

ENGENHARIA DE SOFTWARE

Ciclo de Vida do Desenvolvimento de Software

Professor Giuliano Bertoti

<https://github.com/giulianobertoti>

<https://abseil.io/resources/swe-book/html/toc.html>

We see three critical differences between programming and software engineering: time, scale, and the trade-offs at play. On a software engineering project, engineers need to be more concerned with the passage of time and the eventual need for change. In a software engineering organization, we need to be more concerned about scale and efficiency, both for the software we produce as well as for the organization that is producing it. Finally, as software engineers, we are asked to make more complex decisions with higher-stakes outcomes, often based on imprecise estimates of time and growth.

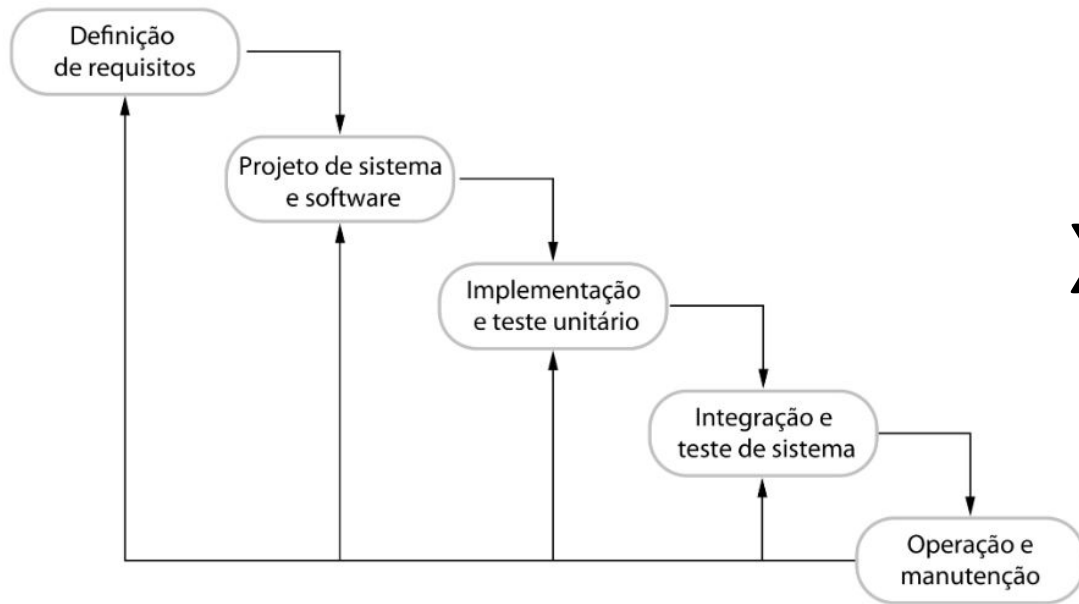
Within Google, we sometimes say, “Software engineering is programming integrated over time.” Programming is certainly a significant part of software engineering: after all, programming is how you generate new software in the first place. If you accept this distinction, it also becomes clear that we might need to delineate between programming tasks (development) and software engineering tasks (development, modification, maintenance). The addition of time adds an important new dimension to programming. Cubes aren’t squares, distance isn’t velocity. Software engineering isn’t programming.

Github - repositório “bertoti” - pasta “engenharia-de-software

1. Comentários so livro Software Engineering at Google
2. Três exemplos de trade-offs
3. Requisito não funcional de usabilidade - avaliar 10 heurísticas
4. Diagrama de Casos de Uso
5. Diagrama de Classes UML (várias versões aperfeiçoando a cada uma)
6. Código Java (espelhando as classes UML)
7. Testes automatizados com técnica de teste

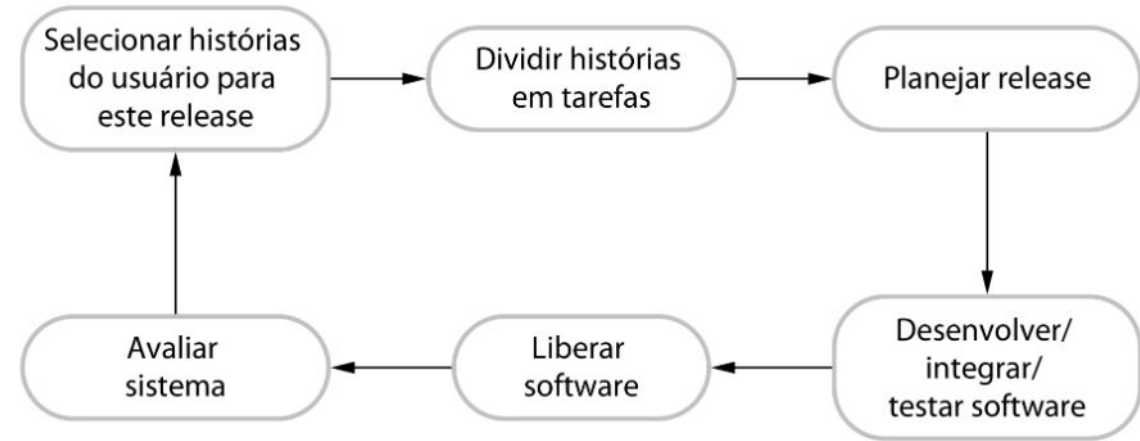
Pergunta	Resposta
O que é software?	Softwares são programas de computador e documentação associada. Produtos de software podem ser desenvolvidos para um cliente específico ou para o mercado em geral.
Quais são os atributos de um bom software?	Um bom software deve prover a funcionalidade e o desempenho requeridos pelo usuário; além disso, deve ser confiável e fácil de manter e usar.
O que é engenharia de software?	É uma disciplina de engenharia que se preocupa com todos os aspectos de produção de software.
Quais são as principais atividades da engenharia de software?	Especificação de software, desenvolvimento de software, validação de software e evolução de software.
Qual a diferença entre engenharia de software e ciência da computação?	Ciência da computação foca a teoria e os fundamentos; engenharia de software preocupa-se com o lado prático do desenvolvimento e entrega de softwares úteis.
Qual a diferença entre engenharia de software e engenharia de sistemas?	Engenharia de sistemas se preocupa com todos os aspectos do desenvolvimento de sistemas computacionais, incluindo engenharia de hardware, software e processo. Engenharia de software é uma parte específica desse processo mais genérico.
Quais são os principais desafios da engenharia de software?	Lidar com o aumento de diversidade, demandas pela diminuição do tempo para entrega e desenvolvimento de software confiável.
Quais são os custos da engenharia de software?	Aproximadamente 60% dos custos de software são de desenvolvimento; 40% são custos de testes. Para software customizado, os custos de evolução frequentemente superam os custos de desenvolvimento.
Quais são as melhores técnicas e métodos da engenharia de software?	Enquanto todos os projetos de software devem ser gerenciados e desenvolvidos profissionalmente, técnicas diferentes são adequadas para tipos de sistemas diferentes. Por exemplo, jogos devem ser sempre desenvolvidos usando uma série de protótipos, enquanto sistemas de controle críticos de segurança requerem uma especificação analisável e completa. Portanto, não se pode dizer que um método é melhor que outro.
Quais diferenças foram feitas pela Internet na engenharia de software?	A Internet tornou serviços de software disponíveis e possibilitou o desenvolvimento de sistemas altamente distribuídos baseados em serviços. O desenvolvimento de sistemas baseados em Web gerou importantes avanços nas linguagens de programação e reuso de software.

Processos



Tradicional

X



Ágeis

(Sommerville 9ª edição)

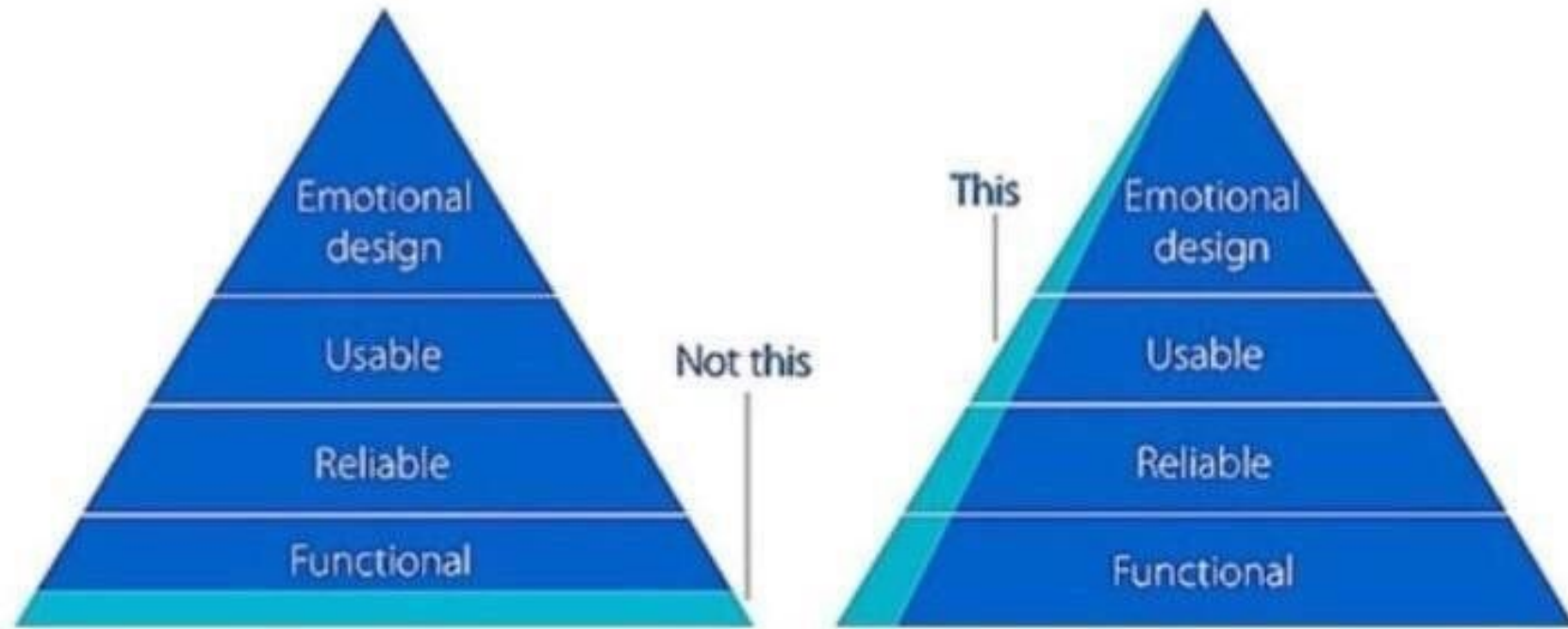
Tabela 3.1 Os princípios dos métodos ágeis

Princípios	Descrição
Envolvimento do cliente	Os clientes devem estar intimamente envolvidos no processo de desenvolvimento. Seu papel é fornecer e priorizar novos requisitos do sistema e avaliar suas iterações.
Entrega incremental	O software é desenvolvido em incrementos com o cliente, especificando os requisitos para serem incluídos em cada um.
Pessoas, não processos	As habilidades da equipe de desenvolvimento devem ser reconhecidas e exploradas. Membros da equipe devem desenvolver suas próprias maneiras de trabalhar, sem processos prescritivos.
Aceitar as mudanças	Deve-se ter em mente que os requisitos do sistema vão mudar. Por isso, projete o sistema de maneira a acomodar essas mudanças.
Manter a simplicidade	Focalize a simplicidade, tanto do software a ser desenvolvido quanto do processo de desenvolvimento. Sempre que possível, trabalhe ativamente para eliminar a complexidade do sistema.

