

Fluxos de Modelagem com UML

Prof. Dr. Giovani Gracioli
giovani@lisha.ufsc.br

**ROTA2030
FUNDEP**

- Qualquer editor UML
 - dia
 - Eclipse e suporte a UML-MARTE no papyrus
 - Qualquer outro editor
 - Se não possuir nenhum, papel

- Introdução
- Fluxo de projeto Modelo C4
- Fluxo de projeto usando a linguagem C e UML
 - Conceituação
 - Exemplos
 - Exercícios
- Fluxo de projeto usando a linguagem C++ e UML
 - Conceituação
 - Exemplos
 - Exercícios

- Introdução
- Fluxo de projeto Modelo C4
- Fluxo de projeto usando a linguagem C e UML
 - Conceituação
 - Exemplos
 - Exercícios
- Fluxo de projeto usando a linguagem C++ e UML
 - Conceituação
 - Exemplos
 - Exercícios
- Integração com SOTR e Drivers
 - Visão geral
 - Exemplos e exercícios

- Vimos que a UML possui 13 diagramas
 - Estudamos alguns com maior profundidade
- Alguns deles representam a mesma informação, com mais ou menos detalhes
- O uso de todos os diagramas é impraticável por qualquer equipe de desenvolvimento
 - Aumenta a complexidade
 - Nenhum projetista/desenvolvedor irá ser especialista ou entender os 13 diagramas de forma a ajudar com o sistema a ser desenvolvido

- Neste conjunto de slides iremos estudar como utilizar apenas um subconjunto de diagramas da UML para realizar um projeto de software
- Começaremos com o modelo C4
- Na sequência estudaremos um fluxo de modelagem para a linguagem C
- E um fluxo de modelagem para a linguagem C++
- Finalizaremos com a integração com RTOS/drivers

- Introdução
- Fluxo de projeto Modelo C4
- Fluxo de projeto usando a linguagem C e UML
 - Conceituação
 - Exemplos
 - Exercícios
- Fluxo de projeto usando a linguagem C++ e UML
 - Conceituação
 - Exemplos
 - Exercícios

Modelo C4

- O Modelo C4 consiste em um método para descrever uma arquitetura de software
 - Criado por Simon Brown (consultor independente especialista em arquitetura de software) e inspirado no modelo 4+1 (visão lógica, processo, desenvolvimento, física + casos de uso)
- O modelo C4 foi proposto pela dificuldade em se ter uma forma comum para descrever a arquitetura de software
 - Embora exista UML, as vezes a linguagem pode ser muito ampla e nem todos a conhecem em uma equipe

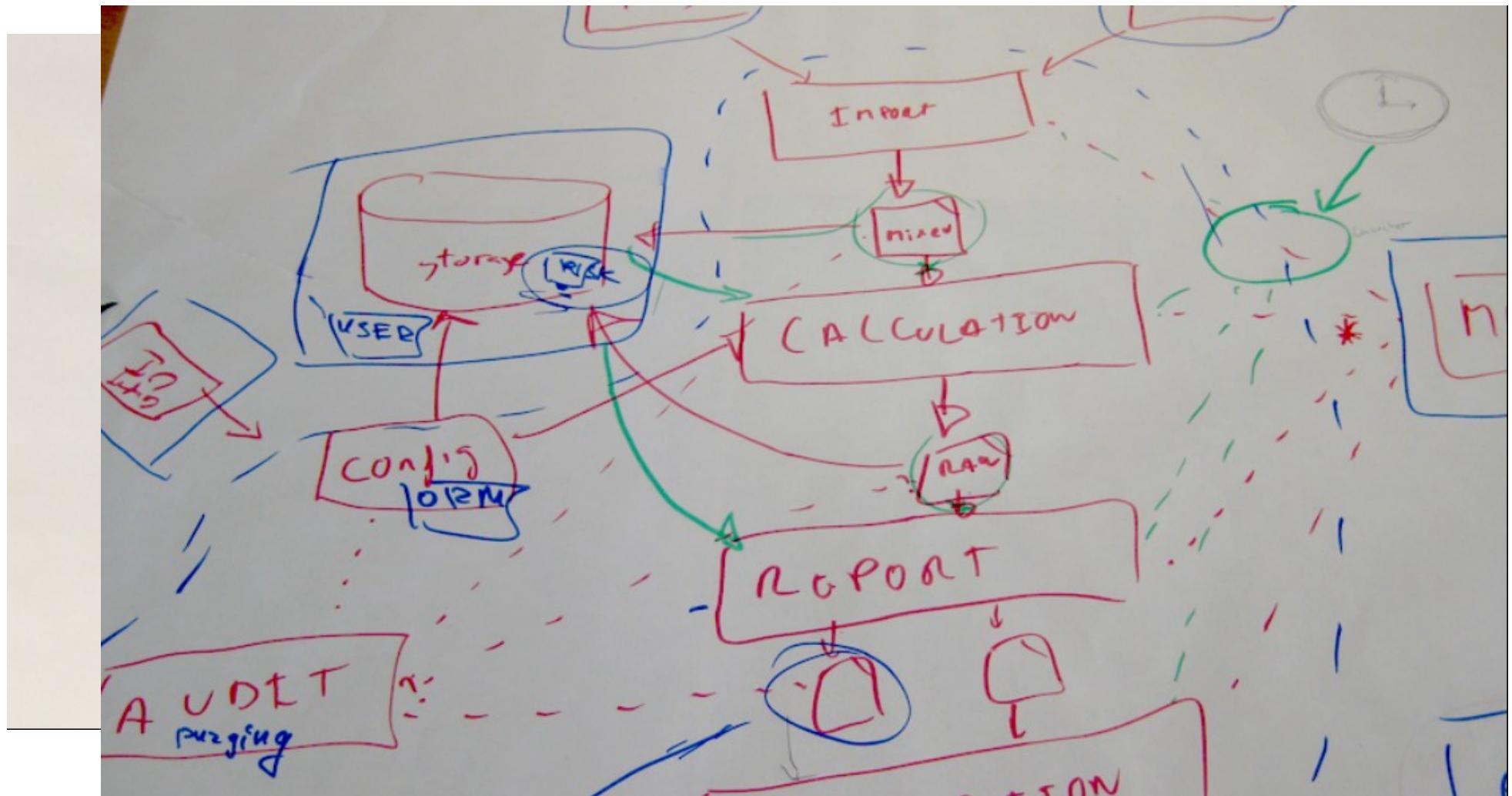
■ Estrutura

- A definição de software em termos de seus blocos e suas interações

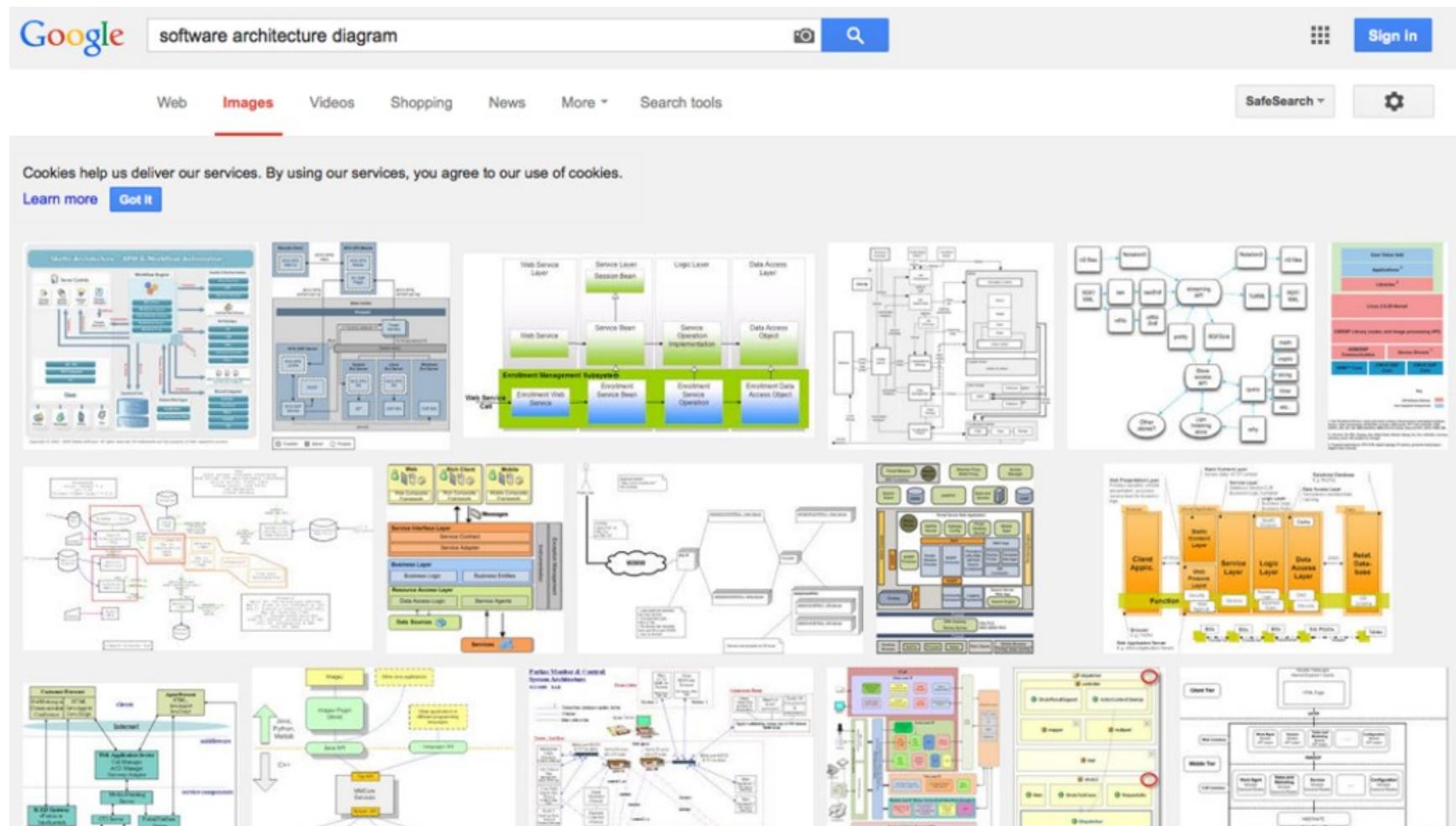
■ Visão

- O processo de “arquitetar” ou projetar
- Tomar decisões baseadas no objetivo do negócio, requisitos e restrições, sendo capaz de comunicar as decisões para a equipe

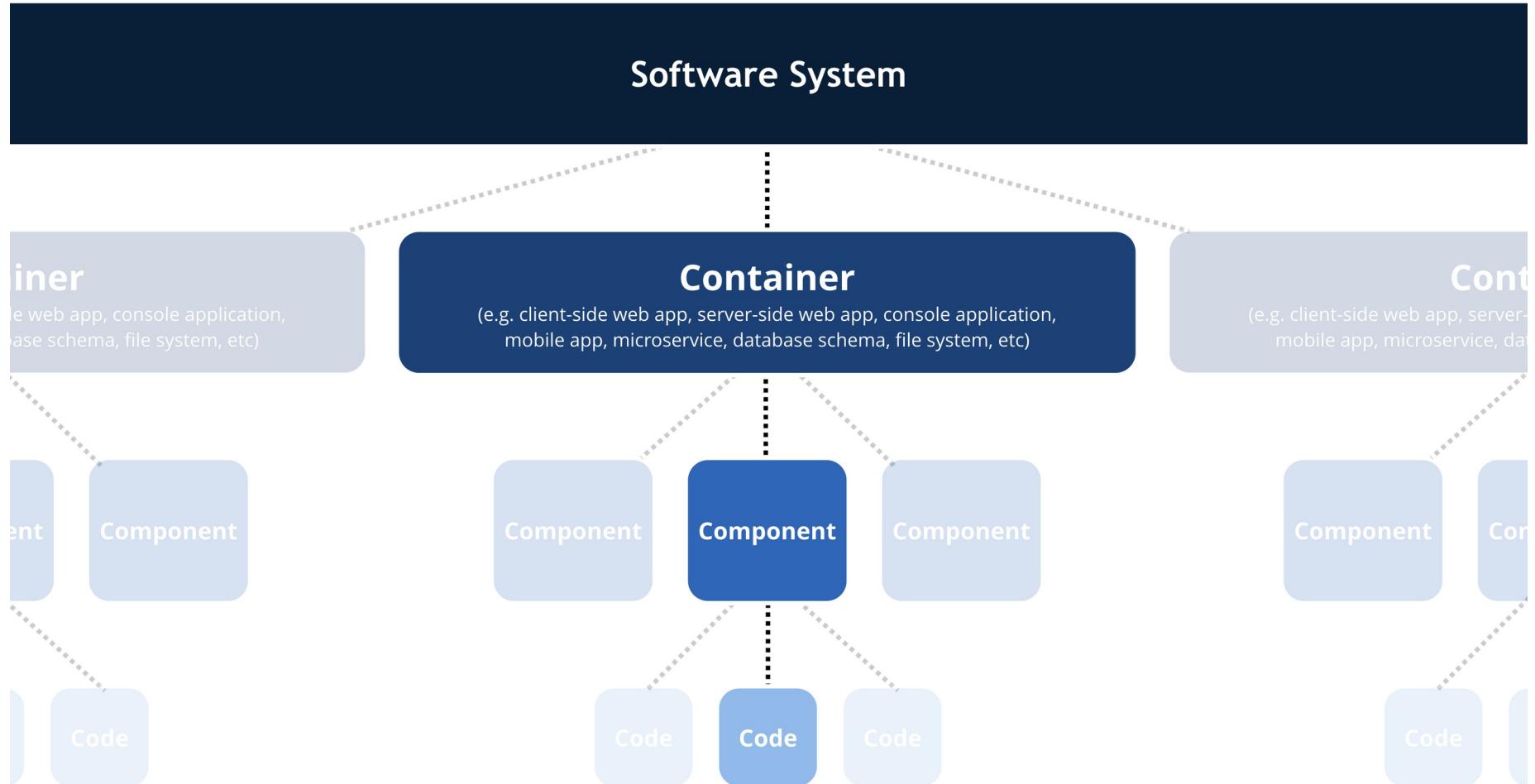
- Exemplos de modelos de arquitetura de software



- Google: mesmo problema (diferentes formas, formatos, linhas, etc)

