, formulários)
- = control (controle) = coordena ações e regras de negócios entre boundary e entity.
- = entity (entidade) = representa dados ou objetos do domínio (ex: cliente, produto)

- > atores só interagem com boundary
- > boundary podem interagir c/ atores e control
- > entity só interagem com control
- > control interage c/ control, boundary e entity.



Protótipos → técnica p/ criar representações iniciais de um sistema ou parte dele, com o objetivo de explorar ideias, validar requisitos e obter feedback dos stakeholders.

- ↳ são bons p/ validar e identificar requisitos funcionais.
- ↳ pode ser refinado até se tornar o produto final.
- ↳ é usado p/ demonstrar conceitos, experimentar opções de projeto e descobrir mais sobre o problema e suas possíveis soluções.

Projeto Detalhado ↗ transformar os arquitetados iniciais em algoritmos e documentações necessárias p/ a construção exata do projeto.

↳ Deve ser feito p/ cada cenário do projeto

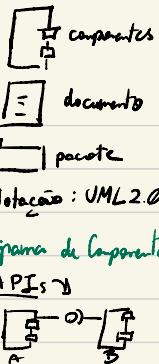
- 1- diagrama de interação de objetos ↗ diagrama de sequência
- 2- diagrama de componentes ↗ diagrama de colaboração
- 3- diagrama de implementação / deployment

Interface: Formas, características da tela ou outra interface.

Interações: Funções ou expectativas do usuário. Estimula e responde.

Usabilidade: Características e melhores práticas ao criar interfaces

↳ 10 heurísticas de Nielsen



Desenvolvimento Orientado a Comportamento

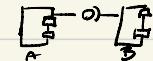
DDD - Domain Driven Development (Teste final)

TDD - Test Driven Development (Teste durante)

BDD - Behavior Driven Development (Teste inicio).

* Diagrama de Componentes

API's



> Projeto de Persistência



- **Objetivos**: integridade, consistência, desempenho, segurança e durabilidade dos dados
- **Etapas**:
 - análise de requisitos (segurança, desempenho, escalabilidade)
 - escolha do modelo de dados (relacional, não-relacional, etc)
 - escolha da ferramenta de BD (MongoDB, ChromeDB, etc)
 - projeto da estrutura de armazenamento
 - implementação da camada de persistência
 - otimização de desempenho, testes e documentação.

"Not only SQL"

> SGBD Relacional

↳ estruturado em tabelas com linhas e colunas, seguindo o modelo relacional.

- usa SQL
- vertical (aumentando a capacidade de um único servidor)

> SGBD NoSQL

↳ modelos de dados com diversos formatos como documentos, chave-valor, grafos, etc via suas linguagens

Horizontal (distribuíndo dados entre múltiplos servidores/nós).

Propriedade	ACID	BASE
Consistência	Consistência imediata	Consistência eventual
Disponibilidade	Pode sacrificar disponibilidade em cenários críticos	Alta disponibilidade prioritária
Escalabilidade	Menos escalável (vertical)	Altamente escalável (horizontal)
Uso Ideal	Transações críticas (bancos, finanças)	Grandes volumes de dados distribuídos

Métodos do HTTP } { Post = Create
Get = Read
Put = Update
Delete = Delete

Node { Django DB
Mongo DB

Diferenças em Projetos de Persistência

Implementação e Desenvolvimento

- Relacional: Geralmente utiliza ORMs (Object-Relational Mappers) como Entity Framework ?
- NoSQL: Utiliza diretamente as APIs fornecidas pelo motor NoSQL, sem necessidade de ORM ?

Modelagem de Domínio

- Relacional: Exige normalização e mapeamento cuidadoso entre objetos e tabelas ?
- NoSQL: Permite modelos mais denormalizados, facilitando a implementação de agregados

DDD em bancos orientados a documentos ?

Transações e Consistência

- Relacional: Conformidade com propriedades ACID, garantindo integridade entre objetos e tabelas ?
- NoSQL: Frequentemente segue propriedades BASE, priorizando disponibilidade sobre consistência imediata. 4

atomicidade, consistência,
isolamento e durabilidade

Quando Usar SCBD NoSQL

- Projetos com grandes volumes de dados não estruturados ou semi-estruturados 3 5
- Aplicações que requerem alta disponibilidade e escalabilidade horizontal 3 8
- Sistemas com esquemas de dados que evoluem rapidamente 3
- Cenários com alta velocidade de escrita e leitura de dados simples 6 6

Considerações de Performance

- Relacional: Excelente para consultas complexas em dados relacionados, mas pode perder performance com o crescimento do volume 8
- NoSQL: Superior em velocidade para operações simples e grande escala, mas menos eficiente para consultas complexas 4 8

10 heurísticas de Nielsen: ajudam a projetar uma boa interface e UI experience.

- 1- visibilidade do sistema
- 2- compatibilidade entre o sistema e o mundo real
- 3- controle e liberdade pr o usuário
- 4- consistência e padronizações
- 5- prevenção de erros
- 6- reconhecimento em vez de memorização
- 7- eficiência, flexibilidade de uso, design minimalista e estético.
- 8- ajudar na recuperação, diagnóstico e reconhecimento de erros
- 9- ajuda e documentação.
- 10- ajuda e documentação.