(Sommerville 9ª edição)

Estamos usando métodos ágeis no PI - Perceba como cada um dos princípios acima faz parte do nosso ciclo de vida do desenvolvimento de software (ciclo de vida que é composto por atividades fundamentais presentes em qualquer processo como, por exemplo, análise de requisitos, projeto do software, desenvolvimento, testes ...).

Minimum Viable Product



What MVP really means

Atividade de Requisitos

- Coletar os requisitos funcionais e não-funcionais do sistema.
- (O que são requisitos? São as necessidades do cliente!)
- Funcionais: são tarefas ou ações do sistema
- Não-funcionais: são qualidades do sistema

Atividade de Requisitos

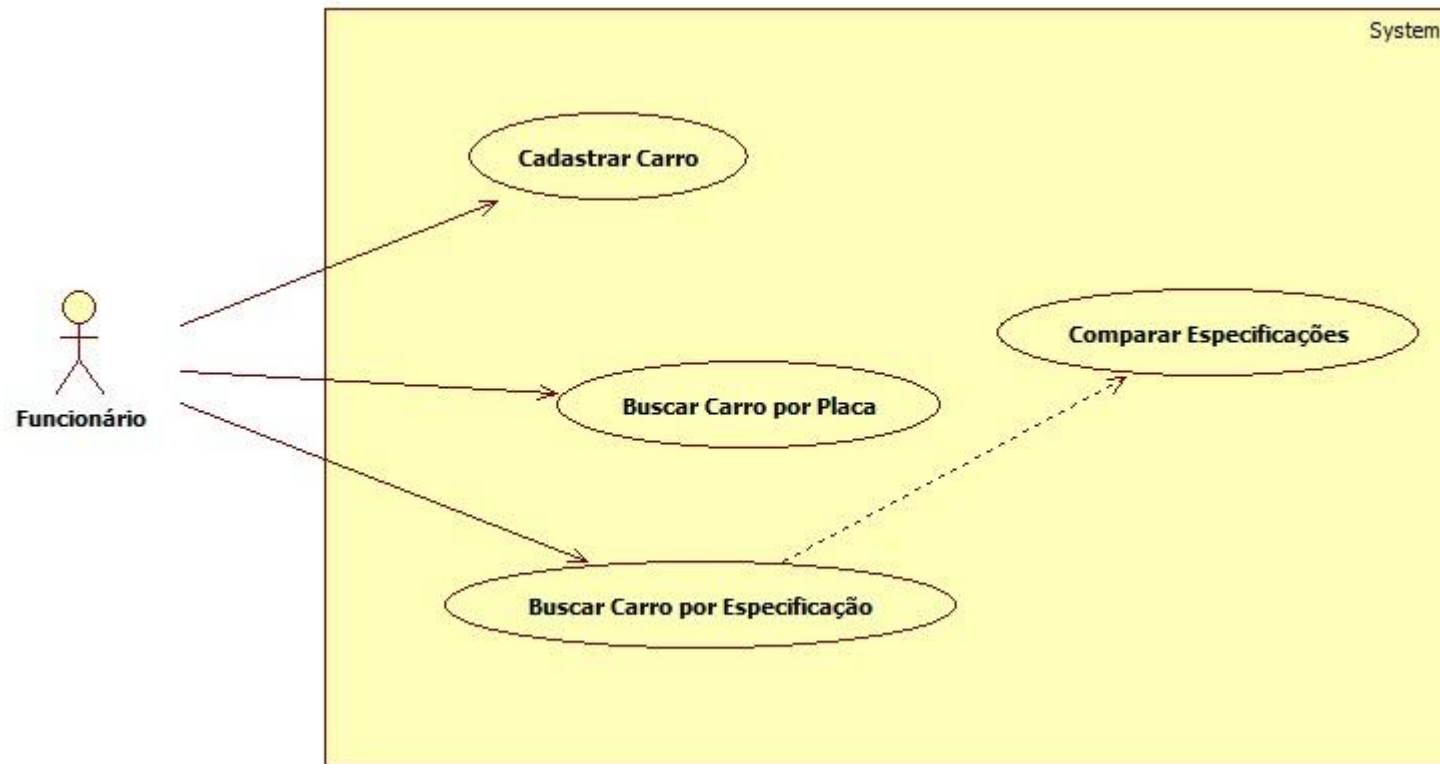
- Funcionais (story cards):
 - O funcionário do estacionamento (ator) pode cadastrar a entrada de carros inserindo placa, marca, modelo e cor do carro;
 - O funcionário do estacionamento pode buscar um conjunto de carros passando para o sistema marca, modelo e cor.
 - O funcionário do estacionamento pode buscar 1 carro passando para o sistema sua placa.
 - Etc...

Atividade de Requisitos

- Você pode também especificar os requisitos funcionais utilizando-se da UML (mais formal):
 - Diagrama de Casos de Uso
 - Diagrama de Sequência

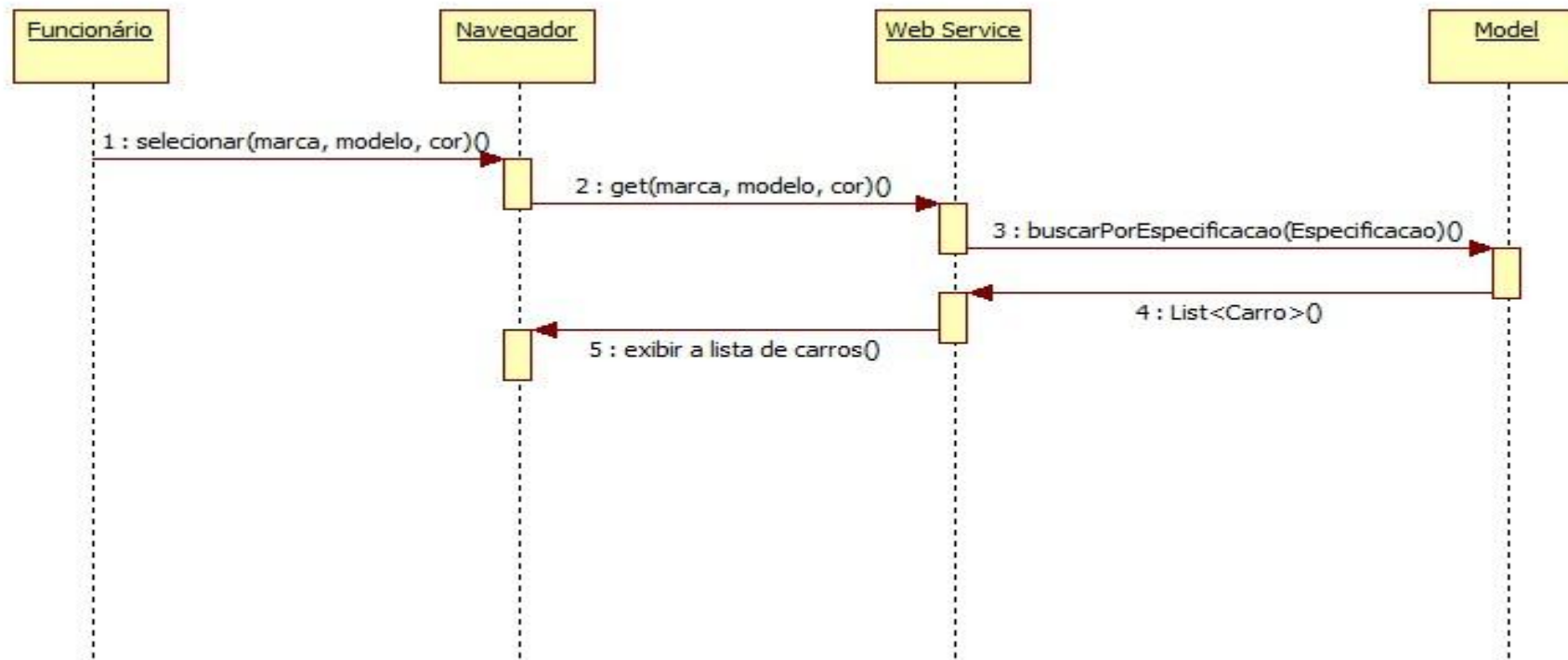
Atividade de Requisitos

- Casos de Uso



Atividade de Requisitos

- Sequência



Atividade de Requisitos

- Não-funcionais:
 - Separação de Interesses: definir uma arquitetura de aplicação com modelo MVC (separando assim lógica de negócios, comportamentos e interação com usuário).
 - Portabilidade: executar o sistema em diferentes plataformas (ex. web, nativa iOS, nativa Android e etc). Importante: a Separação de Interesses do item anterior me ajuda nisso, porque como minha lógica de negócios está encapsulada em um módulo do sistema, se torna possível reutilizá-las em diferentes plataformas.
 - Usabilidade: criar uma interface de simples acesso e uso às funções do sistema.
 - Etc.. (Desempenho é desejável também)

Atividade de Projeto

- Transformar os requisitos (tanto funcionais quanto não-funcionais) em algo que possa ser implementado.