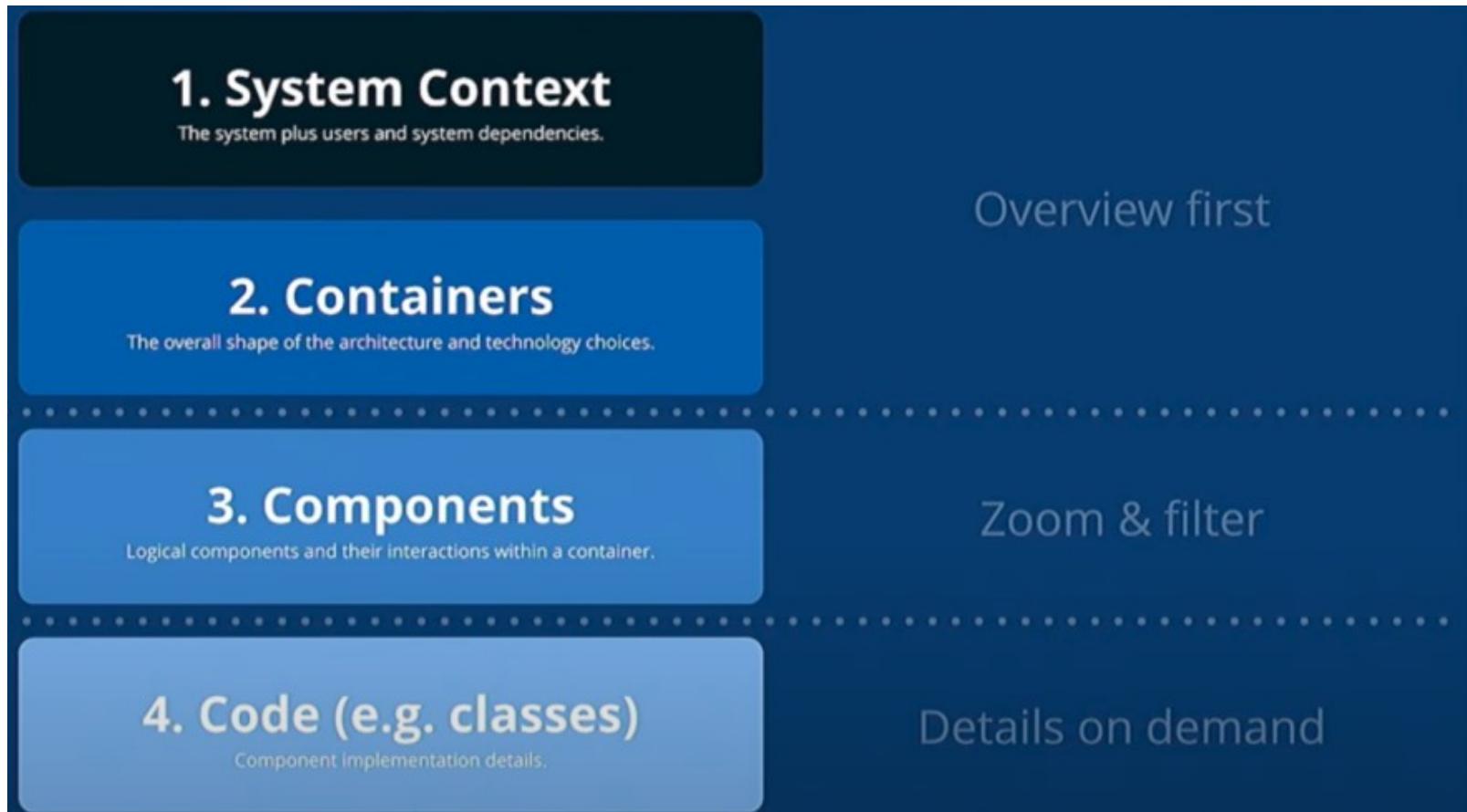


- Atingir os objetivos como equipe de uma maneira rápida exige **boa comunicação**
- Pessoas de diferentes áreas dentro da mesma equipe têm diferentes visões sobre o mesmo problema ou projeto que estão trabalhando
- Segundo C4, um conjunto de abstrações comum é mais importante que uma notação comum
 - Uma das razões que os desenvolvedores dizem para não usar UML é devido sua notação complicada

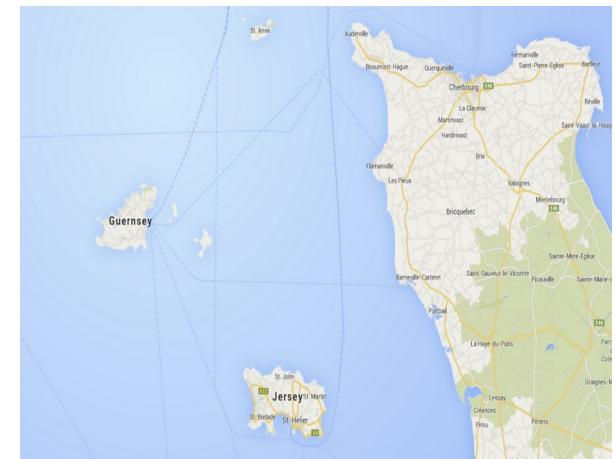
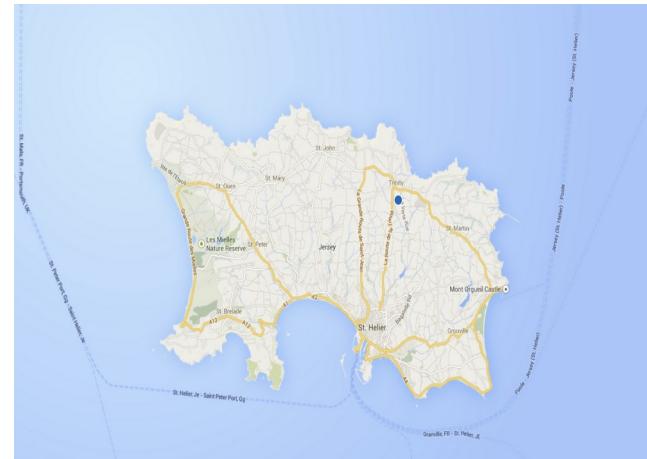


Modelo C4

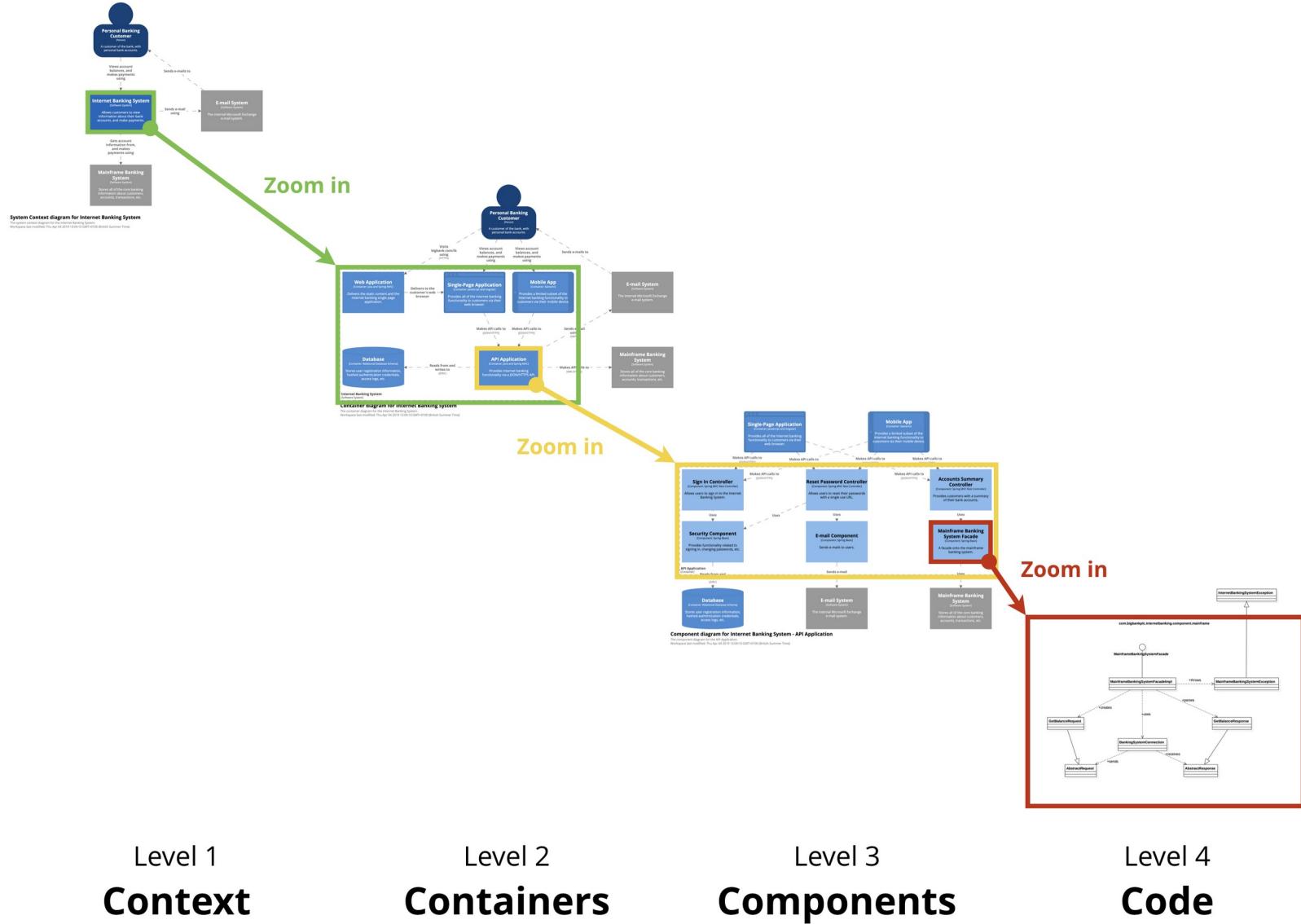
- São 4 **diagramas** que descrevem a **estrutura estática** do sistema de software
- Os diagramas formam um conjunto hierárquico da arquitetura de software



- Diagramas são mapas que ajudam os desenvolvedores a navegar em códigos grandes e complexos
- Diferentes níveis de detalhes, como zoom no google maps



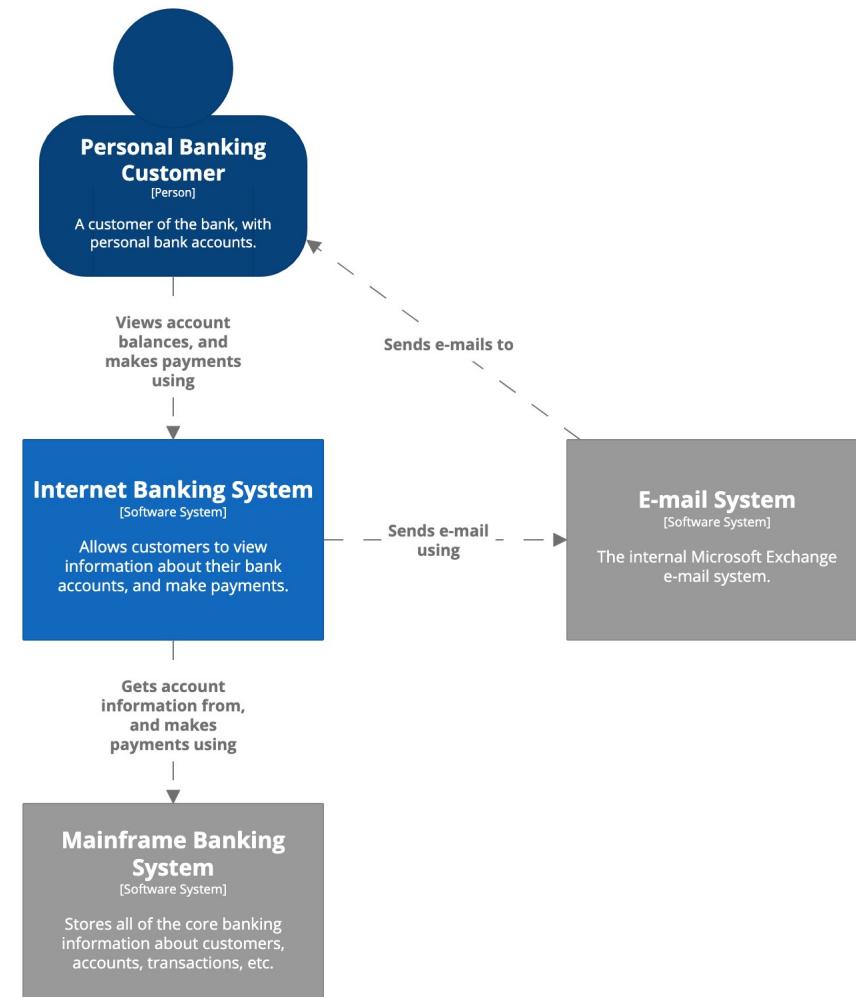
Analogia com mapas



■ Nível 1: Diagrama de Contexto do Sistema

- Diagrama de alto nível que mostra uma visão geral do sistema e interação com o mundo exterior
- Caixa no centro, rodeada pelos usuários e outros sistemas que interagem com o seu sistema
- Audiência: todos (técnicos e não técnicos)

■ Nível 1: Diagrama de Contexto do Sistema



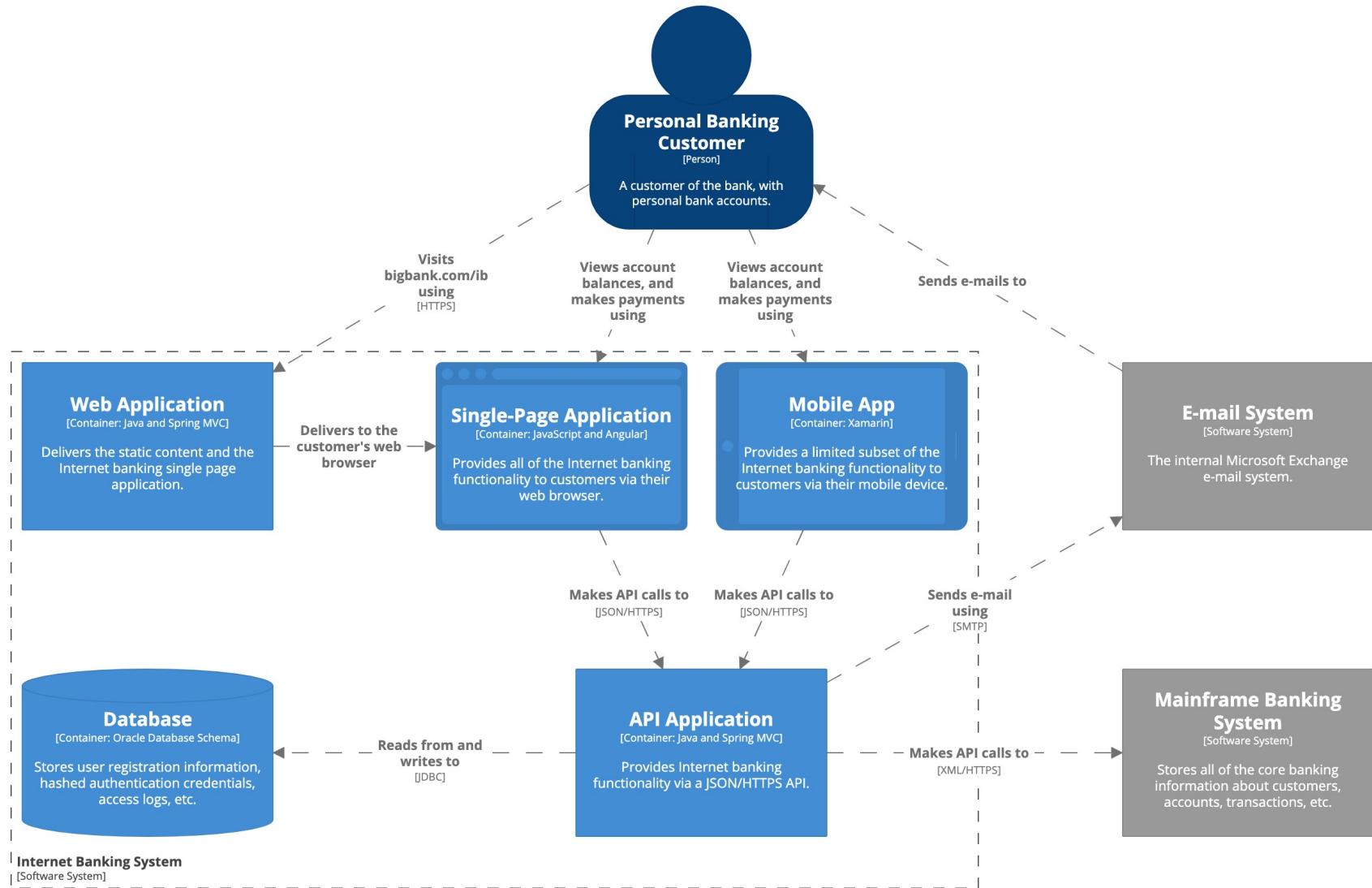
System Context diagram for Internet Banking System

The system context diagram for the Internet Banking System.
Workspace last modified: Wed Feb 05 2020 09:33:36 GMT+0100 (Central European Standard Time)