ARQUITETURA LÓGICA vs ARQUITETURA FÍSICA

Representa o modelo conceitual e abstrato do sistema, descrevendo como os componentes se interagem e se organizam para atender aos requisitos funcionais.

- organização dos dados
 - fluxos de informação
 - regras de negócio
 - interação entre componentes
- ex: diagramas UML, fluxogramas, etc

Representa a implementação concreta do sistema, detalhando a infraestrutura tecnológica necessária pr suportar o funcionamento do modelo lógico

- hardware
 - servidores e redes
 - SOs
 - bancos de dados físicos
- ex: especificações da BD(MYSQL, etc)

Design Pattern → É uma solução reutilizável p/ problemas recorrentes no design de interfaces. É uma abordagem de reuso que utiliza abstrações genéricas, não incluindo detalhes de implementação, mostrando objetos abstratos, concretos e interação.

Design System → É uma abordagem de projeto de interface e interação que padroniza para os fornecedores, clientes e colaboradores de uma organização, característicos como layouts, componentes padrão, etc. Assim, todos produtos web seguem os mesmos princípios em interfaces, padronizando e facilitando manutenção e redução de custos de projeto.

Framework → Projeto de subistema composto por um conjunto de classes abstratas e concretas. Estabelece a arquitetura para aplicações em um domínio. Uma aplicação específica é construída a partir de subclases específicas p/ aplicação. A reutilização leva à invenção de controle.

Design Visual → A função do site e a informação devem ser soberanas sobre o desenho. Qualquer tipo de conformação que beneficie o desenho em detrimento da informação, usabilidade e funcionalidade do site deve ser abandonada.

BAIXO ACOPLAMENTO E ALTA COESÃO:

Na computação, o **acoplamento** corresponde ao grau de interdependência entre dois componentes de um software. Quando há um baixo acoplamento, um componente consegue operar de forma praticamente independente do outro. Alto acoplamento já implica em uma conexão forte entre os dois, que pode torná-los até indistinguíveis. Em geral buscamos um baixo acoplamento também em função de outro conceito chamado **modularização**. Com blocos de código menores e mais simples, é mais fácil dar manutenção em um software, entender o seu funcionamento e até reutilizar aquele módulo sem a necessidade de repetir o código.

Já a **coesão** diz respeito ao propósito de um componente, ou o grau em que as partes de dele estão relacionadas funcionalmente. Quando a coesão é alta, é fácil compreender a função ou foco do componente. Ele só é utilizado para uma única coisa. Quando é baixa, o componente geralmente faz tantas coisas que é difícil compreender qual foi a intenção de quem o projetou. Outra forma de definir a coesão é o **grau de clareza das funcionalidades de um componente**. Por esses motivos, uma coesão alta é desejável.

Em outras palavras, ao criar um design com baixo acoplamento e alta coesão, estamos tornando ele mais modularizado, elegante, simples de entender e fácil de usar.

ou seja
flexibilidade, facilidade de manutenção, reusabilidade, testabilidade, redução de riscos e melhor organização do código.

Diagrama de Componentes UML

↳ estrutura física
do sistema

↳ módulo de classes que representa sistemas ou sub-sistemas
independentes c/ capacidade de interagir c/ o restante do sistema

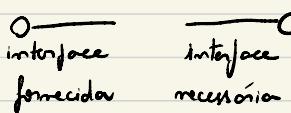
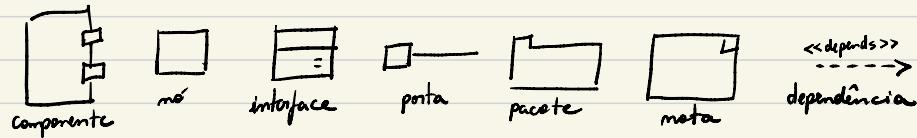


Diagrama de Pecotes

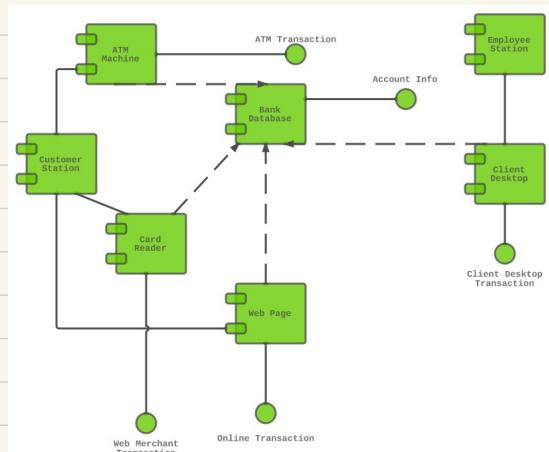
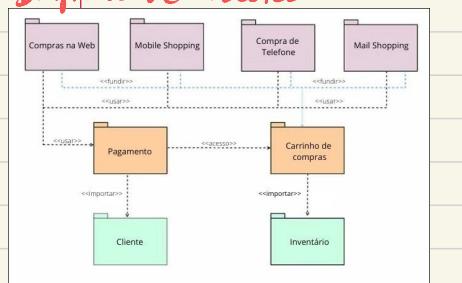


Diagrama de Implementação

↳ modelagem do ambiente de execução do sistema. São compostos por nós e associações

↳ uma associação entre dois nós representa uma conexão física entre os nós, como uma conexão Ethernet, linha serial ou link de satélite.