Analyzing Trade-Offs

Thinking like an architect is all about seeing trade-offs in every solution, technical or otherwise, and analyzing those trade-offs to determine what is the best solution. To quote Mark (one of your authors):

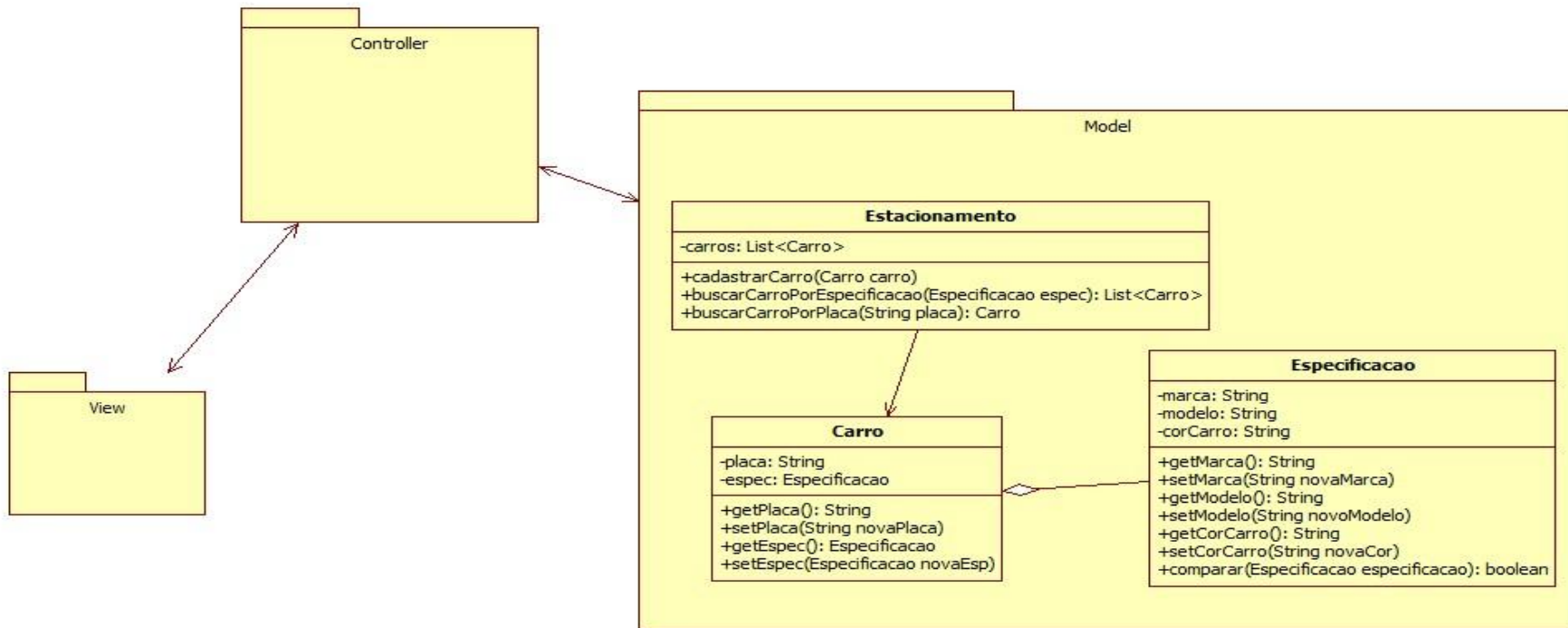
Architecture is the stuff you can't Google or ChatGPT.

Atividade de Projeto

- Projeto da Arquitetura do Sistema
 - O Projeto de Arquitetura me ajuda a obter os requisitos não-funcionais “Separação de Interesses” e “Portabilidade”.
 - Será utilizado o Diagrama de Classes da UML (Unified Modeling Language)
 - Lembre-se: Classes são nomeadas como substantivos com todas as primeiras letras maiúsculas, atributos como substantivos com a primeira minúscula e métodos como verbos com a primeira minúscula (e as demais maiúsculas)
 - Note: os métodos das Classes são os meus requisitos funcionais.

Atividade de Projeto

- Projeto da Arquitetura do Sistema (Diagrama de Classes UML)

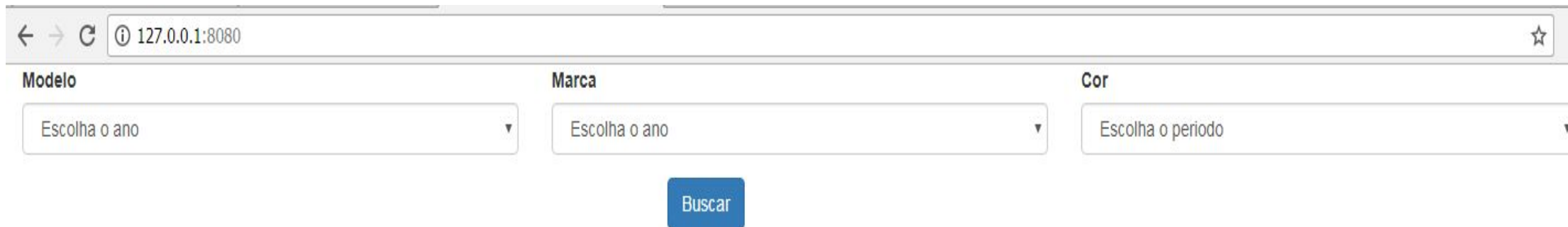


Atividade de Projeto

- Projeto de Interação com o Usuário
 - O Projeto de Interação com o Usuário me ajuda a obter o requisito não-funcional de Usabilidade.
 - Veja Usability Guidelines (princípios norteadores)
 - <https://www.nngroup.com/articles/ten-usability-heuristics/>

Atividade de Projeto

- Projeto de Interação com o Usuário
 - Estética de Design Minimalista (8): “cada unidade extra de informação compete [na cognição do usuário] com unidades relevantes”.
 - Na busca por especificação do carro, por exemplo, oferecer para o funcionário apenas as informações de fato relevantes: marca, modelo e cor do carro.



A screenshot of a web interface for searching car specifications. At the top is a browser address bar showing the URL '127.0.0.1:8080'. Below it are three dropdown menus labeled 'Modelo', 'Marca', and 'Cor'. The 'Modelo' dropdown has the placeholder text 'Escolha o ano'. The 'Marca' dropdown also has the placeholder text 'Escolha o ano'. The 'Cor' dropdown has the placeholder text 'Escolha o período'. Below these dropdowns is a blue button labeled 'Buscar'.

Atividade de Projeto

- Projeto de Interação com o Usuário
 - Reconhecimento ao invés de lembrança (6): “minimize o carregamento de memória do usuário...”.
 - No nosso caso, por exemplo, oferecer uma lista de marcas ao invés de requisitar que o funcionário se lembre disso.

Marca

Escolha o ano ▼

Escolha o ano

vw

fiat

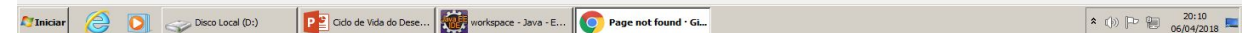
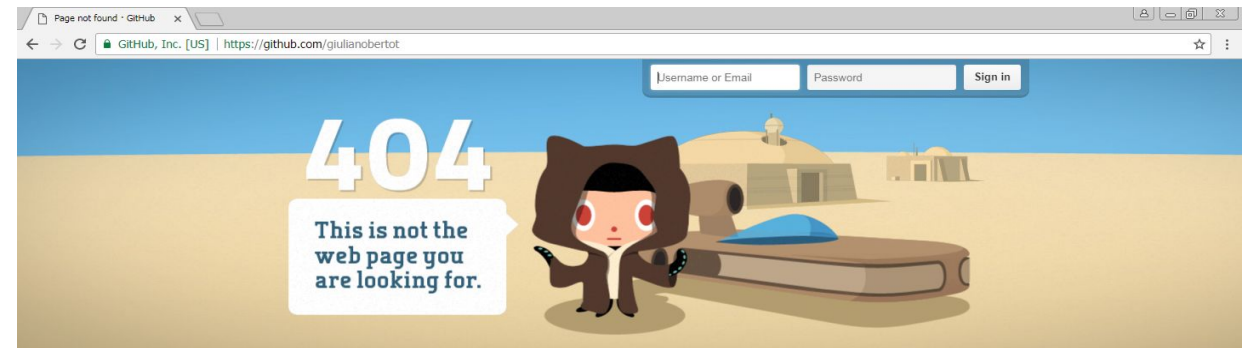
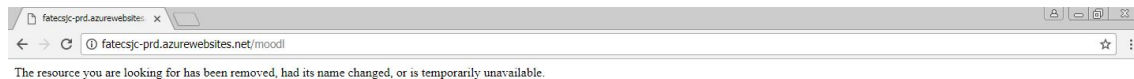
ferrari

Atividade de Projeto

- Projeto de Interação com o Usuário
 - Prevenção de Erros (5): “design que previna o usuário a entrar, por exemplo, com valores errados”.
 - Ex. Ao lado do campo de busca por placa, mostrar um exemplo de como o valor da placa deve ser preenchido: AAA-1111 ou AAA1111 (você pode, além disso, colocar máscaras nos campos para prevenir entradas indesejadas)

Atividade de Projeto

- Projeto de Interação com o Usuário
 - “Ajudar o usuário a diagnosticar, reconhecer e corrigir problemas” (9)




19:04

56%


etapa

[Apresentação](#)

[Planejar](#)

ação

[Copiar Plano](#)

[Planejar Aula](#)