■ Nível 2: Diagrama de Container

- Essencialmente um Container dentro do modelo C4 é uma unidade de um processo em execução (aplicação)
- Mostra a forma em alto nível da arquitetura do software e como as responsabilidades são atribuídas
- Mostra também as principais escolhas tecnológicas e como os containers se comunicam
- Audiência: Pessoas técnicas da equipe ou fora da equipe

■ Nível 2: Diagrama de Container



Container diagram for Internet Banking System

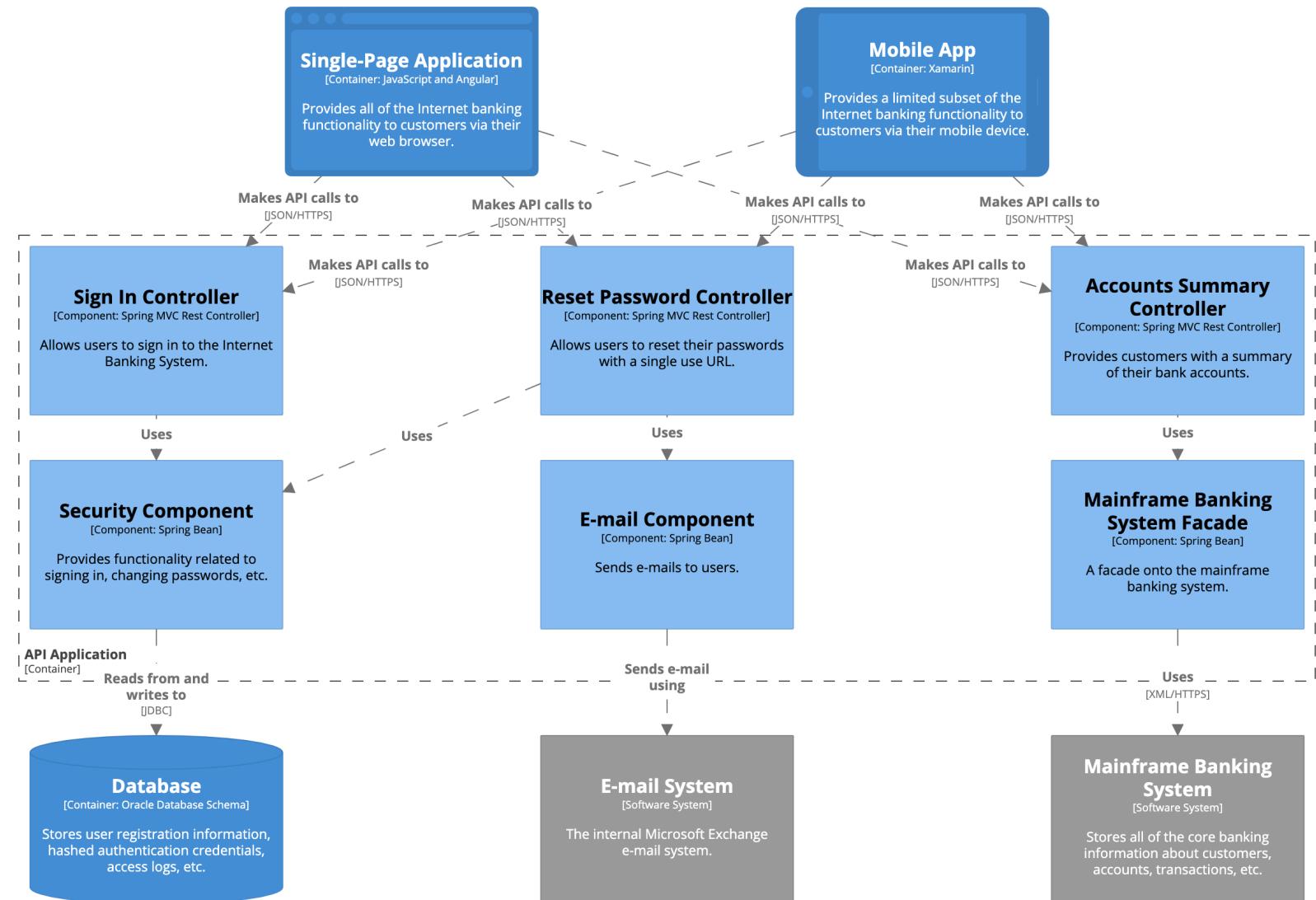
The container diagram for the Internet Banking System.

Workspace last modified: Wed Feb 05 2020 09:33:36 GMT+0100 (Central European Standard Time)

■ Nível 3: Diagrama de Componente

- Decompõe cada container para identificar os principais blocos estruturais e suas interações
- Mostra como um container é feito usando um número de componentes, quais são esses componentes, suas responsabilidades e detalhes tecnológicos e de implementação
- Audiência: arquitetos de software e desenvolvedores

■ Nível 3: Diagrama de Componente

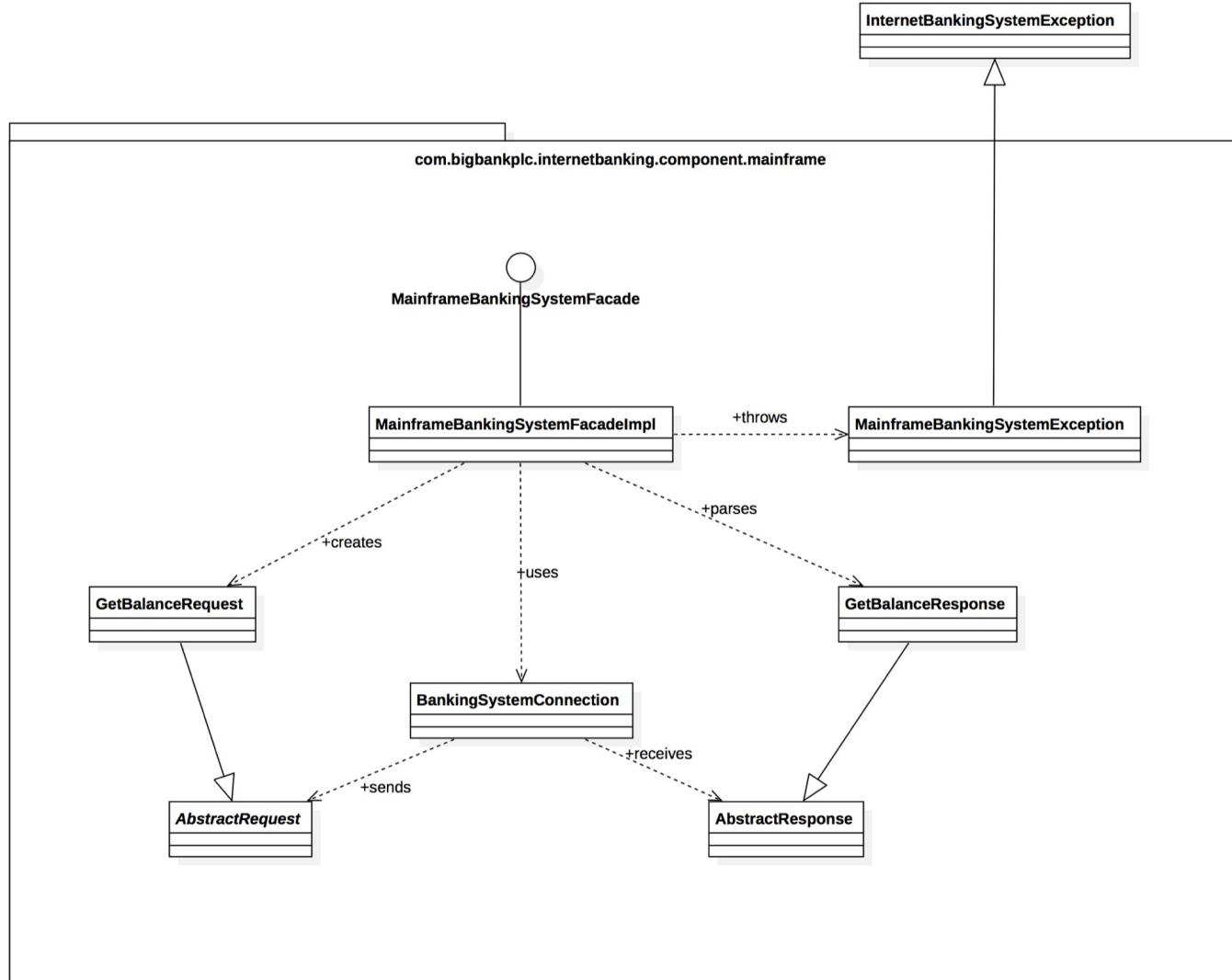


Modelo C4

■ Nível 4: Código

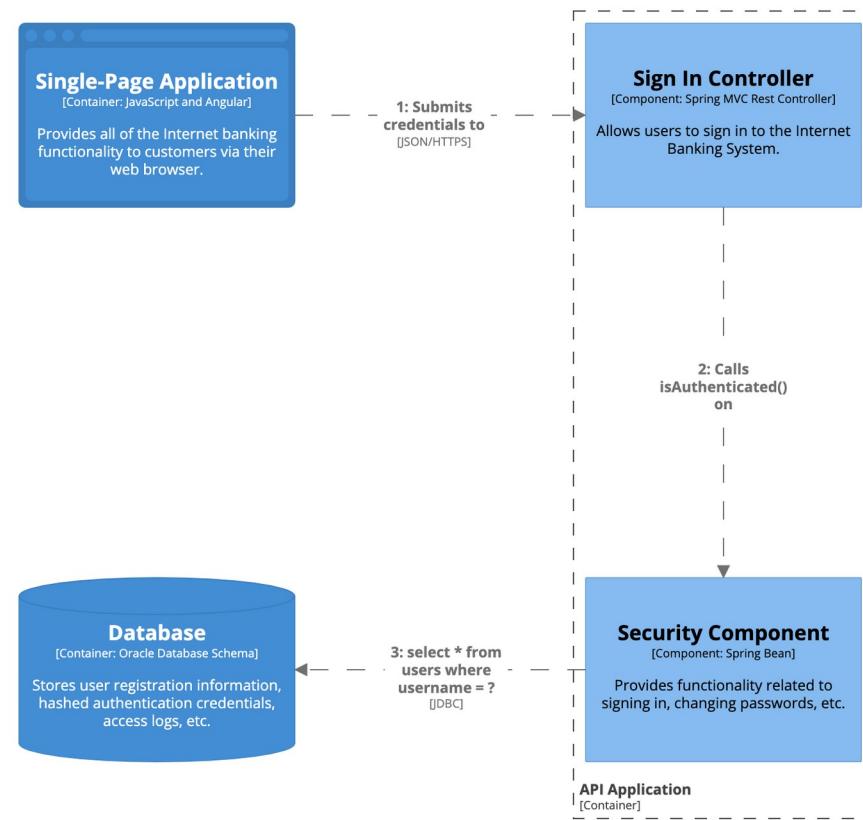
- Visão de cada componente, mostrando como são implementados
- Diagrama de classes UML
- Audiência: arquitetos de software e desenvolvedores

■ Nível 4: Código



- Mostram aspectos complementares
- **Diagrama de Paisagem do Sistema**
 - Mostra como o sistema de software se encaixa dentro da empresa
- **Diagrama Dinâmico**
 - Mostra como os elementos colaboram em tempo de execução para implementar um caso de uso
 - Baseado no diagrama de comunicação UML (parecido com o de sequência)
- **Diagrama de Deployment**
 - Mapeia os containers na infraestrutura
 - Baseado no diagrama UML de deployment

■ Ex. Diagrama Dinâmico



Dynamic diagram for API Application

Summarises how the sign in feature works in the single-page application.
Workspace last modified: Wed Feb 05 2020 09:33:36 GMT+0100 (Central European Standard Time)