☐ Mostra apenas a disciplina

☐ Mostra somente na

Status

[x] Apresentação

[x] Planejamento de aulas

[x] Materiais

[x] Plano de avaliação

[x] Extra-Classe

[x] Bibliografia

Processo de cópia finalizada

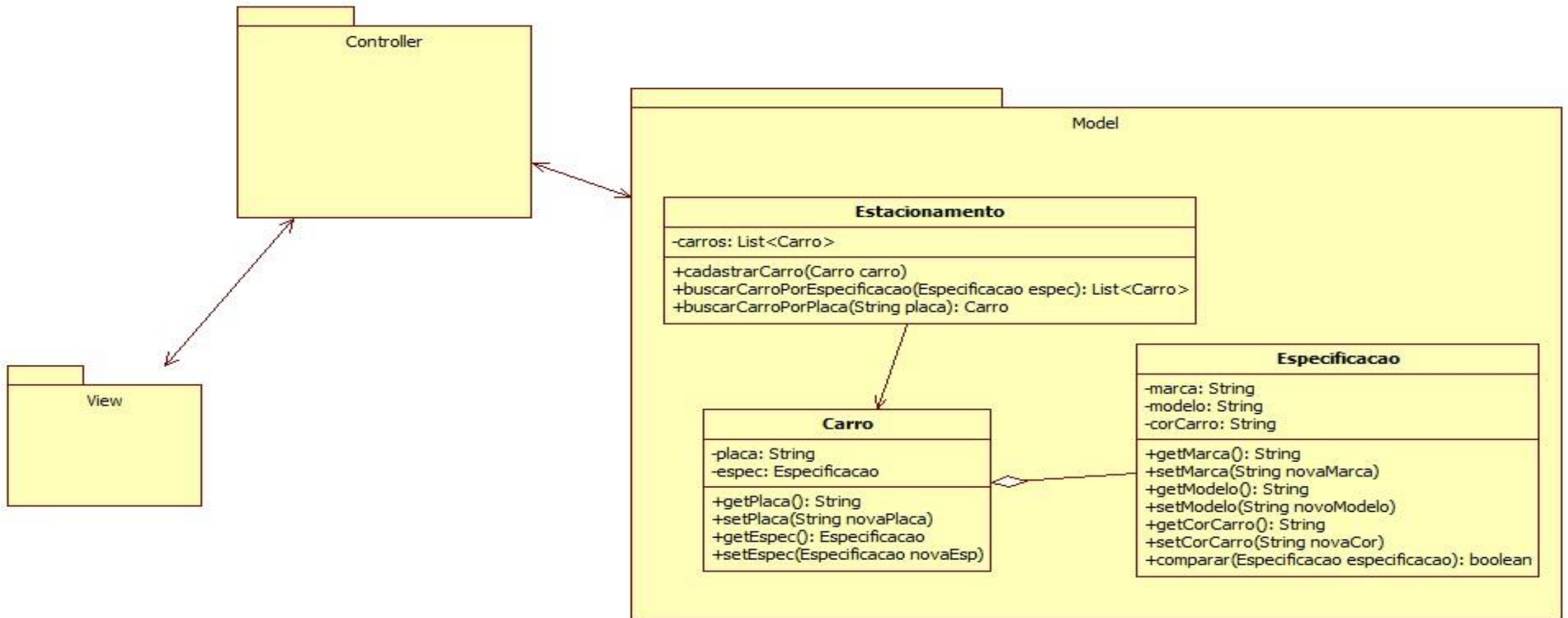
Planos Anteriores

1) Selecione uma disciplina anterior que contém o plano de e

Sem/Ano	Sigla	Disciplina	Uni
20232	TTG121	A Trabalho de Graduação I	FAT SJC
20232	IBD183	A Laboratório de Desenvolvimento em Banco de Dados III	FAT SJC
20232	IES001	A Engenharia de Software	FAT SJC
20232	IPP002	A Padrões de Projetos de Sistemas	FAT SJC
20232	TTG001	A Metodologia da Pesquisa Científico-Tecnológica	FAT SJC
20232	TTG021	A Projeto Trabalho de Graduação em Banco de Dados I	FAT SJC
20232	IES100	A Engenharia de Software I	FAT SJC
20232	IES300	A Engenharia de Software III	FAT SJC
» 20232	IHC001	A Interação Humano Computador	FAT SJC

Atividade de Desenvolvimento

- Iniciaremos o desenvolvimento propriamente dito (implementação) do software a partir da sua arquitetura do sistema (que definimos no Diagrama de Classes UML na atividade anterior: “Atividade de Projeto”).



<https://drive.google.com/file/d/14Ag8l7cmzkdYyg53iVLRs1O26Sz-7kPe/view?usp=sharing>

Atividade de Desenvolvimento

- Para a edição de código, compilação, execução, construção (build do projeto) e etc do projeto, utilizamos uma ferramenta CASE ou IDE de desenvolvimento.
- Aqui, vamos utilizar o Eclipse (mas você poderia escolher o IntelliJ, por exemplo).
- Coloquei no moodle um pdf que oferece uma introdução ao Eclipse (para quem perder algo que estou explicando aqui no VNC).

1.Instalar JRE (meu caso windows
offline x64)

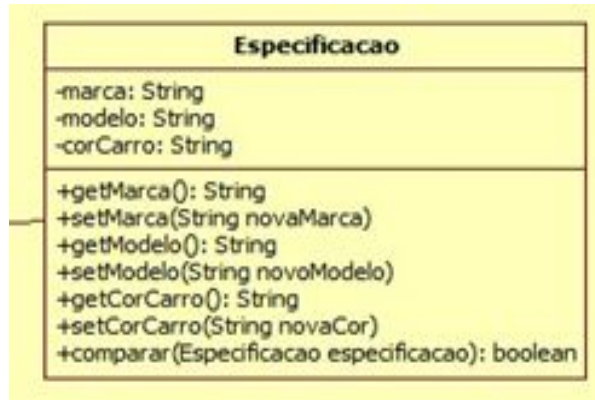
<https://java.com/en/download/manual.jsp>

2. Baixar Eclipse (Eclipse IDE for
Java Developers)

<https://www.eclipse.org/downloads/packages/>



```
public class Carro {  
  
    private String placa;  
    private Especificacao espec;  
  
    public Carro(String placa, Especificacao espec) {  
        this.placa = placa;  
        this.espec = espec;  
    }  
  
    public String getPlaca(){  
        return placa;  
    }  
  
    public void setPlaca(String novaPlaca){  
        placa = novaPlaca;  
    }  
  
    public Especificacao getEspec() {  
        return espec;  
    }  
  
    public void setEspec(Especificacao espec) {  
        this.espec = espec;  
    }  
  
}
```



```
public class Especificacao {

    private String marca;
    private String modelo;
    private String corCarro;

    public Especificacao(String marca, String modelo, String corCarro) {
        this.marca = marca;
        this.modelo = modelo;
        this.corCarro = corCarro;
    }

    public String getMarca() {
        return marca;
    }

    public void setMarca(String novaMarca) {
        this.marca = novaMarca;
    }

    public String getModelo() {
        return modelo;
    }

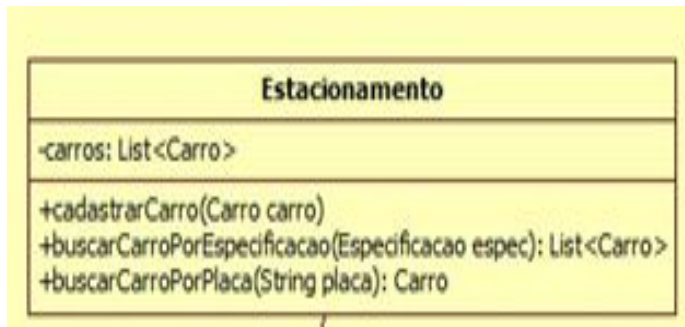
    public void setModelo(String novoModelo) {
        this.modelo = novoModelo;
    }

    public String getCorCarro() {
        return corCarro;
    }

    public void setCorCarro(String novaCor) {
        this.corCarro = novaCor;
    }

    public boolean comparar(Especificacao especificacao){
        if(this.marca.equals(especificacao.marca) &&
this.modelo.equals(especificacao.modelo)
&&
this.corCarro.equals(especificacao.corCarro)){
            return true;
        } else {
            return false;
        }
    }

}
```

```
import java.util.LinkedList;
import java.util.List;

public class Estacionamento {

    private List<Carro> carros = new LinkedList<Carro>();

    public void cadastrarCarro(Carro carro){
        carros.add(carro);
    }

    public List<Carro> buscarCarroPorEspecificacao(Especificacao espec){
        List<Carro> carrosEncontrados = new LinkedList<Carro>();
        for(Carro carro:carros){
            if(carro.getEspec().comparar(espec)) carrosEncontrados.add(carro);
        }
        return carrosEncontrados;
    }

    public Carro buscarCarroPorPlaca(String placa){
        for(Carro carro:carros){
            if(carro.getPlaca().equals(placa)) return carro;
        }
        return null;
    }

    public List<Carro> getCarros(){
        return carros;
    }
}
```

Atividade de Desenvolvimento

- Implementamos juntos no VNC
- Agora, precisamos testar!

Atividade de Teste

- O que é testar? É (tentar) garantir que o software faz o que deveria fazer.
- É possível testar tudo? Não! Por que não?

```
funcao(int a){  
  
    b = a + 1 //deveria ser a - 1  
    c = b / 30000  
    return c  
  
}
```

Vou considerar apenas inteiros no range -32768 até +32767

Dentre estes 65536 números, apenas 4 revelam o defeito do software (-29999, 29999, -30000, 30000)

Atividade de Teste

- Nunca testar aleatoriamente, mas sim utilizar técnicas de teste de software.
- Por exemplo, no caso anterior uma técnica chamada “Teste de Valor Limite” poderia ser utilizada.
 - Logo, eu encontraria os 4 números mais facilmente testando valores “em cima” e próximos de 30000 (por que a divisão antes do retorno é por este número).

Atividade de Teste

- Para este caso do estacionamento, podemos utilizar uma técnica chamada “Testes por Classes de Equivalência”.
- Vamos começar pelo método de “cadastrarCarro”:
 - Se eu adiciono 5 carros, certamente adiciono 6 (5 e 6 fazem parte da mesma classe de equivalência)
 - Vamos definir as classes de equivalência neste caso:
 - 1 classe de equivalência: 0 carros (lista vazia)
 - 1 classe de equivalência: 1 carro (guardando 1 objeto)
 - 1 classe de equivalência: n carros (utilizando a Lista para guardar n carros)

Atividade de Teste

- Método “buscarCarroPorEspecificacao”:
 - 1 classe de equivalência: 0 carros encontrados (estou testando “lista vazia”)
 - 1 classe de equivalência: 1 carro (guardando 1 objeto)
 - 1 classe de equivalência: n carros (encontrando n carros)
- Método “buscarCarroPorPlaca”:
 - 1 classe de equivalência: retorna 1 carro
 - 1 classe de equivalência: retorna null

Atividade de Teste

- Toda linguagem de programação possui bibliotecas para auxiliar no teste. No nosso caso, vamos utilizar o Junit. No Eclipse: File-> New -> Junit Test Case.

Atividade de Teste

```
import static org.junit.Assert.*;

import org.junit.Test;

public class Teste {

    @Test
    public void test() {

        Estacionamento estacionamento = new Estacionamento();

        //Testes de Classes de Equivalencia para o método cadastrarCarro

        //1 classe de equivalência: 0 carros (lista vazia)

        assertEquals(estacionamento.getCarros().size(), 0);

        //1 classe de equivalência: 1 carro (guardando 1 objeto)

        estacionamento.cadastrarCarro(new Carro("AAA1111", new Especificacao("VW", "fusca", "verde")));

        assertEquals(estacionamento.getCarros().size(), 1);

        //1 classe de equivalência: n carros (utilizando a Lista para guardar n carros)

        estacionamento.cadastrarCarro(new Carro("BBB1111", new Especificacao("VW", "fusca", "amarelo")));
        estacionamento.cadastrarCarro(new Carro("CCC1111", new Especificacao("VW", "variant", "azul")));

        assertEquals(estacionamento.getCarros().size(), 3);

    }

}
```

Atividade de Teste

//Testes de Classes de Equivalencia para o método buscarCarroPorEspecificacao

//1 classe de equivalência: 0 carros encontrados (estou testando “lista vazia”)

List<Carro> encontrados = estacionamento.buscarCarroPorEspecificacao(new Especificacao("fiat", "fusca", "azul"));

assertEquals(encontrados.size(), 0);

//1 classe de equivalência: 1 carro (guardando 1 objeto)

List<Carro> encontrados2 = estacionamento.buscarCarroPorEspecificacao(new Especificacao("VW", "fusca", "verde"));

assertEquals(encontrados2.size(), 1);

//1 classe de equivalência: n carros (encontrando n carros)

estacionamento.cadastrarCarro(new Carro("FFF1111", new Especificacao("VW", "fusca", "amarelo")));

estacionamento.cadastrarCarro(new Carro("ABC1111", new Especificacao("VW", "fusca", "amarelo")));

List<Carro> encontrados3 = estacionamento.buscarCarroPorEspecificacao(new Especificacao("VW", "fusca", "amarelo"));

assertEquals(encontrados3.size(), 3);

//Testes de Classes de Equivalencia para o método buscarCarroPorPlaca

//1 classe de equivalência: retorna 1 carro

Carro carro1 = estacionamento.buscarCarroPorPlaca("ABC1111");

assertEquals(carro1.getEspec().getModelo(), "fusca");

assertEquals(carro1.getEspec().getCorCarro(), "amarelo");

//1 classe de equivalência: retorna null

Carro carro2 = estacionamento.buscarCarroPorPlaca("DSA7878");

assertEquals(carro2, null);

workspace - Java - estacionamento/src/Teste.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer JUnit

Finished after 0,017 seconds

Runs: 1/1 Errors: 0 Failures: 0

Teste [Runner: JUnit] (0,001s)

Failure Trace

Carro.java Especificacao.java Estacionamento.java Teste.java

```
1 import static org.junit.Assert.*;
2
3 import org.junit.Test;
4
5 public class Teste {
6
7     @Test
8     public void test() {
9
10
11         Estacionamento estacionamento = new Estacionamento();
12
13         //Testes de Classes de Equivalencia para o método cadastrarCarro
14
15         //1 classe de equivalência: 0 carros (lista vazia)
16
17         assertEquals(estacionamento.getCarros().size(), 0);
18
19         //1 classe de equivalência: 1 carro (guardando 1 objeto)
20
21         estacionamento.cadastrarCarro(new Carro("AAA1111", new Especificacao("VW", "fusca", "verde")));
22
23         assertEquals(estacionamento.getCarros().size(), 1);
24
25         //1 classe de equivalência: n carros (utilizando a lista para guardar n carros)
26
27         estacionamento.cadastrarCarro(new Carro("BBB1111", new Especificacao("VW", "fusca", "amarelo")));
28         estacionamento.cadastrarCarro(new Carro("CCC1111", new Especificacao("VW", "variant", "azul")));
29
30         assertEquals(estacionamento.getCarros().size(), 3);
31
32     }
33 }
34
35 }
36
```

Connect Mylyn

Connect to your t
ALM tools or crea
task.

Teste

test()

Problems Javadoc Declaration Console

<terminated> Teste [JUnit] C:\Program Files\Java\jre1.8.0_131\bin\javaw.exe (2 de abr de 2018 20:15:26)

Picked up _JAVA_OPTIONS: -Xms256m -Xmx512m

Writable Smart Insert 20 : 39

Iniciar

Cido de Vida do Dese...

workspace - Java - ...

Calculadora

20:17
02/04/2018

Atividade de Teste

- Com relação à atividade de teste em si, conseguimos validar os métodos “cadastrarCarro”, “buscarCarroPorEspecificacao” e “buscarCarroPorPlaca” de acordo com a técnica de “Classes de Equivalência”
- Com relação ao Ciclo de Vida como um todo, o teste automatizado me ajudou (tanto com relação à custo quanto a tempo) a iterar entre as etapas do ciclo de vida (ou seja, consigo voltar na prática para atividades anteriores que posso ter errado) e principalmente eu consigo alterar código (Atividade de Manutenção) por exemplo colocando novos requisitos e validar se esse acréscimo manteve o software funcionando corretamente ou não.
- O teste não serve apenas para validar a funcionalidade, mas também para auxiliar na atividade de manutenção (que faz parte do Ciclo de Vida).
- Refatoração de código só é possível com boas baterias de testes automatizados.

20/10

Teste: Classes de Equivalência

Exemplo: "Login"

	Entrada	Saída
1	nome e senha corretos	Login Validado
2	nome correto senha errada	Login Inválido
3	nome errado senha correta	Login Inválido
4	nome e senha errados	Login Inválido

No JUnit você faria pelo menos estes 4 testes unitários considerando o método de "login"

Para valores numéricos você pode usar a técnica de teste de análise de valor limite (já vista em aula)

Concluindo o Desenvolvimento da Arquitetura

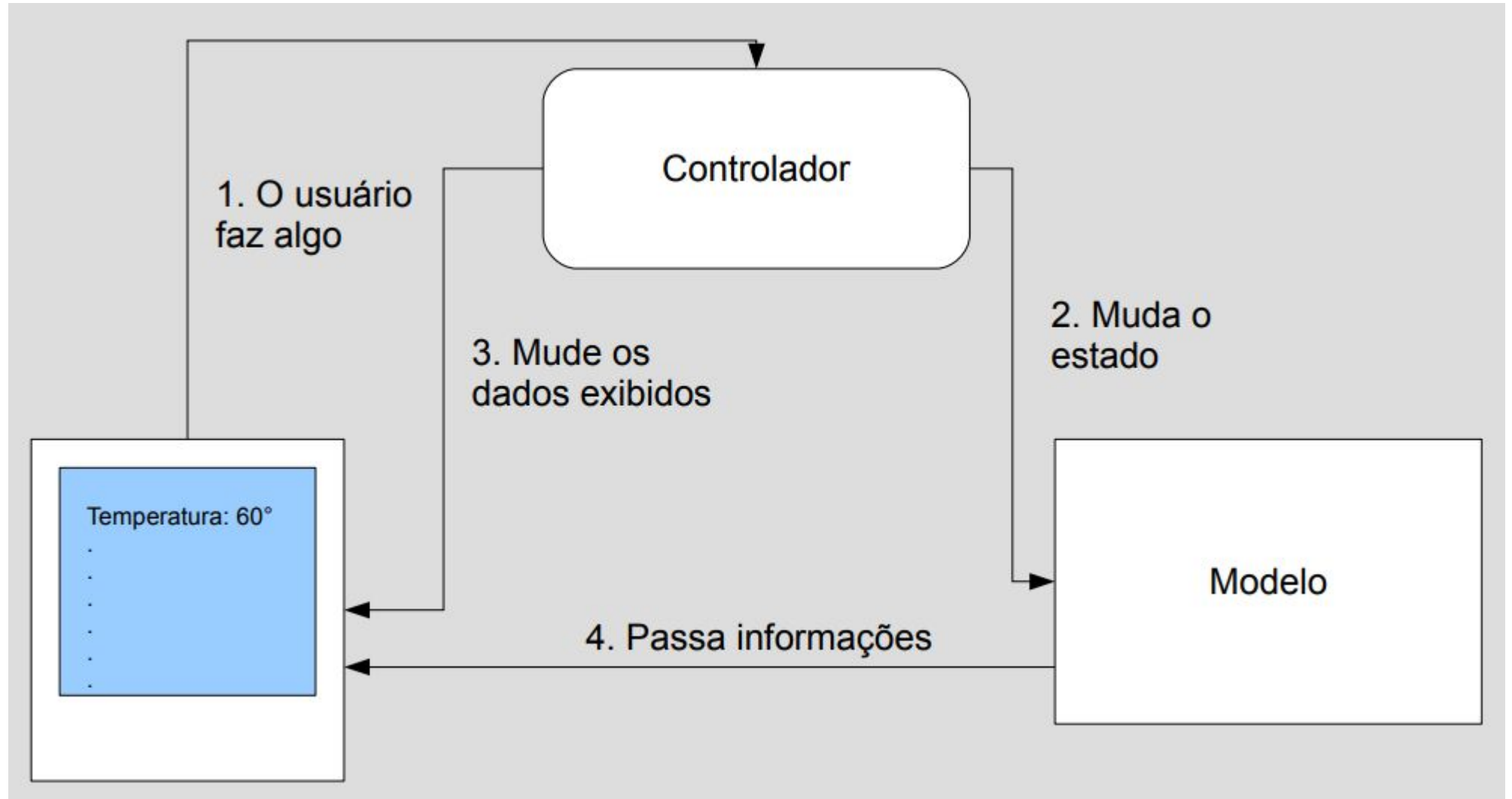
Terminamos a implementação e os testes do “Model” da nossa arquitetura.

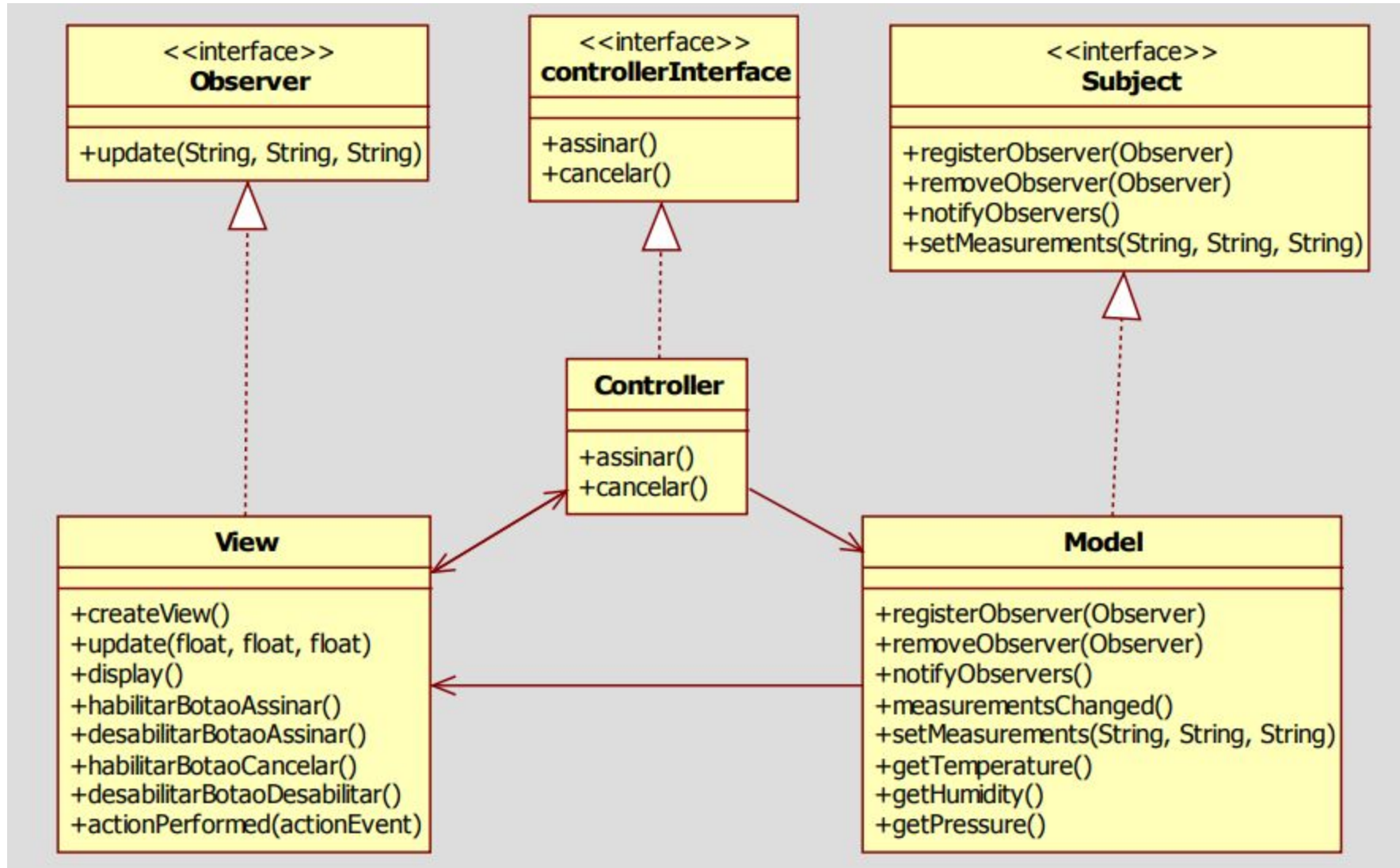
Você pode revisar tudo aqui: <https://drive.google.com/file/d/14Ag8I7cmzkdYyg53iVLRs1O26Sz-7kPe/view?usp=sharing>