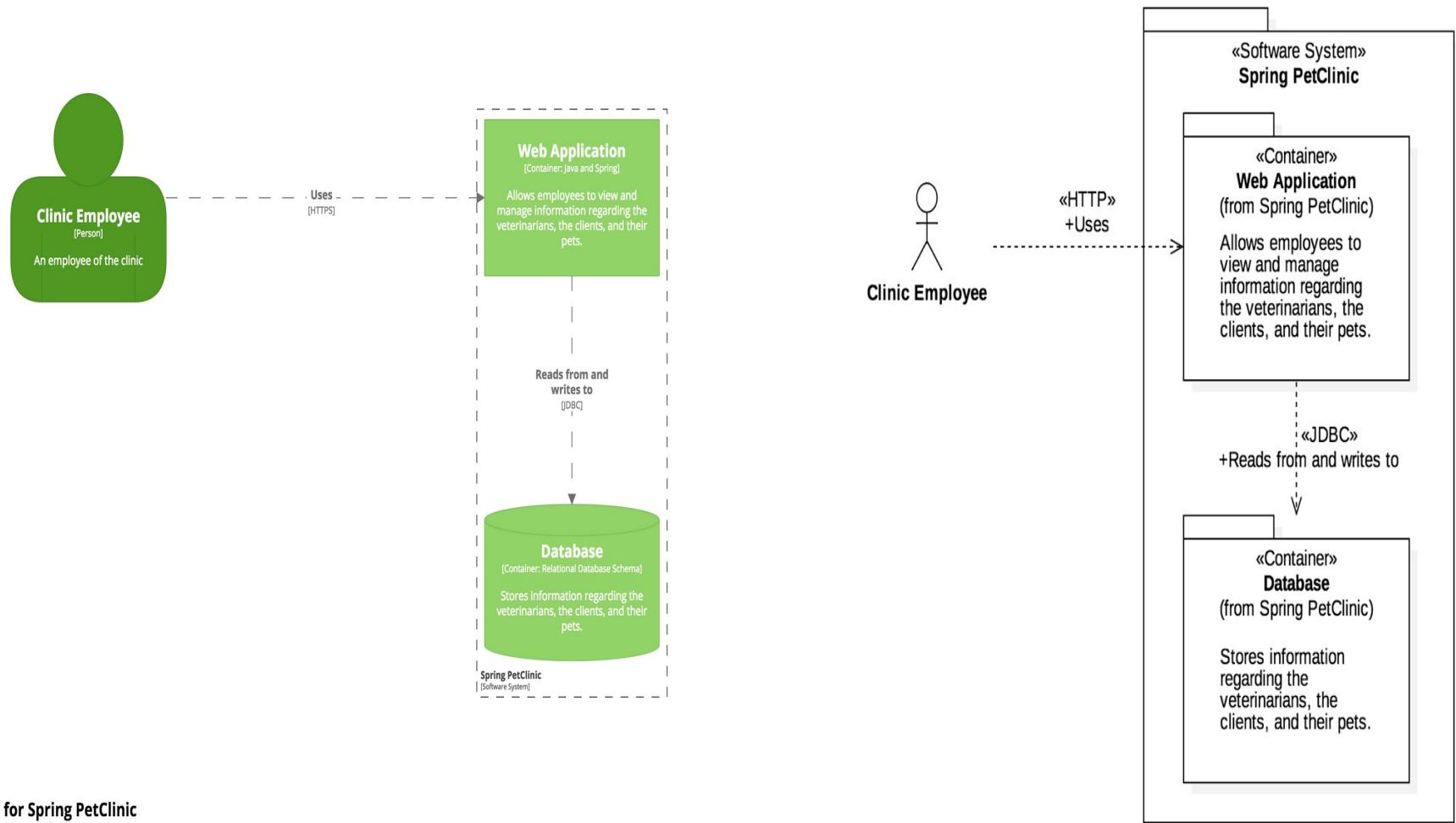
Diagrams Suplementares

■ Ex. Diagrama de Deployment

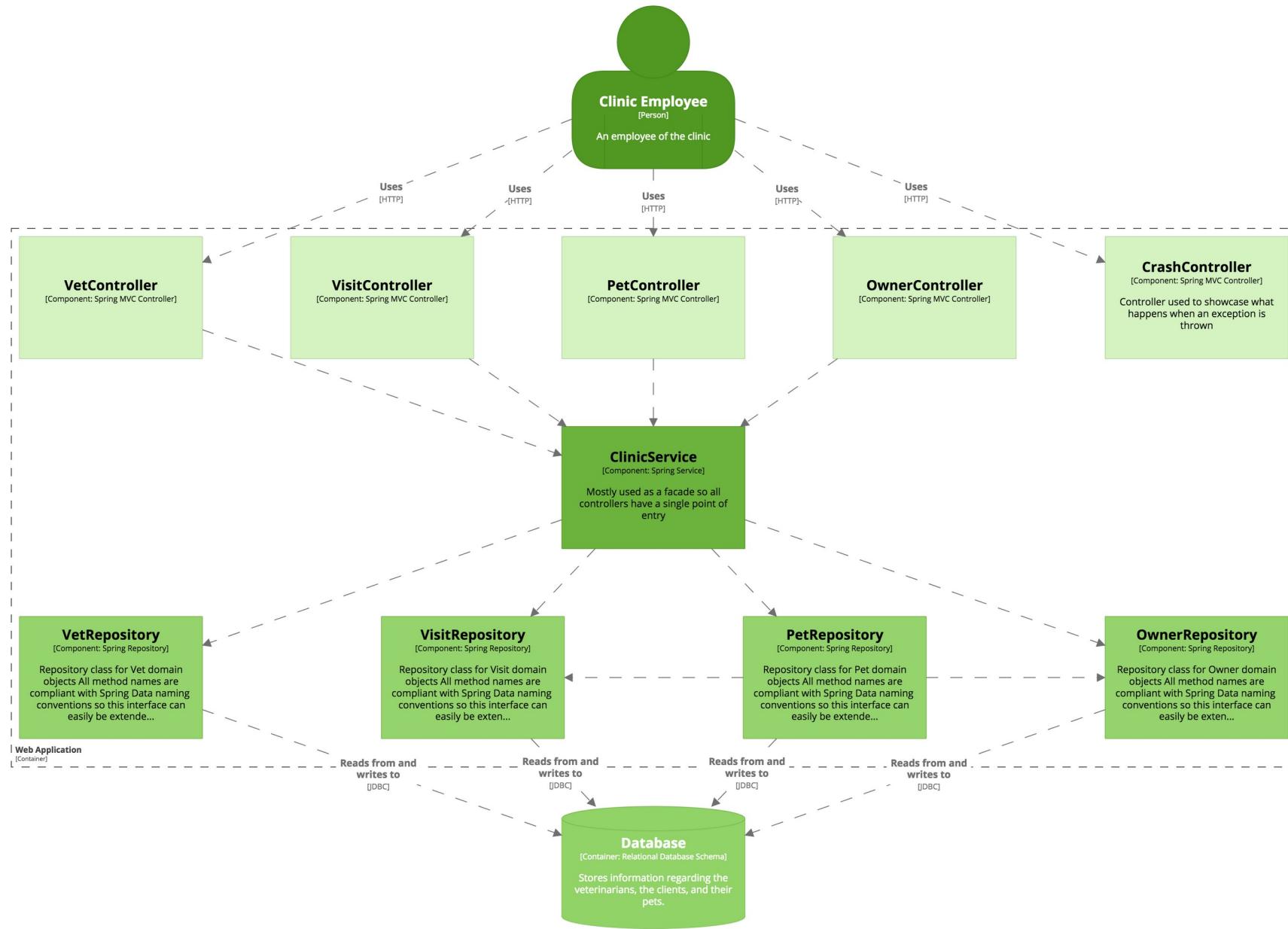


- Não há uma notação própria
- C4 e UML
 - Os principais diagramas podem ser descritos usando UML (pacotes, componentes e estereótipos)





Notação no Modelo C4

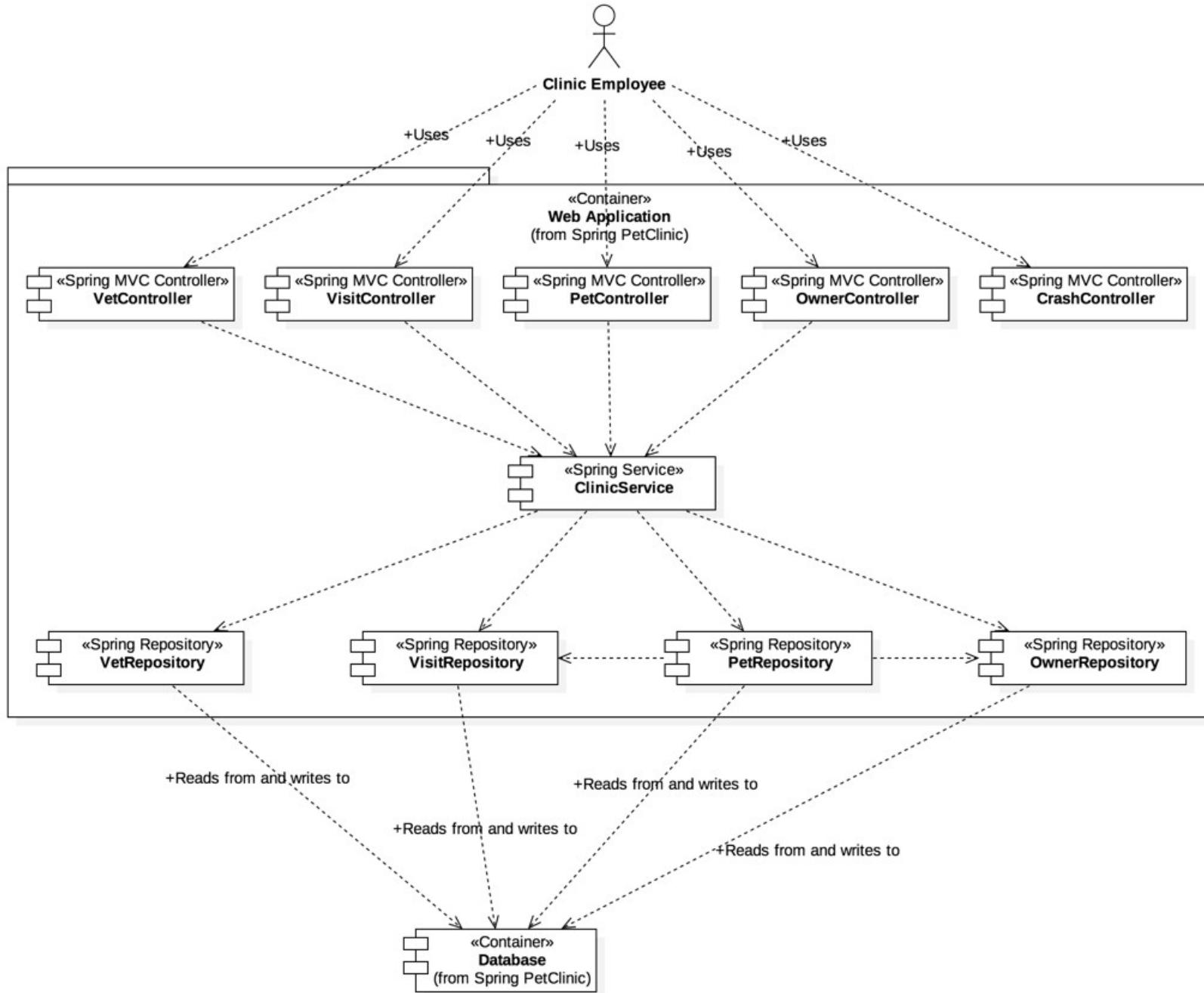


Component diagram for Spring PetClinic - Web Application

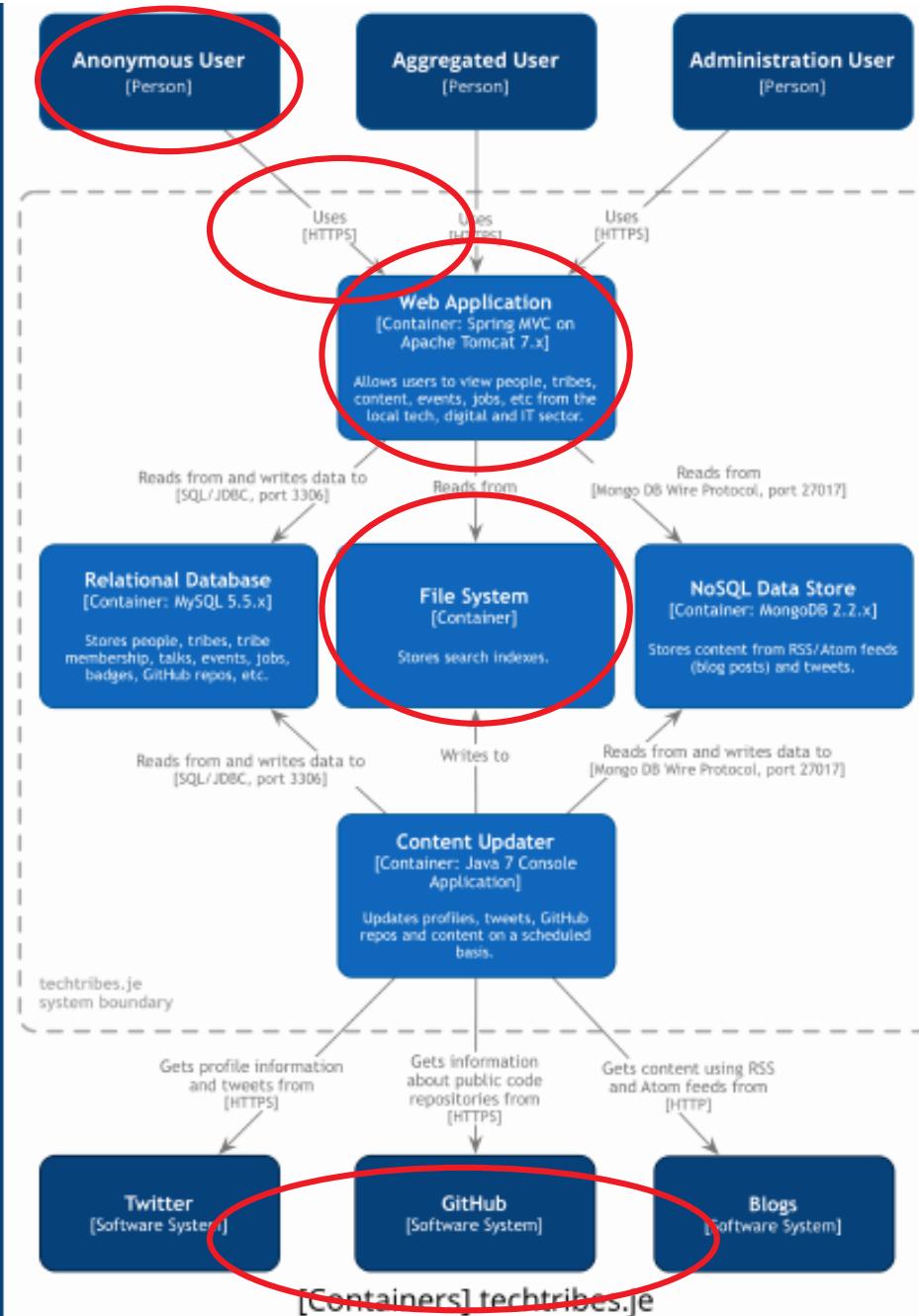
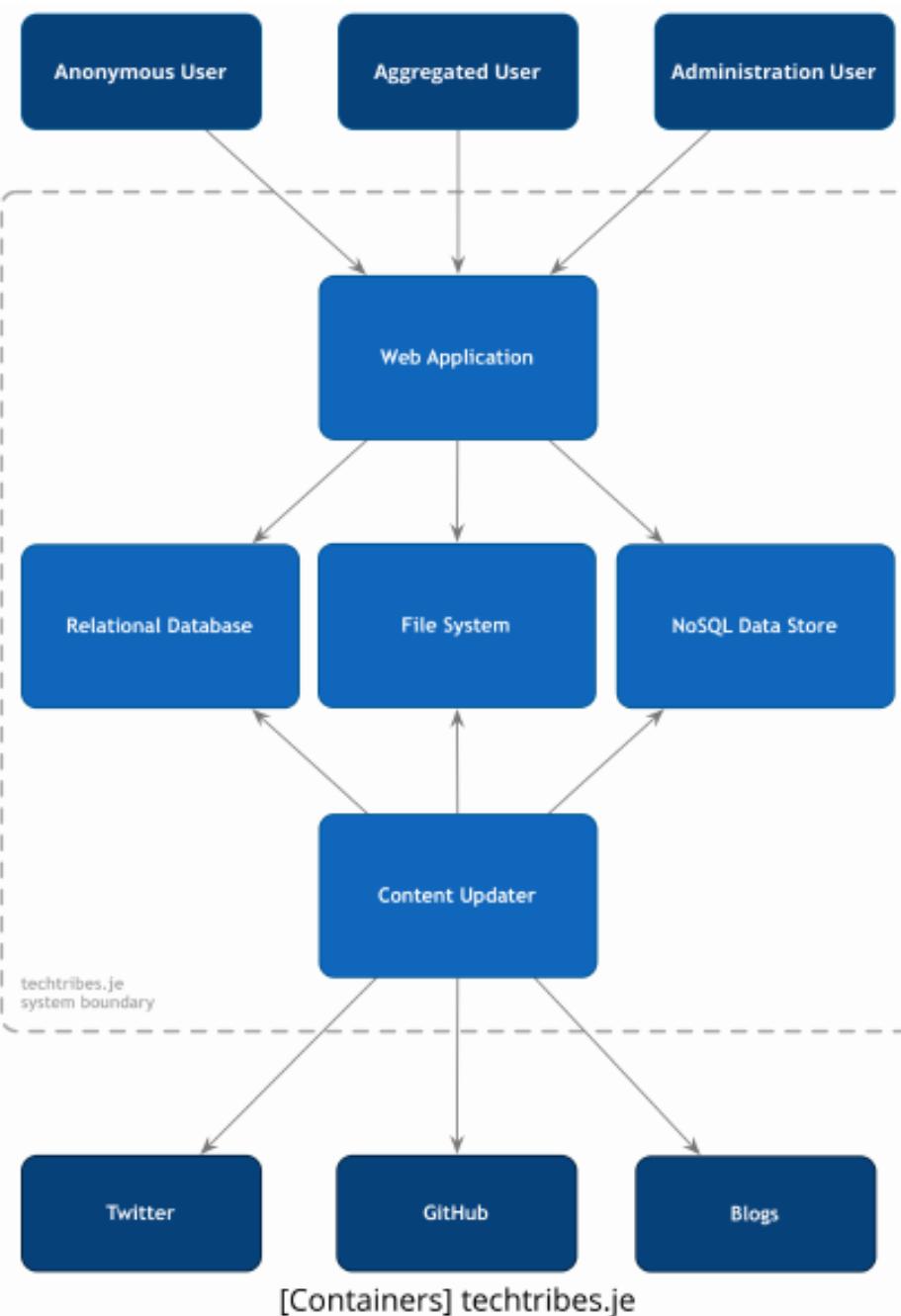
The Components diagram for the Spring PetClinic web application.