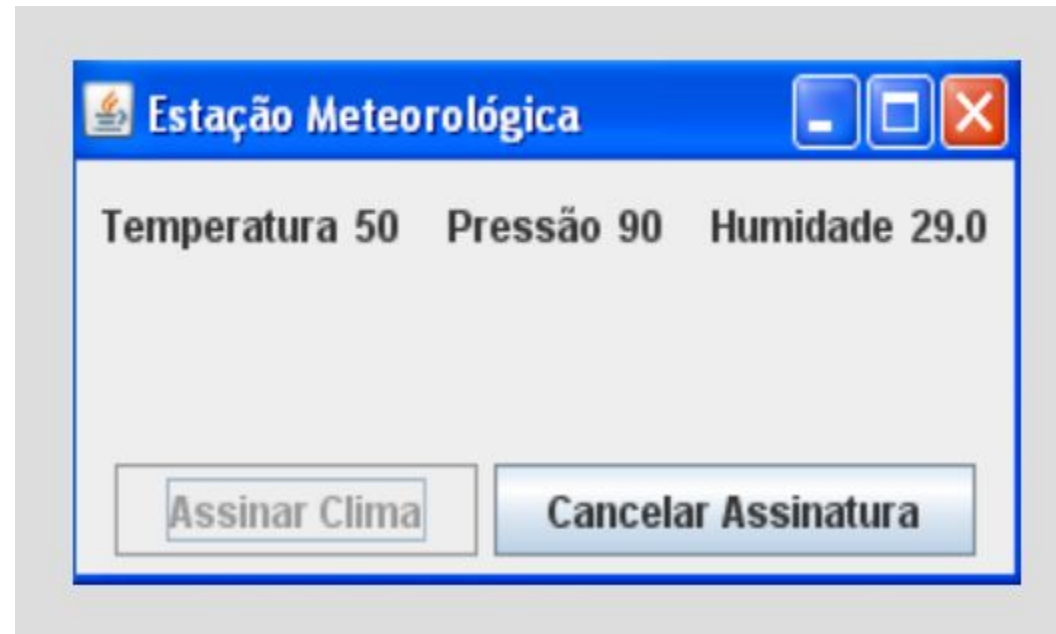
O projeto completo está aqui:

<https://github.com/giulianobertoti/projetointegrador/tree/master/SoftwareArchitecture>

<https://abseil.io/resources/swe-book>







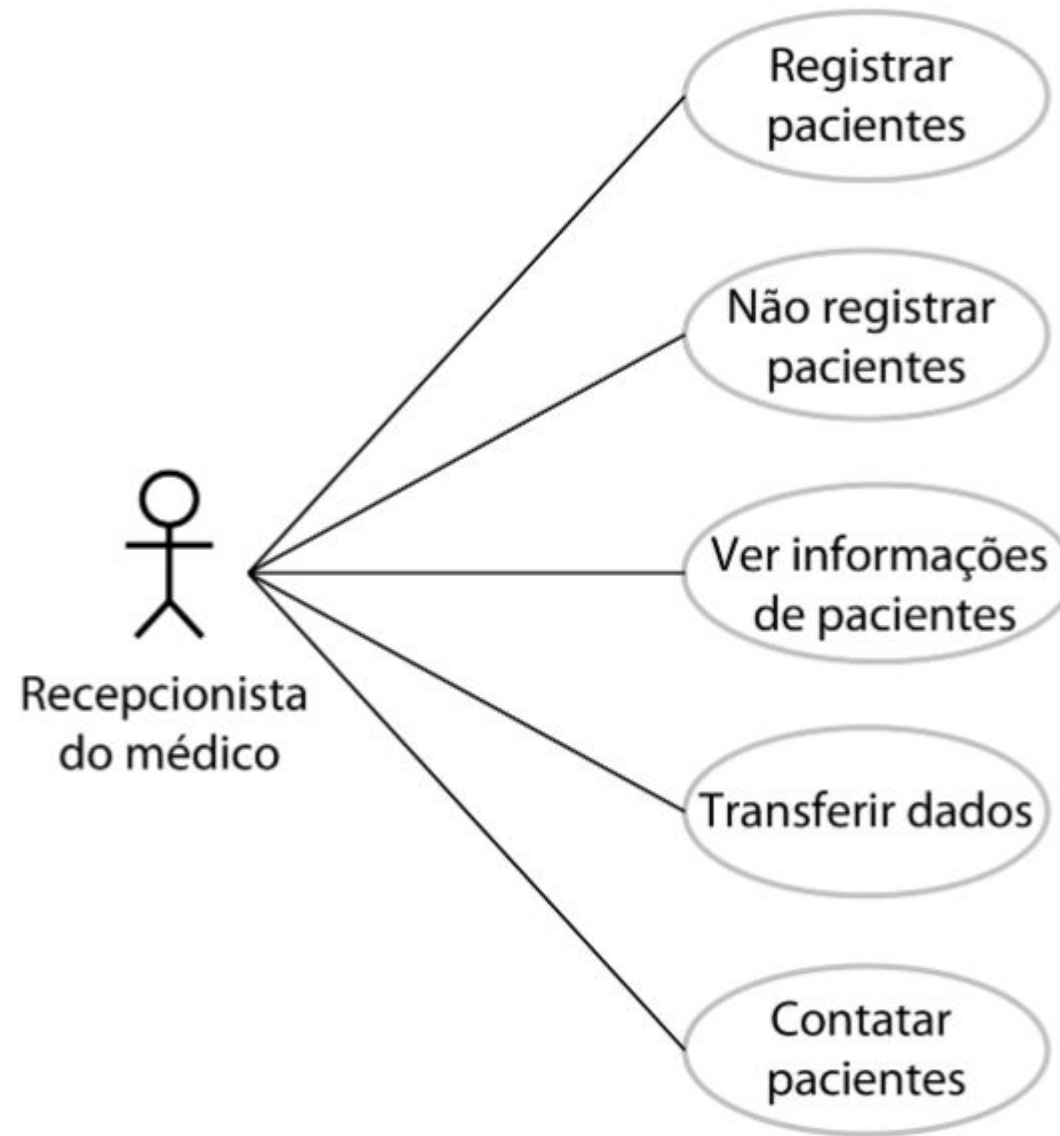
https://github.com/giulianobertoti/DesignPatternsJava/tree/master/MVC_desktop

Material adicional

(Sommerville 9ª edição)

Princípio ou prática	Descrição
Planejamento incremental	Os requisitos são gravados em cartões de história e as histórias que serão incluídas em um release são determinadas pelo tempo disponível e sua relativa prioridade. Os desenvolvedores dividem essas histórias em 'Tarefas'. Veja os quadros 3.1 e 3.2.
Pequenos releases	Em primeiro lugar, desenvolve-se um conjunto mínimo de funcionalidades útil, que fornece o valor do negócio. Releases do sistema são frequentes e gradualmente adicionam funcionalidade ao primeiro release.
Projeto simples	Cada projeto é realizado para atender às necessidades atuais, e nada mais.
Desenvolvimento test-first	Um framework de testes iniciais automatizados é usado para escrever os testes para uma nova funcionalidade antes que a funcionalidade em si seja implementada.
Refatoração	Todos os desenvolvedores devem refatorar o código continuamente assim que encontrarem melhorias de código. Isso mantém o código simples e manutenível.
Programação em pares	Os desenvolvedores trabalham em pares, verificando o trabalho dos outros e prestando apoio para um bom trabalho sempre.
Propriedade coletiva	Os pares de desenvolvedores trabalham em todas as áreas do sistema, de modo que não se desenvolvam ilhas de expertise. Todos os conhecimentos e todos os desenvolvedores assumem responsabilidade por todo o código. Qualquer um pode mudar qualquer coisa.
Integração contínua	Assim que o trabalho em uma tarefa é concluído, ele é integrado ao sistema como um todo. Após essa integração, todos os testes de unidade do sistema devem passar.
Ritmo sustentável	Grandes quantidades de horas-extra não são consideradas aceitáveis, pois o resultado final, muitas vezes, é a redução da qualidade do código e da produtividade a médio prazo.
Cliente no local	Um representante do usuário final do sistema (o cliente) deve estar disponível todo o tempo à equipe de XP. Em um processo de Extreme Programming, o cliente é um membro da equipe de desenvolvimento e é responsável por levar a ela os requisitos de sistema para implementação.