Last modified: Thursday 17 August 2017 10:15 UTC | Version: 95de1d9f8bf63560915331664b27a4a75ce1f1f6

LISHA Notação no Modelo C4

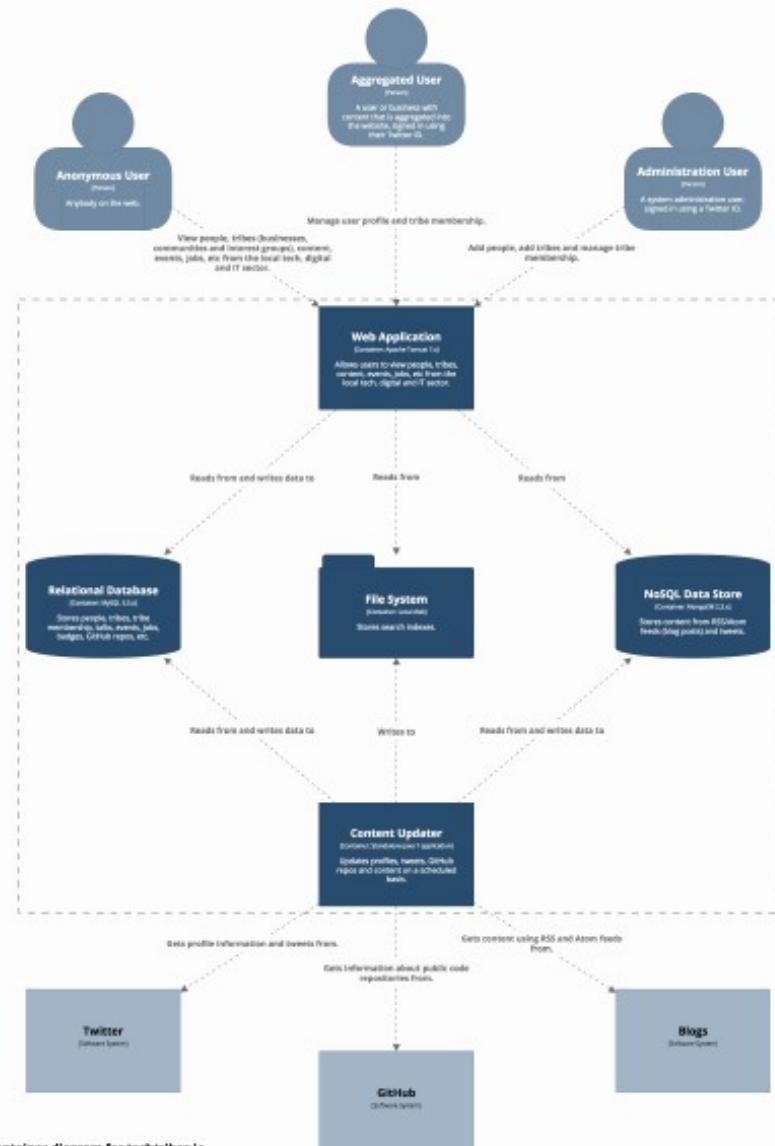
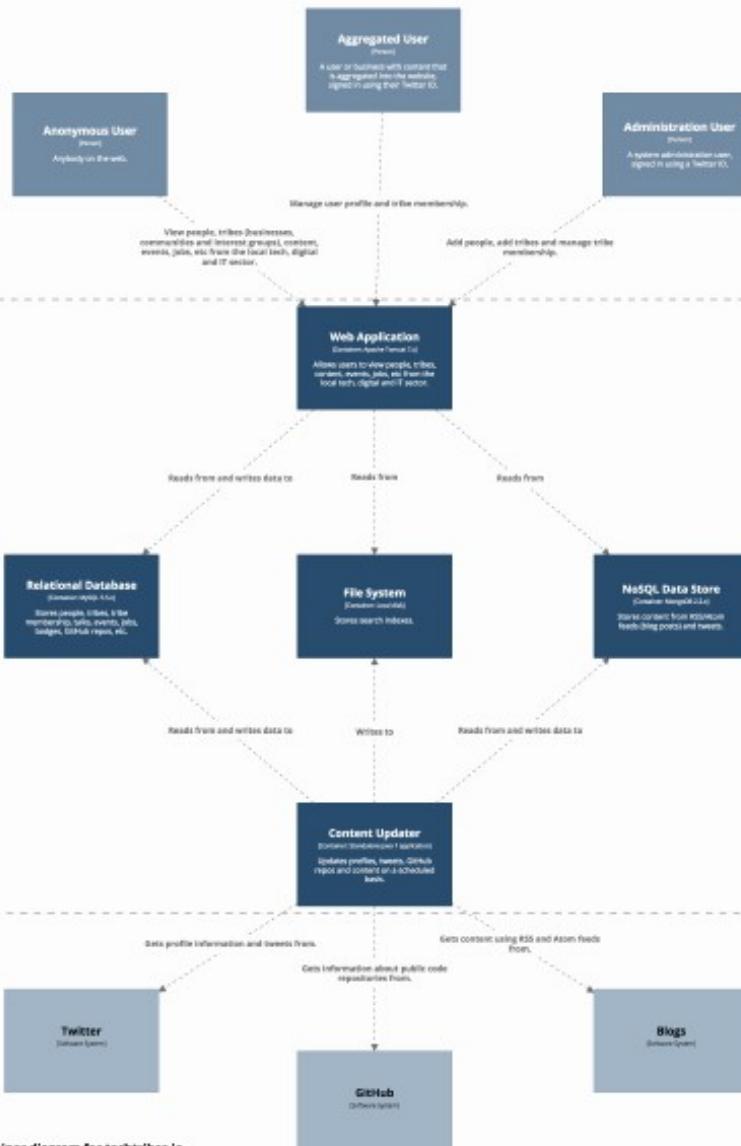


Dicas de Notação



Use formas e cores

Qual dois dos diagramas você usaria?



Notation, notation, notation

A software architecture diagram review checklist

General

Does the diagram have a title?	Yes	No
Do you understand what the diagram type is?	Yes	No
Do you understand what the diagram scope is?	Yes	No
Does the diagram have a key/legend?	Yes	No

Elements

Does every element have a name?	Yes	No
Do you understand the type of every element? (i.e. the level of abstraction; e.g. software system, container, etc)	Yes	No
Do you understand what every element does?	Yes	No
Where applicable, do you understand the technology choices associated with every element?	Yes	No
Do you understand the meaning of all acronyms and abbreviations used?	Yes	No
Do you understand the meaning of all colours used?	Yes	No

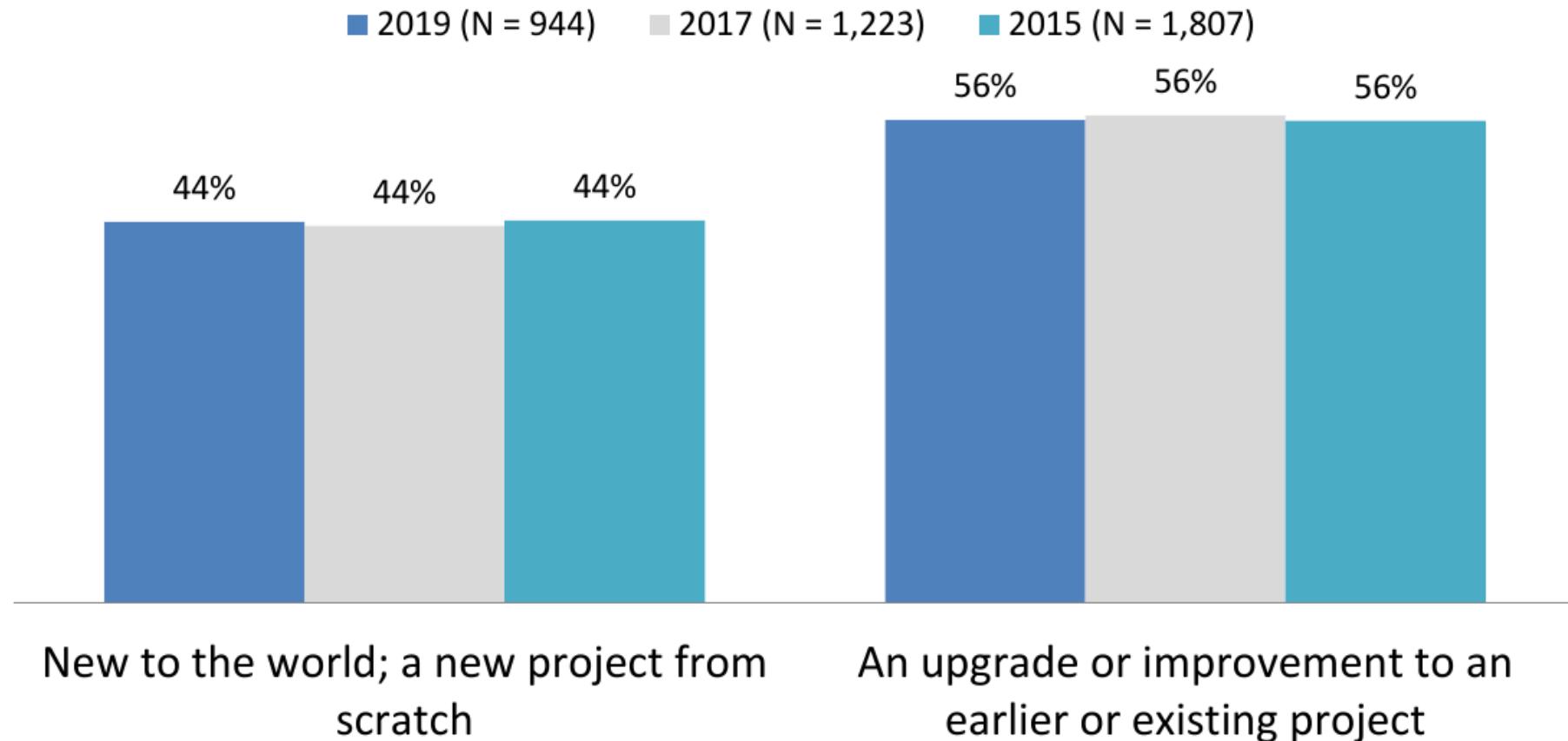
Quais ferramentas suportam C4?

- No site do modelo (<https://c4model.com/>) existe uma lista de ferramentas
 - C4-PlantUML
 - Structurizr (do próprio autor)
 - Archi
 - Sparx Enterprise Architect
 - Draw.io
 - etc
- Discussão e comentários?

- Introdução
- Fluxo de projeto Modelo C4
- Fluxo de projeto usando a linguagem C e UML
 - Conceituação
 - Exemplos
 - Exercícios
- Fluxo de projeto usando a linguagem C++ e UML
 - Conceituação
 - Exemplos
 - Exercícios

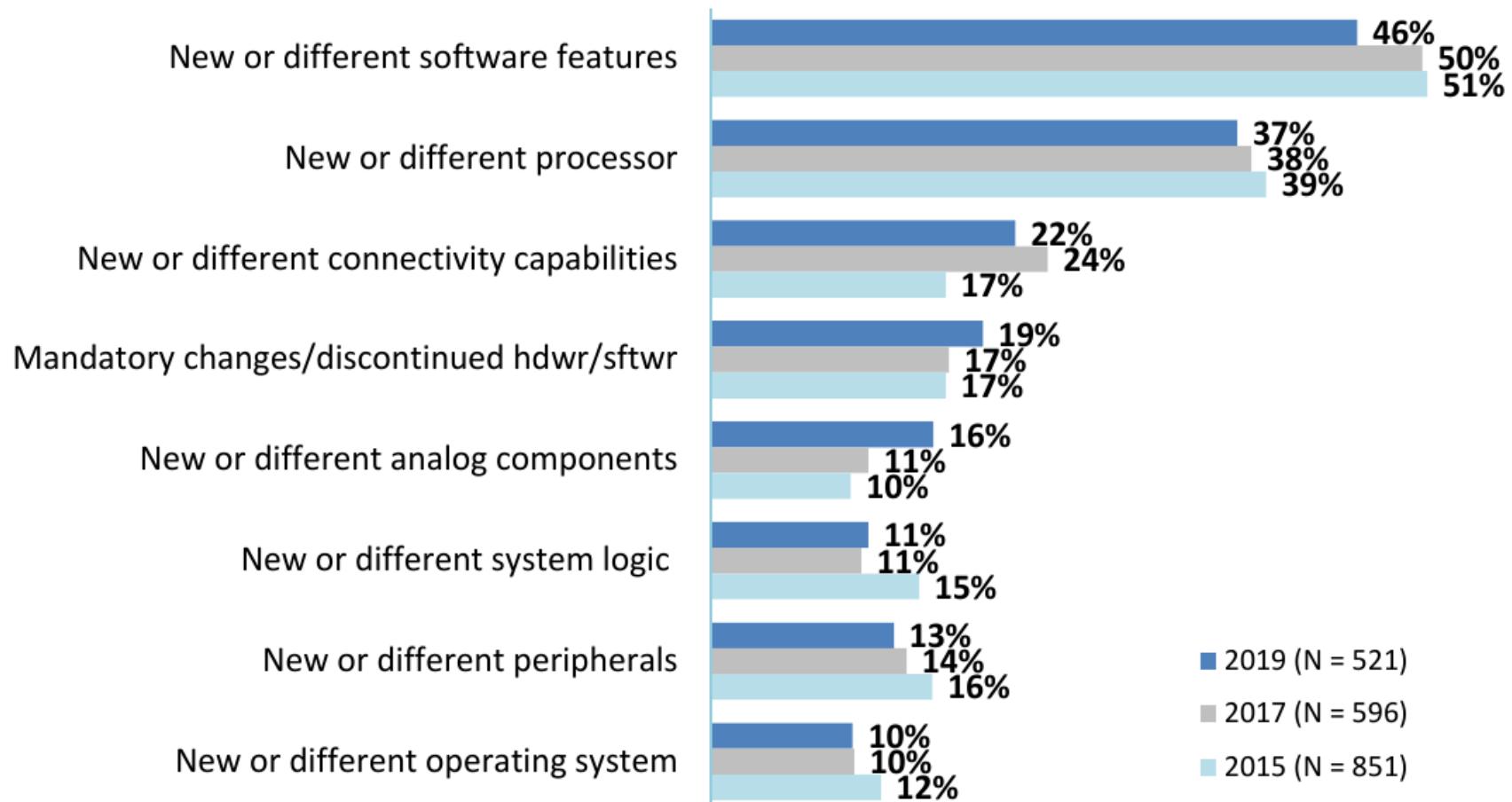
■ EETimes Embedded Markets Study 2019

My current embedded project is...



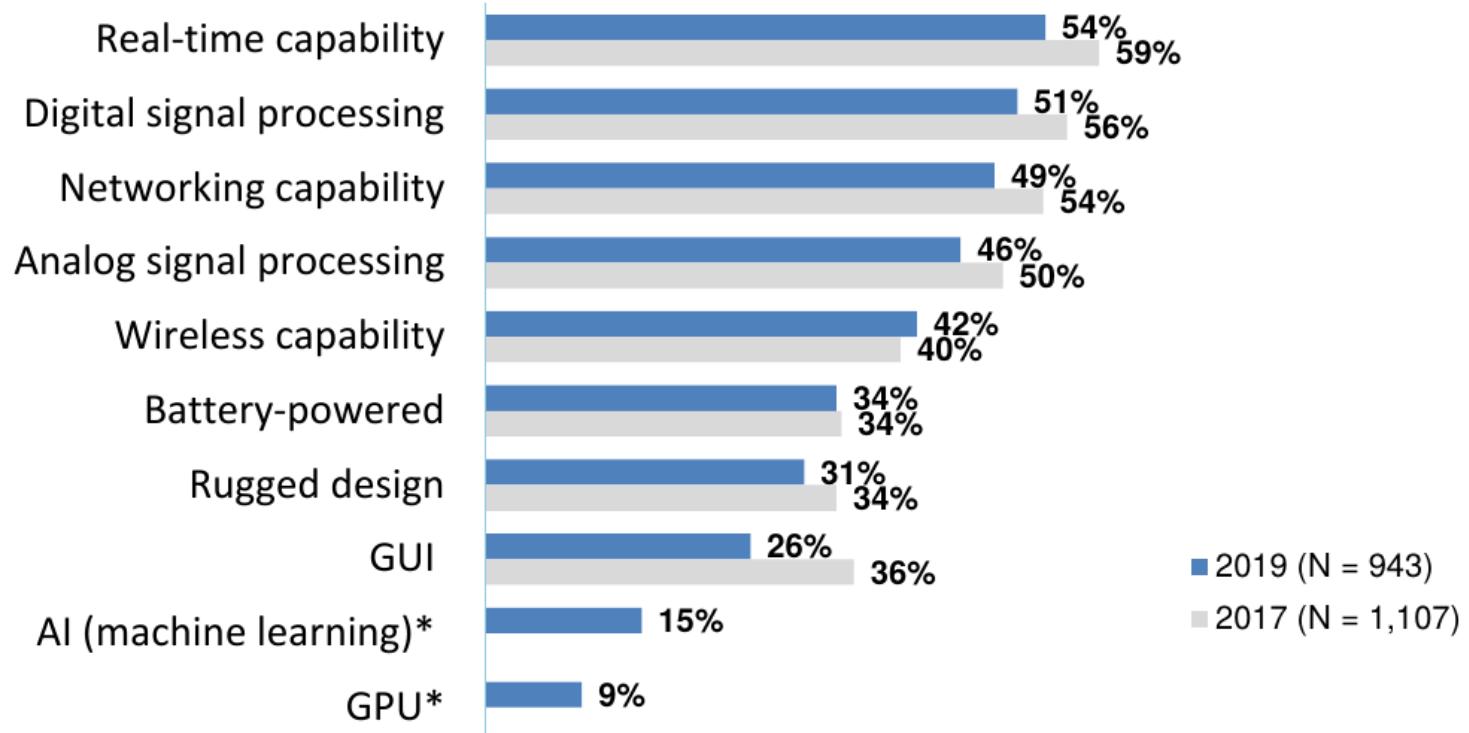
■ EETimes Embedded Markets Study 2019

What does the upgrade or improvement include?



■ EETimes Embedded Markets Study 2019

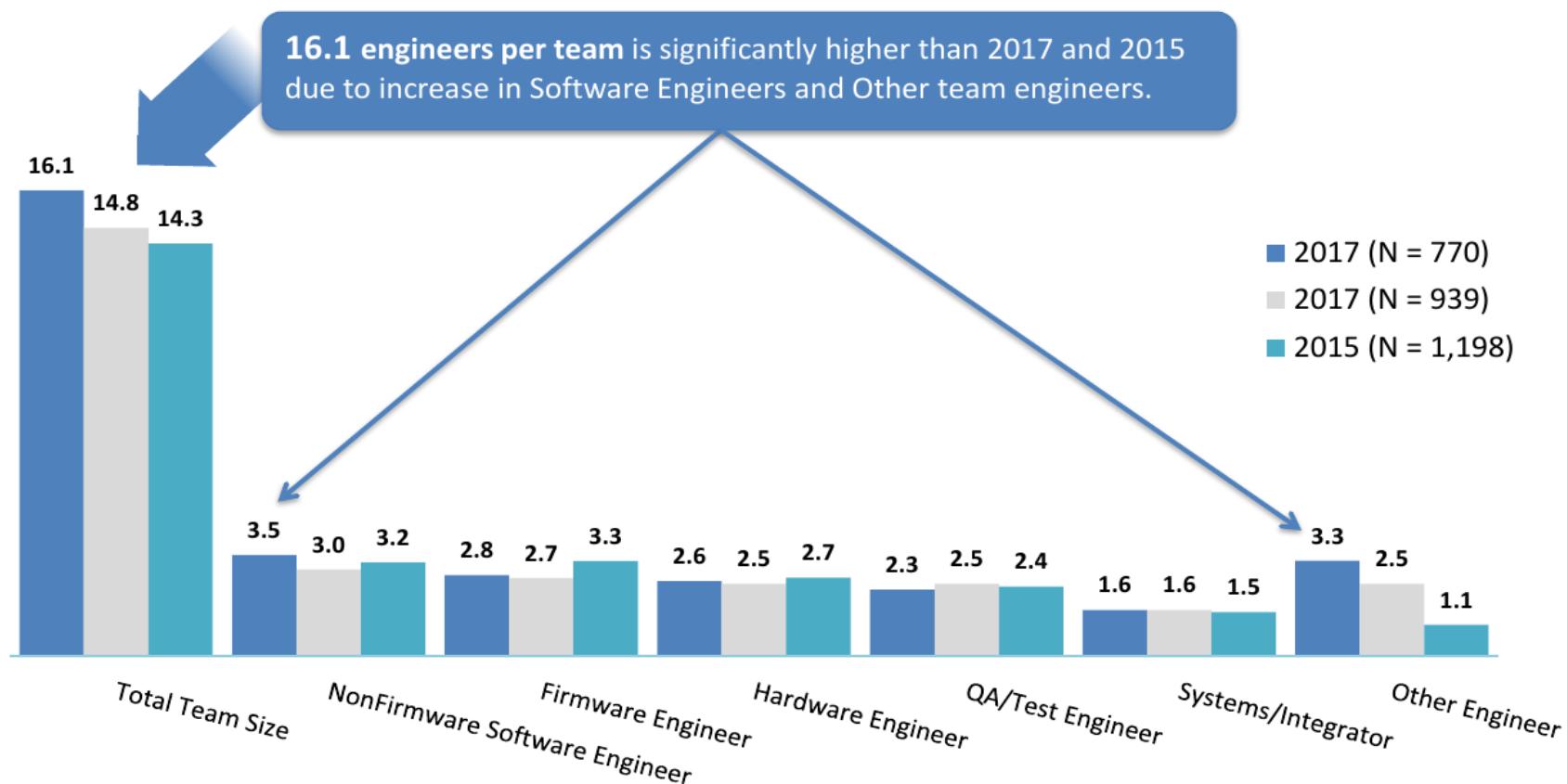
Which of the following capabilities are included in your current embedded project?



*AI and GPU were added in 2019.

■ EETimes Embedded Markets Study 2019

How many people are on your embedded project team?



Team size for Americas is **15.1** engineers/ team.

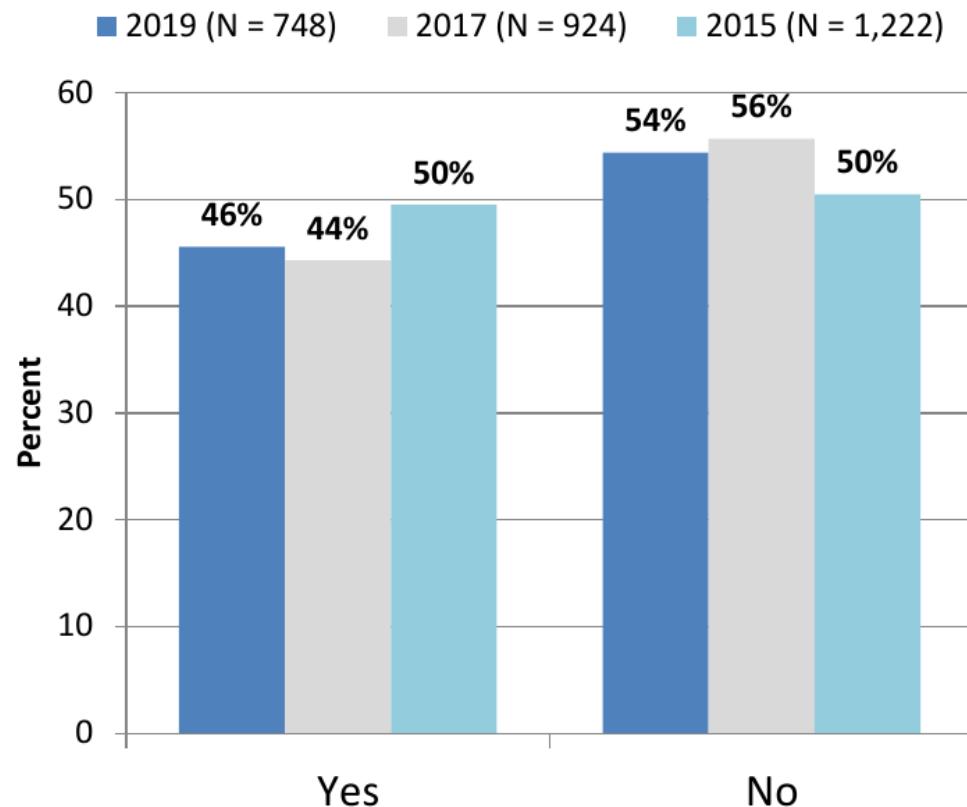
Team size for EMEA is **14.1** engineers/ team.

Team size for APAC is **19.6** engineers/ team.

Teams also work with an average of 2.7 outside vendors on a typical project.

■ EETimes Embedded Markets Study 2019

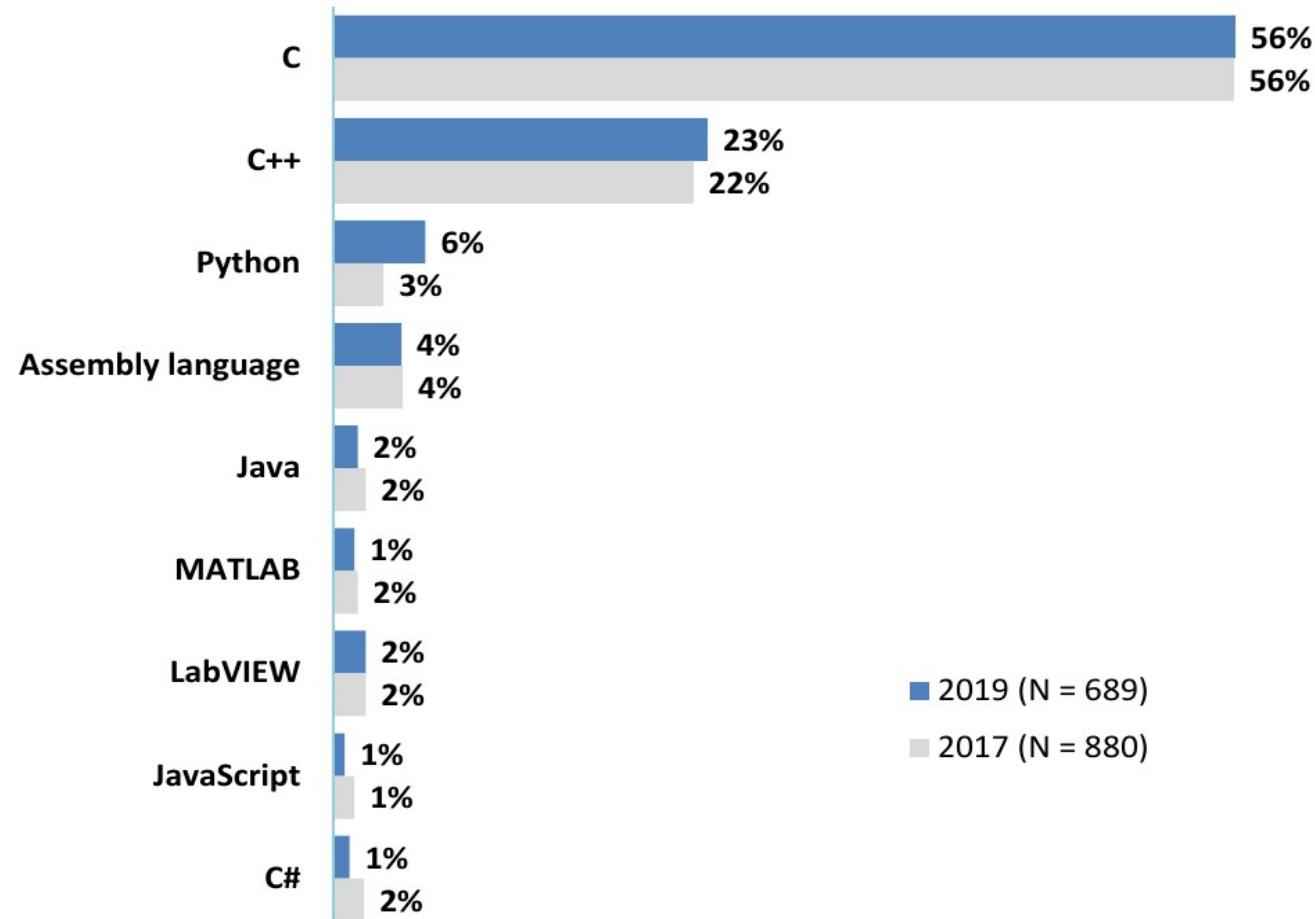
Did you start your current embedded design with a development board?



Development Board Started With (Write-in recall answers only)	N=281	%
STMicroelectronics	43	15.3%
TI	30	10.7%
NXP	20	7.1%
Raspberry Pi	19	6.8%
Microchip	14	5.0%
Arduino	13	4.6%
Xilinx	13	4.6%
Atmel	11	3.9%
Espressif ESP-32	7	2.5%
Renesas	7	2.5%
Silicon Labs	6	2.1%
Nordic	5	1.8%
Digilent	4	1.4%
Nucleo Board	4	1.4%
ZedBoard	4	1.4%
Analog Devices	3	1.1%
Beaglebone Black	3	1.1%
Cypress	3	1.1%
AdaFruit 'Feather' Cortex-M4	2	0.7%
ARM	2	0.7%
Atmega	2	0.7%
Avnet Picozed	2	0.7%

■ EETimes Embedded Markets Study 2019

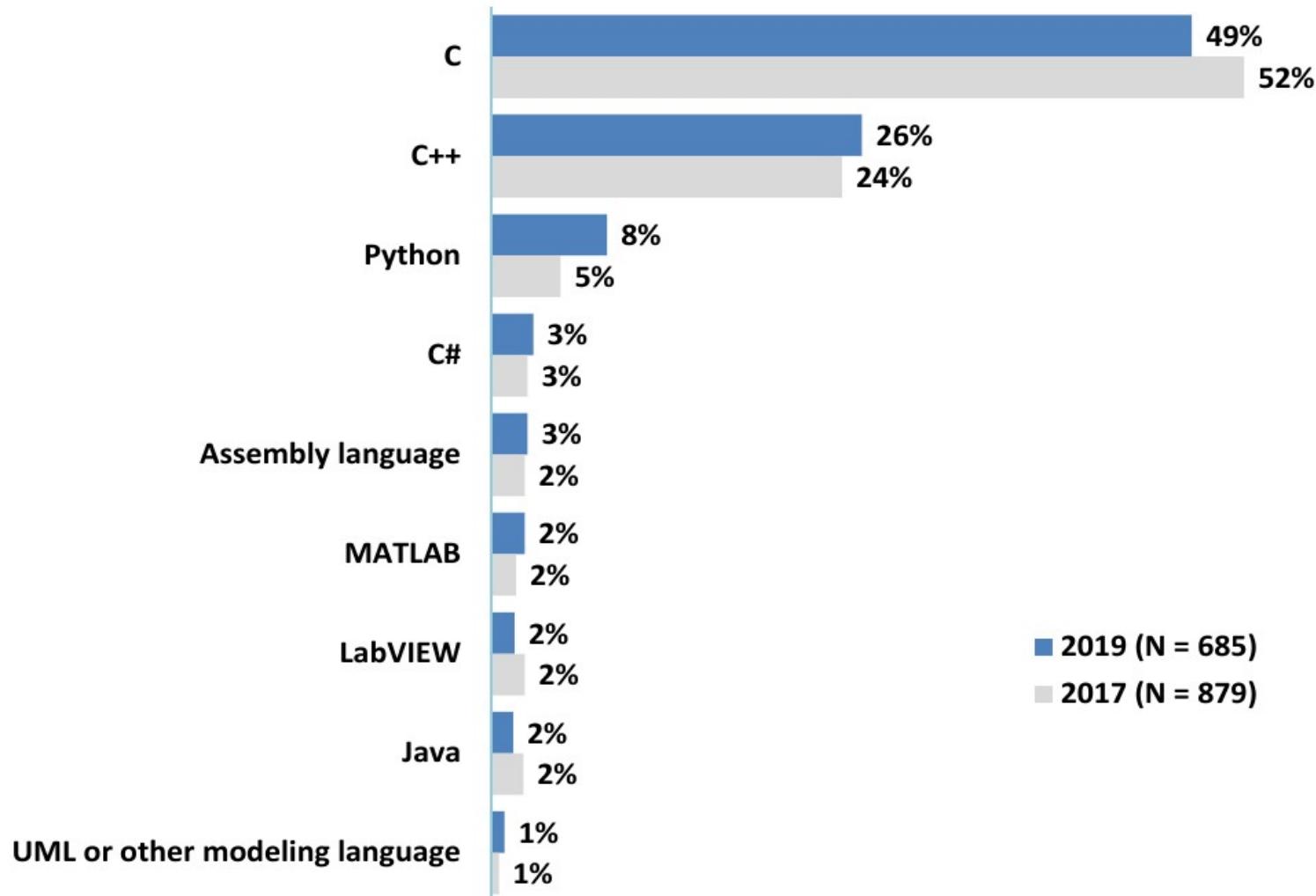
My current embedded project is programmed mostly in:



Importância de C e C++

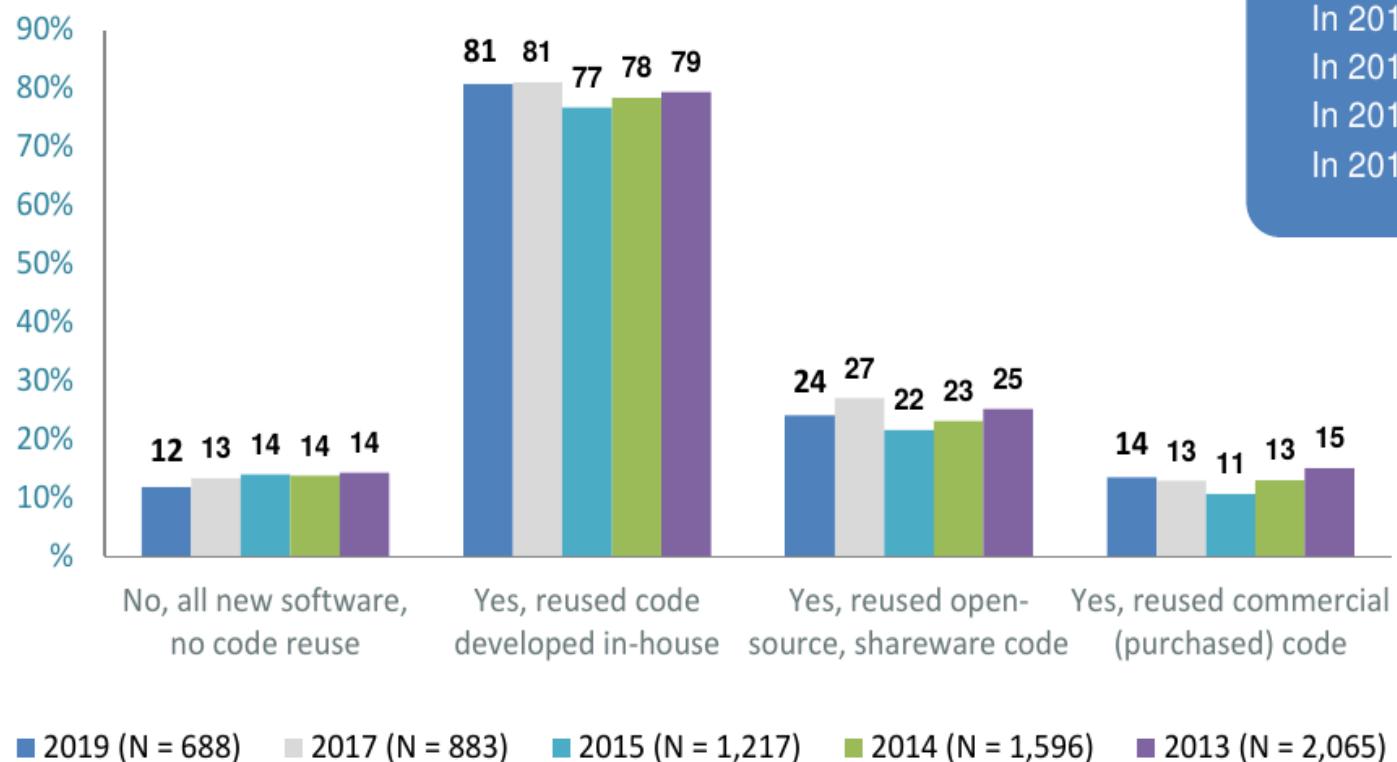
■ EETimes Embedded Markets Study 2019

*My next embedded project will likely be
programmed mostly in:*



■ EETimes Embedded Markets Study 2019

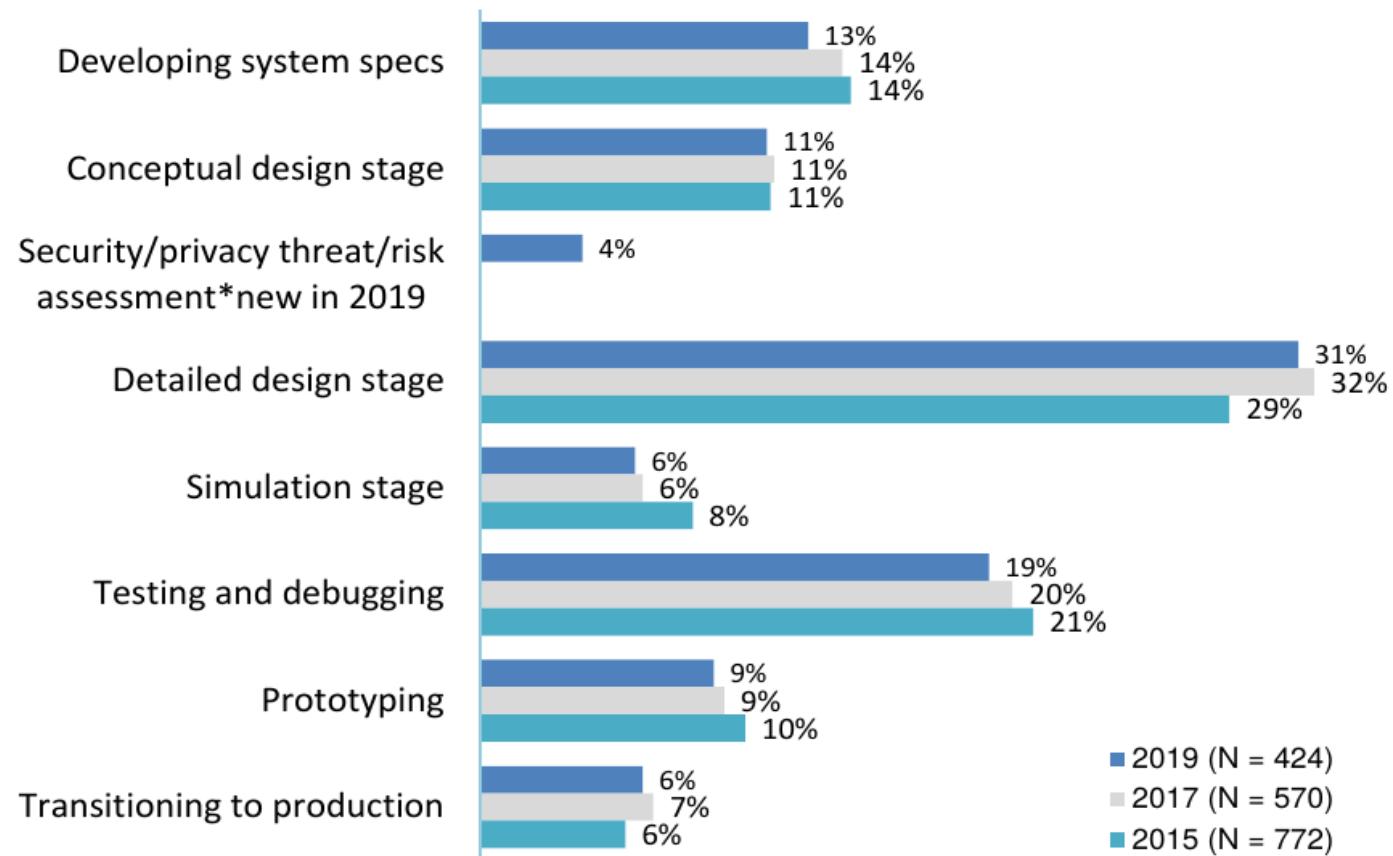
Does your current project reuse code from
a previous embedded project?



Importância do Fluxo de Projeto

■ EETimes Embedded Markets Study 2019

What percentage of your design time is spent on each of the following stages?



- Maior parte dos projetos de sistemas embarcados e tempo real são implementados em C
- UML é baseada em conceitos de orientação a objetos (C++)
- Três formas de usar UML com C
 - Organizar o código em C para ser “parecido” ao orientado a objetos e usar praticamente todos os conceitos de UML
 - Usar o perfil FunctionalC da UML usando estruturas e arquivos
 - Proposta: usar Tipo de Dados Abstratos

- Algumas orientações para usar C em uma estrutura parecida de C++
 - Classes viram estruturas com os dados (atributos) e ponteiros para funções (métodos)
 - Herança é implementada através de estruturas aninhadas (nested) representando a classe base e as derivadas
 - Polimorfismo é gerado através de tabelas de funções virtuais (usando ponteiro de função)

■ Qual o problema desta abordagem?

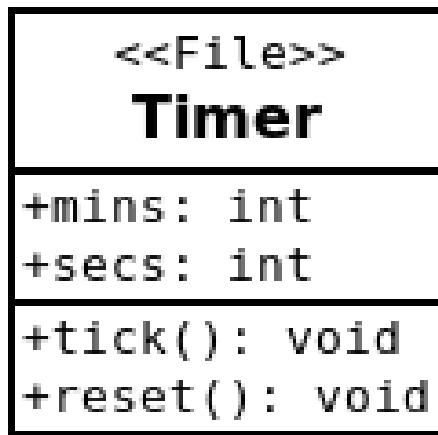
- Representar código legado sem ter que modificar todo o código existente
- Os desenvolvedores podem se sentirem mais confortáveis quando estiverem usando conceitos tradicionais da linguagem C
- Questões de certificações tornam-se complicadas

- A segunda proposta é usar o perfil FuncionalC
- Proposto pela IBM, sendo as diferenças:

Diagram type	EC diagram	UML basis diagram	Description
Requirements	Use case diagram	Use case diagram	Represents uses of the system with relations to actors
Structure	Build diagram	Component diagram	Shows the set of artifacts constructed from the source files, such as executables and libraries
	Call graph	Class diagram	Shows the calls and their sequences among sets of functions
	File diagram	Class diagram	Shows the set of .c and .h files and their relations, including
	Source code diagram	<none>	Shows the generated source code as editable text
Behavior	Message diagram	Sequence diagram	Shows sequences of calls and events sent among a set of files, including passed parameter values
	State diagram	State diagram	Shows the state machine for files and how their included functions and actions are executed as events (whether synchronous or asynchronous) are received
	Flowchart	Activity diagram	Details the flow of control for a function or use case

- Uma das principais razões pela não difusão da UML em programadores C é devido ao estilo de programação procedural ao contrário de uma forma de pensar orientado a objetos/classes
- A ideia é permitir que as funcionalidades tradicionais de C, como arquivos, funções e variáveis, sejam colocadas nos diagramas UML
- Isso permite usar a UML sem ter um amplo conhecimento de orientação a objetos

- Considere um objeto Timer que conta o tempo em minutos e segundos e possui duas operações (reset e tick)



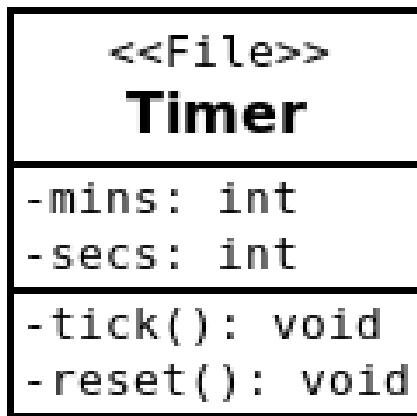
```
extern int mins;
extern int secs;

/* operação reset() */
void reset();

/* operação tick() */
void tick();
```

```
void tick() {
    secs++;
    if (secs > 59) {
        secs=0;
        mins++;
    }
}
```

- Se mudássemos o diagrama do Timer para o representado abaixo, o que impactaria no código?



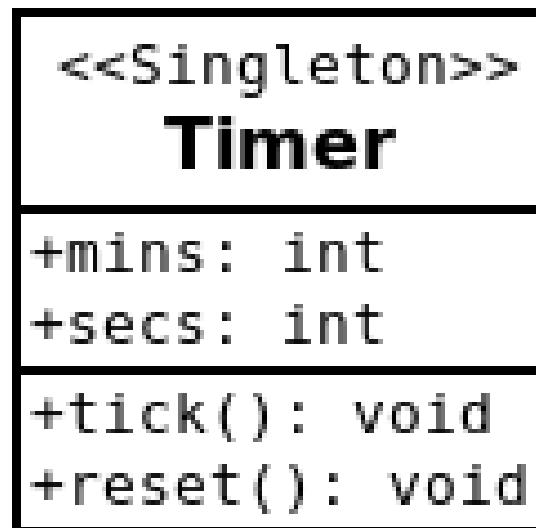
```
static void reset() {
    secs = 0;
    mins = 0;
}
```

**Uma variável estática só pode ser usada dentro do arquivo, então em C não faz sentido declarar a variável no .h.
Declare a variável no .c e sem usar extern**

- Em geral, com algumas regras, podemos reutilizar os diagramas da UML para C
 - Object-based modeling
- Diagramas de Classes
 - Mostra a API de bibliotecas C
 - Mostra as relações através de dependência
 - Mostra as estruturas e enumerações através dos estereótipos <<structure>> e <<enumerate>>
- Diagramas de Pacotes
 - Pode representar as bibliotecas utilizadas e suas relações
- Diagramas de Sequência
 - Troca de mensagens entre as funções
- Diagramas de Máquina de Estado

- Mapeamento dentro de uma Classe
 - Funções globais externas => métodos públicos
 - Funções locais estáticas => métodos privados
 - Variáveis globais externas => atributos públicos
 - Variáveis locais estáticas => atributos privados
 - Estruturas => classe com o estereótipo <<struct>> ou <<structure>>
 - Constantes com #define => classe com o estereótipo <<enumerate>> ou <<enumeration>>