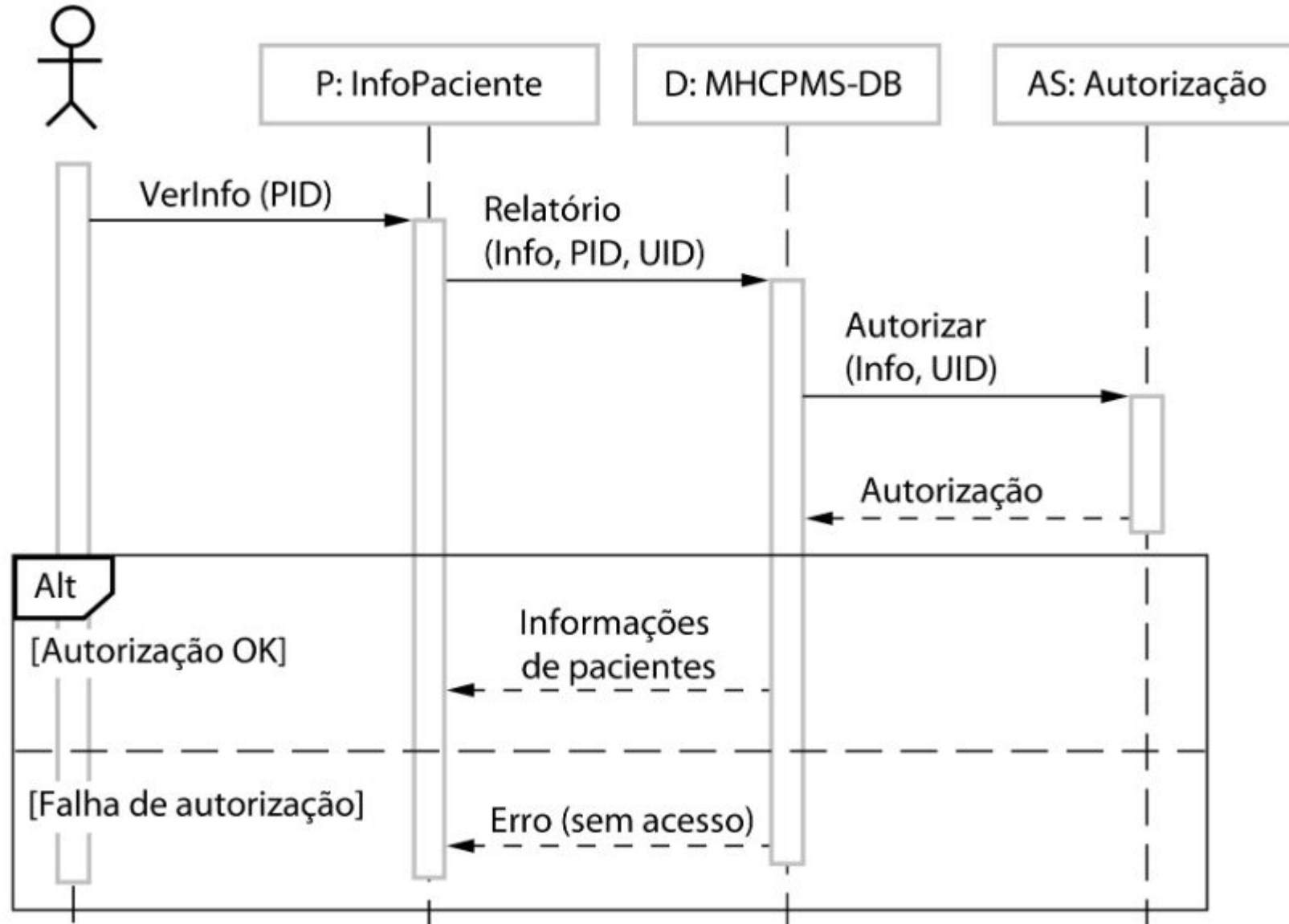
(Sommerville 9ª edição)



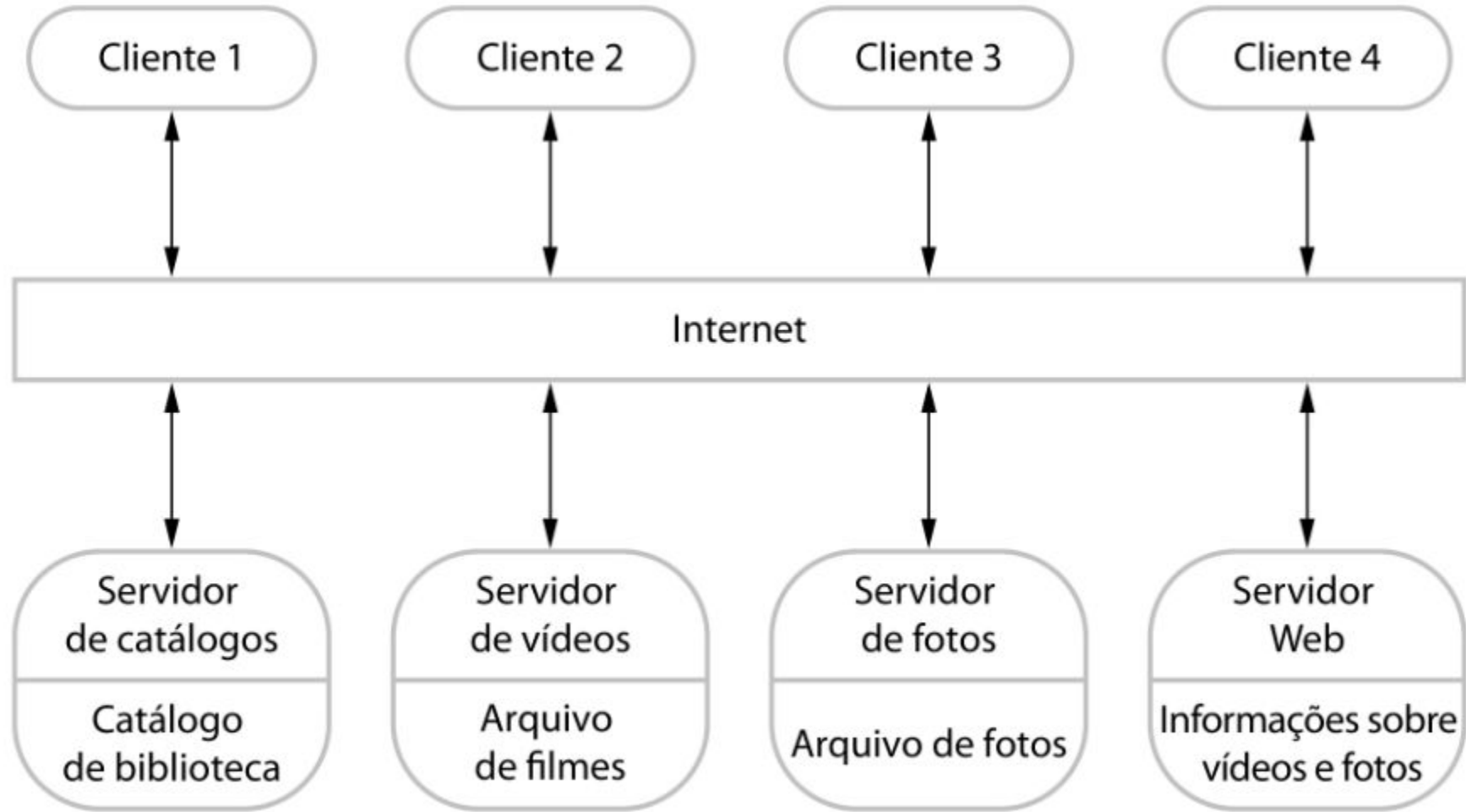
(Sommerville 9ª edição)

Veja que o diagrama de casos de uso do estacionamento está bem mais completo que este do livro do Sommerville...

Recepcionista do médico



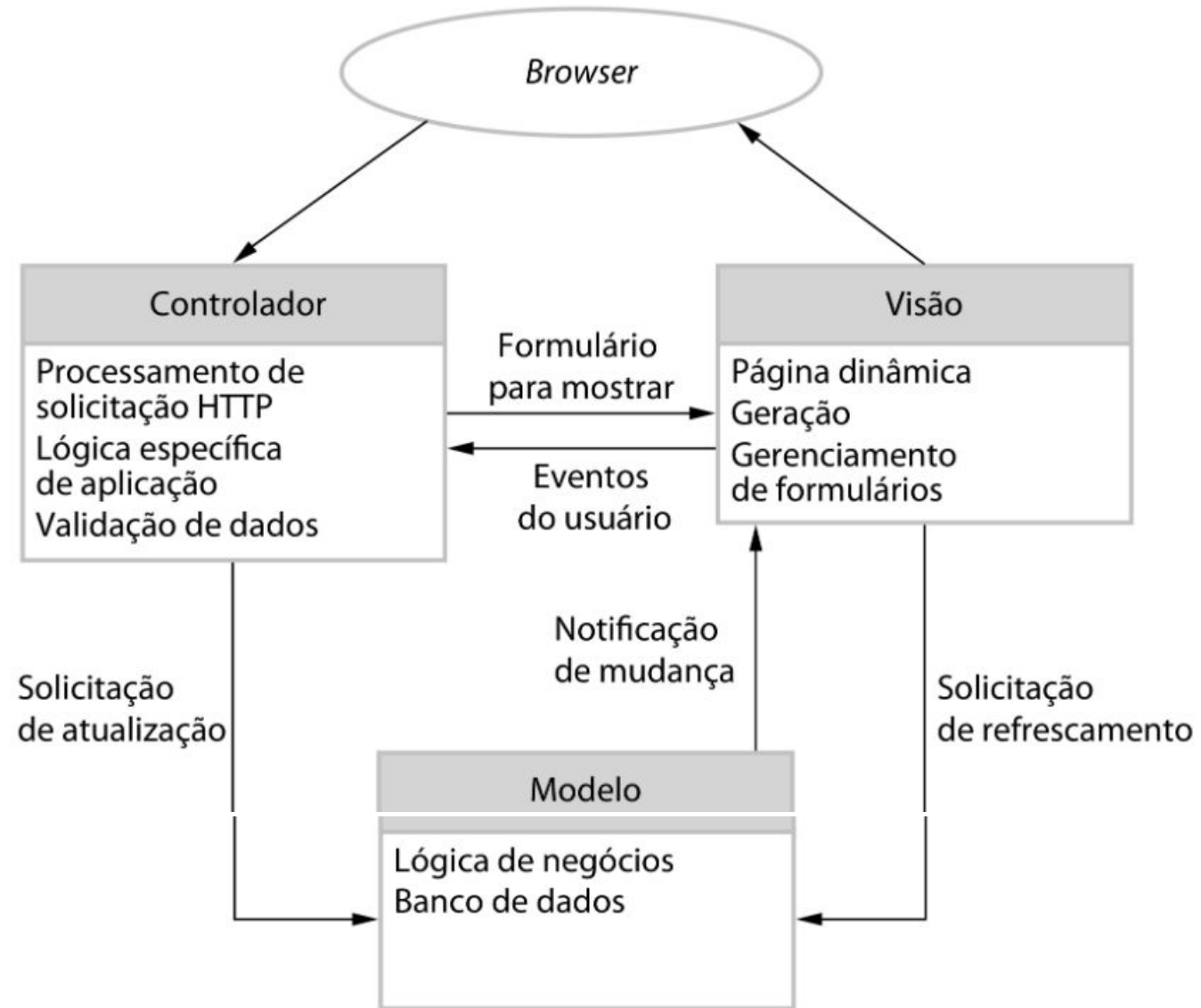
(Sommerville 9ª edição)



(Sommerville 9ª edição)

Nome	MVC (Modelo-Visão-Controlador)
Descrição	Separa a apresentação e a interação dos dados do sistema. O sistema é estruturado em três componentes lógicos que interagem entre si. O componente Modelo gerencia o sistema de dados e as operações associadas a esses dados. O componente Visão define e gerencia como os dados são apresentados ao usuário. O componente Controlador gerencia a interação do usuário (por exemplo, teclas, cliques do mouse etc.) e passa essas interações para a Visão e o Modelo. Veja a Figura 6.2.
Exemplo	A Figura 6.3 mostra a arquitetura de um sistema aplicativo baseado na Internet, organizado pelo uso do padrão MVC.
Quando é usado	É usado quando existem várias maneiras de se visualizar e interagir com dados. Também quando são desconhecidos os futuros requisitos de interação e apresentação de dados.
Vantagens	Permite que os dados sejam alterados de forma independente de sua representação, e vice-versa. Apoia a apresentação dos mesmos dados de maneiras diferentes, com as alterações feitas em uma representação aparecendo em todas elas.
Desvantagens	Quando o modelo de dados e as interações são simples, pode envolver código adicional e complexidade de código.

(Sommerville 9ª edição)



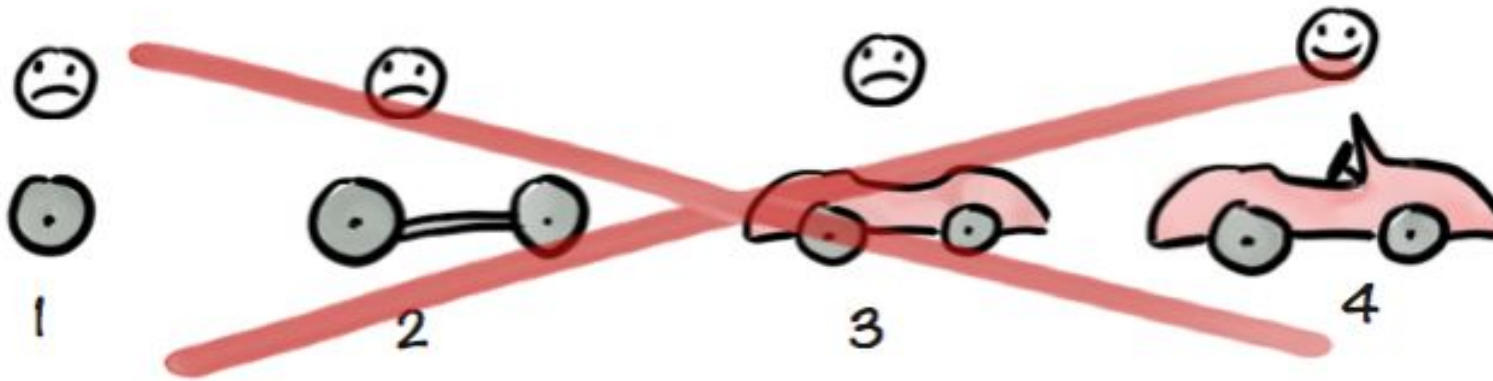
(Sommerville 9ª edição)

Problema	Explicação
Maiores custos de manutenção	Se o código-fonte de um sistema ou componentes de software reusáveis não estiverem disponíveis, os custos de manutenção podem ser maiores, pois os elementos reusados do sistema podem tornar-se cada vez mais incompatíveis com as alterações do sistema.
Falta de ferramentas de suporte	Algumas ferramentas de software não suportam o desenvolvimento com reúso. Pode ser difícil ou impossível integrar essas ferramentas com um sistema de biblioteca de componentes. O processo de software assumido por essas ferramentas pode não considerar o reúso. Isso é particularmente verdadeiro para as ferramentas que oferecem suporte a engenharia de sistemas embutidos, exceto para as ferramentas de desenvolvimento orientado a objetos.
Síndrome de 'não-inventado-aqui'	Alguns engenheiros de software preferem reescrever componentes, pois acreditam poder melhorá-los. Isso tem a ver, parcialmente, com aumentar a confiança e, parcialmente, com o fato de que escrever softwares originais é considerado mais desafiador do que reusar softwares de outras pessoas.
Criação, manutenção e uso de uma biblioteca de componentes	Preencher uma biblioteca de componentes reusáveis e garantir que desenvolvedores de software possam utilizar essa biblioteca podem ser ações caras. Processos de desenvolvimento precisam ser adaptados para garantir que a biblioteca seja usada.
Encontrar, compreender e adaptar os componentes reusáveis	Componentes de software precisam ser descobertos em uma biblioteca, compreendidos e, às vezes, adaptados para trabalhar em um novo ambiente. Os engenheiros precisam estar confiantes de que encontrarão, na biblioteca, um componente, antes de incluírem a pesquisa de componente como parte de seu processo normal de desenvolvimento.

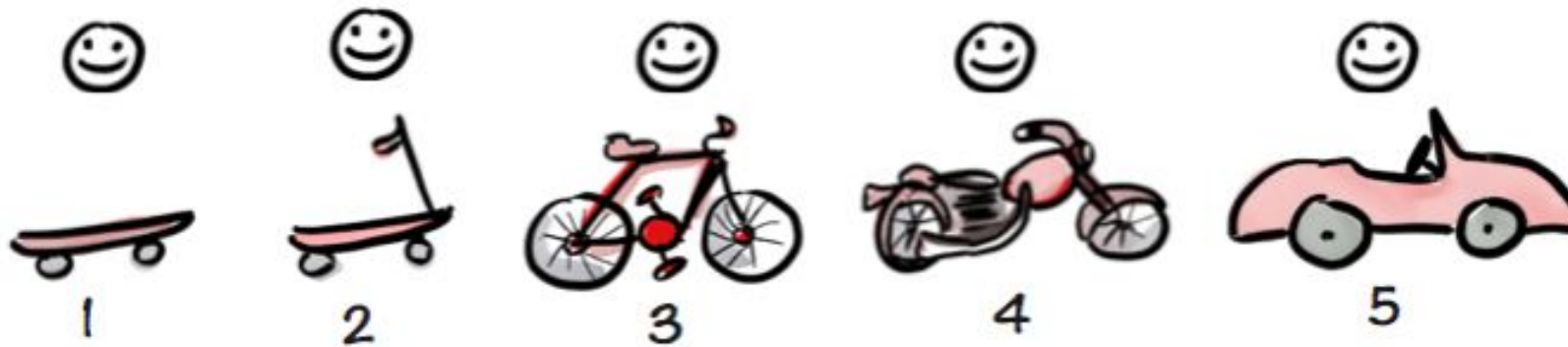
(Sommerville 9ª edição)

MVP

Not like this....

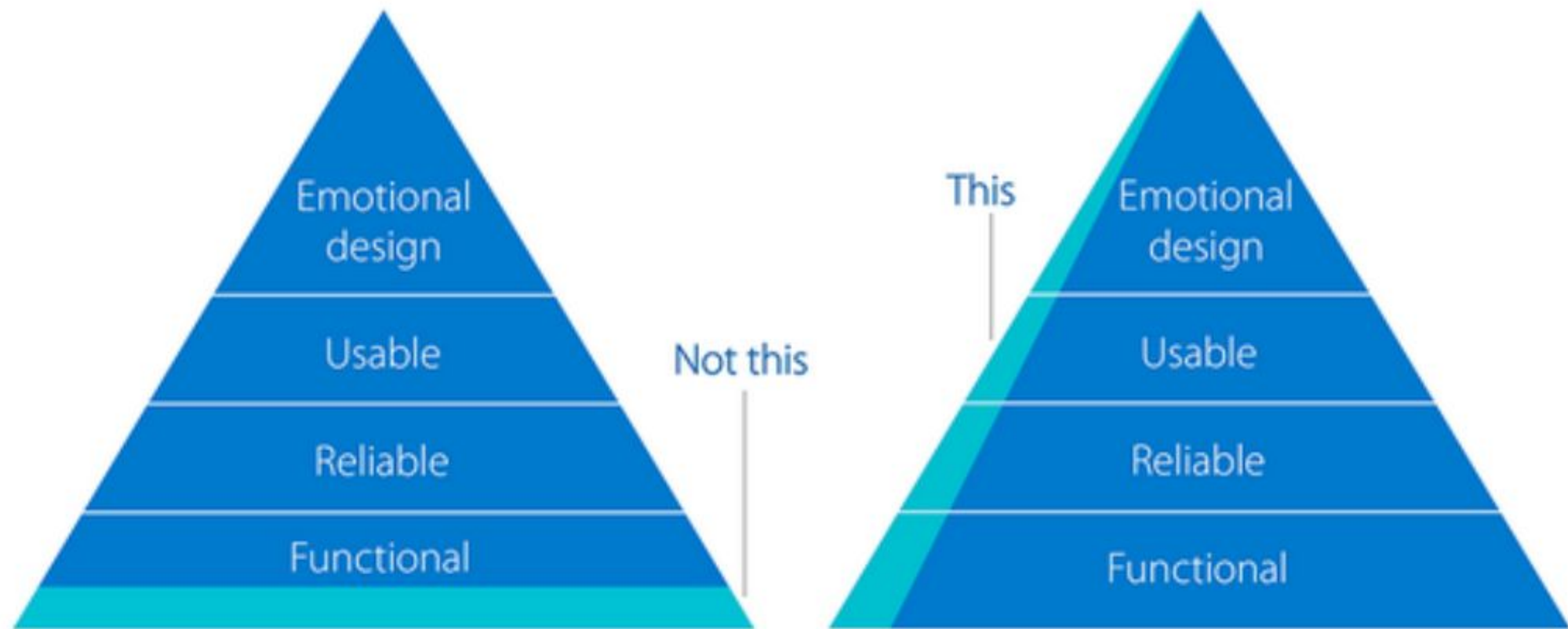


Like this!



The best first step towards building a complex end-to-end system is to build a basic end-to-end system -- not to build a submodule of what you think the complex system should look like (**François Chollet, Google**)

Minimum Viable Product



De que adianta um Hotel ter uma sala de checkin maravilhosa se ele não tem quartos?

De que adianta um Sistema ter um login sofisticado se quando o usuário se loga ele não pode de fato resolver seus problemas?

We are not here to build Products.
We are here to solve Customer Problems.

"An MVP is not about building 3 features out of a total of 10 features.

It's about building a version of your product that can help you learn the most about the market and customers and validate your idea."

Pergunte sempre!