- Considere um objeto Timer que conta o tempo em minutos e segundos e possui duas operações (reset e tick)



```
struct timer_t {
    int mins; /* atributo mins */
    int secs; /* atributo secs */
};

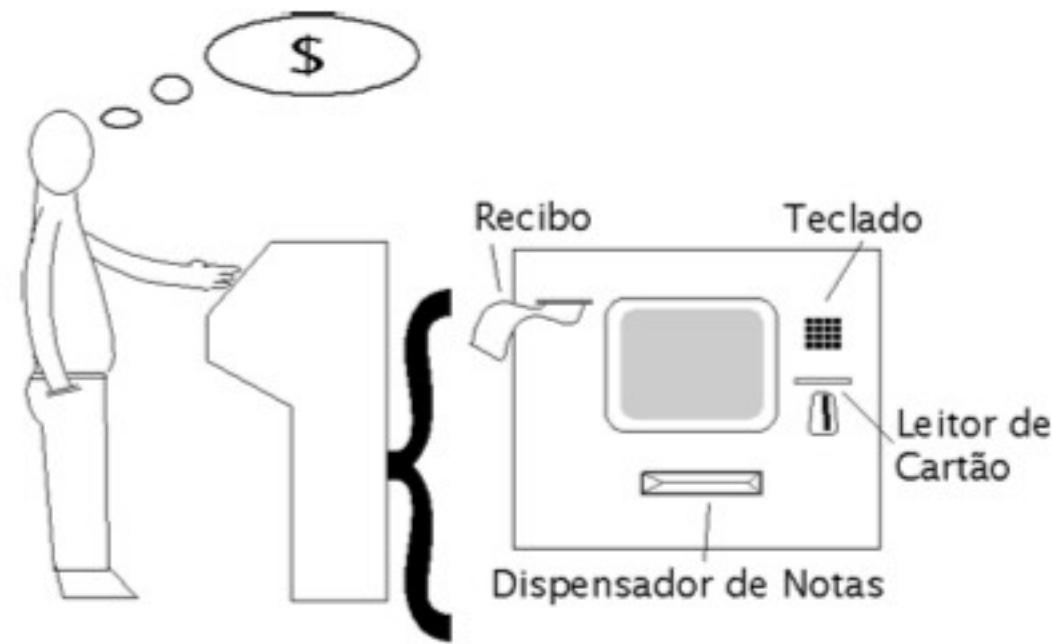
/* operações */
/* reset() */
void Timer_reset();
/* tick() */
void Timer_tick();
```

■ Em resumo, o fluxo proposto consiste em:

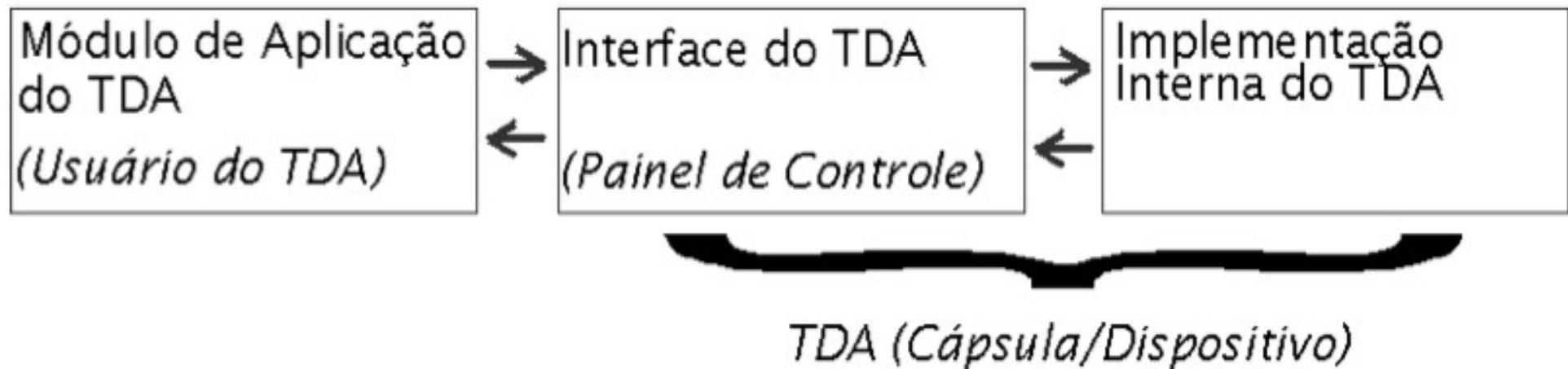
- Diagramas de casos de uso
- Diagramas de máquina de estados
- Diagramas de classes com o mapeamento para a linguagem C
- Diagramas de sequência

- Tipo de Dado Abstrato (TDA)
 - Pensar nas estruturas de dados em termos das operações que elas suportam e não da maneira como são implementadas
 - Encapsular ou esconder a implementação de quem efetivamente usa a estrutura de dados
- A organização do código em TDA permite:
 - Legibilidade (transparência)
 - Reaproveitamento de código
 - Facilidade de teste (isolamento)
 - Programação simplificada da aplicação

- Pode ser visto como uma caixa preta que apresenta uma interface de operações sobre determinada(s) estrutura(s) de dados adequadas a funcionalidade que se pretende implementar
- Ex: caixa eletrônico



- O TDA fornece uma interface para o usuário com todas as operações sobre as estruturas de dados



Ex. TDA fração: operações principais

- `cria_fracao(N, D)` → recebe dois inteiros e retorna a fração N/D
- `acessa_numerador(F)` → recebe a fração (`fracao_t`) e retorna o numerador
- `acessa_denominador(F)` → recebe a fração (`fracao_t`) e retorna o denominador

```
typedef struct {
    int numerador;
    int denominador;
} fracao_t;
```

Ex. TDA fração: operações principais

```
fracao_t *soma(fracao_t f1, fracao_t f2) {
    int n1 = acessa_numerador(f1);
    int n2 = acessa_numerador(f2);
    int d1 = acessa_denominador(f1);
    int d2 = acessa_denominador(f2);
    return cria_fracao( n1 * d2+n2 * d1 , d1 * d2 );
}
```

Aplicação do TDA fração

```
#include fracao.h

int main(void)
{
    int n, d;
    printf ("Digite o numerador: ");
    scanf ("%d", &n);

    printf ("\nDigite o denominador: ");
    scanf ("%d", &d);

    fracao_t f = cria_fracao(n, d);

    fracao_t soma = soma_fracao(f, f);

    return 0;
}
```

Formato de um TDA

- O TDA é estruturado em dois aspectos
- **Modelo de dados** (a estrutura de dados que armazena as informações)
- Conjunto de **operações** ou **funções** que atuam sobre o modelo de dados
- Par (v, o) em que
 - v é o conjunto de valores
 - o é o conjunto de operações sobre os valores

- **Principais** (presentes na interface)
 - Construtoras: criam o TDA
 - De acesso: retornam informações sobre os elementos do TDA sem modificá-los (como get de OO)
 - De manipulação: modificam os elementos do TDA (incluindo as operações como set de OO)
 - Destruíadoras: destroem os dados dinamicamente alocados pelo TDA, se existirem, e garantem que os ponteiros estejam seguros
- **Privativas** (acessíveis apenas pelas funções do próprio TDA – como métodos privados)
 - Auxiliam na implementação das funções principais e não estão presentes na interface TDA

Interface do TDA

- Na interface do TDA, espera-se que sejam descritas todas as informações de utilização do mesmo:
 - Definição de constantes
 - Definições de tipos de dados
 - Definições de cada operação/função:
 - Pré-condições (estado de entrada exigido)
 - Pós-condições (estado de saída gerado)
 - Protótipo ou assinatura das funções

APLICA.H

```
#include "TDA_INTERFACE.H"
```

Descrição do Modelo de Dados da Aplicação

APLICA.C

```
#include "APLICA.H"
```

Implementação das operações sobre o modelo de dados da aplicação

TDA_INTERFACE.H

```
#include <stdio.h>
#include <stdlib.h>
```

...

Protótipo das operações de interface e respectivas pré e pós condições



TDA_PRIVADO.H

```
#include "TDA_INTERFACE.H"
```

Descrição(ões) da(s) estrutura(s) interna(s) do TDA e eventuais operações privativas ao TDA

TDA.C

```
#include "TDA_PRIVADO.H"
```

Implementação das operações sobre o TDA



Exemplo: TDA racional

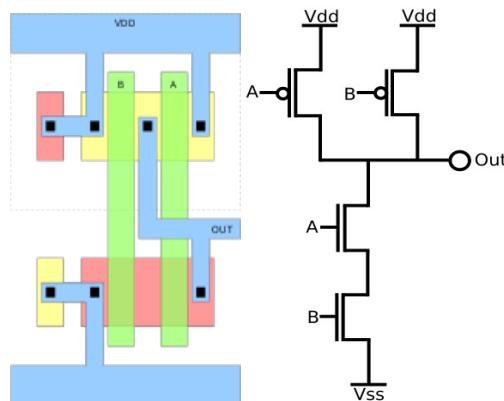
- Veremos o código exemplo do TDA racional
- Disponível online
- Para compilar utilize o Makefile: comando “make”
- Insira a seguinte instrução na função main (arquivo aplicacao.c) e veja o resultado:
 - **a->dem = 10;**
 - Porque houve erro de compilação?
 - Como fazemos para acessar os membros internos da estrutura rational_t?
 - Como podemos escrever as pré- e pós-condições para cada operação pública do TDA racional?

- Introdução
- Fluxo de projeto Modelo C4
- Fluxo de projeto usando a linguagem C e UML
 - Conceituação
 - Exemplos
 - Exercícios
- Fluxo de projeto usando a linguagem C++ e UML
 - Conceituação
 - Exemplos
 - Exercícios

Evolução do Nível de Abstração

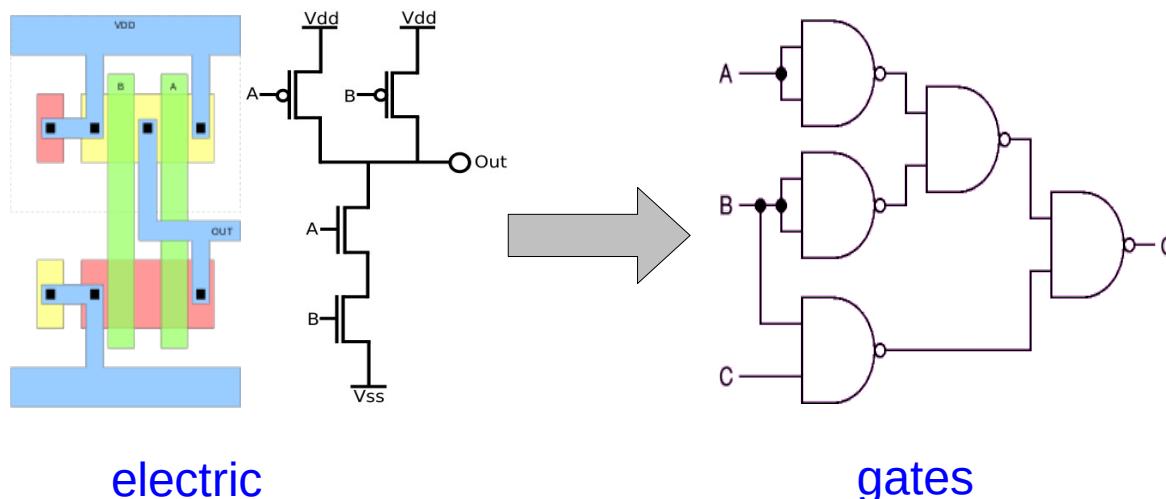
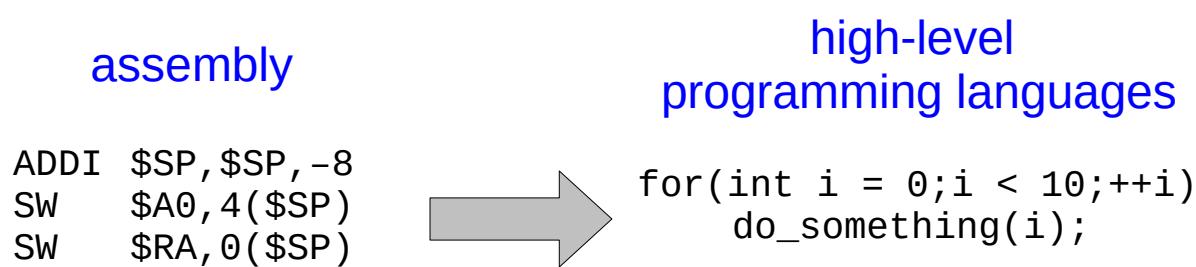
assembly

```
ADDI $SP, $SP, -8  
SW    $A0, 4($SP)  
SW    $RA, 0($SP)
```

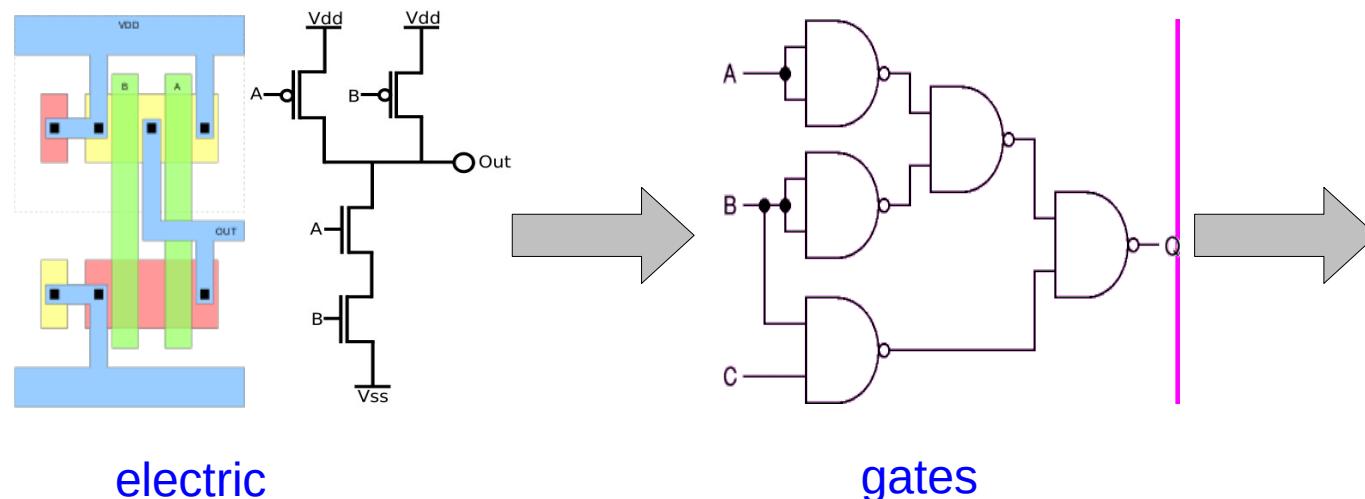
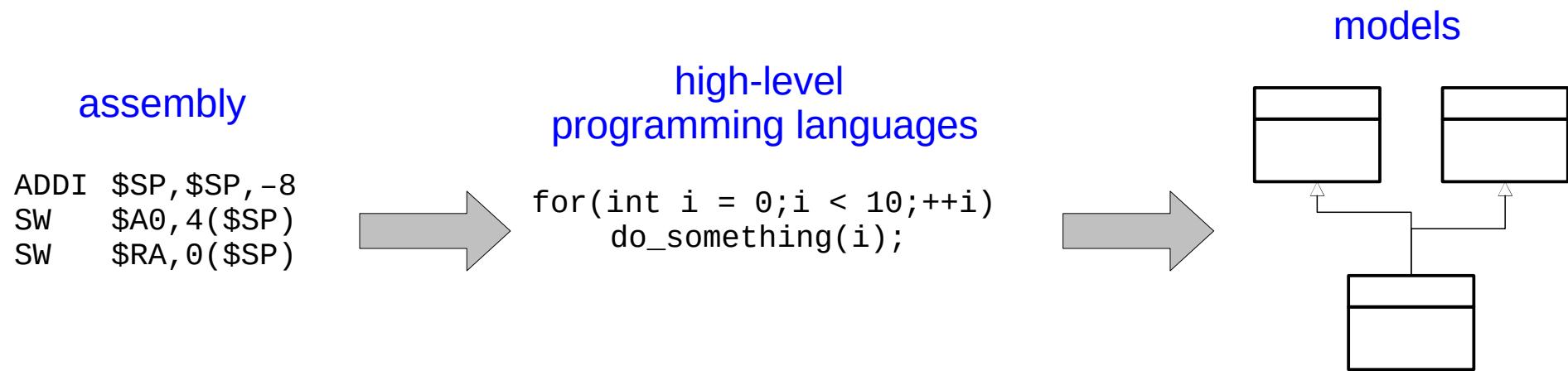


electric

Evolução do Nível de Abstração

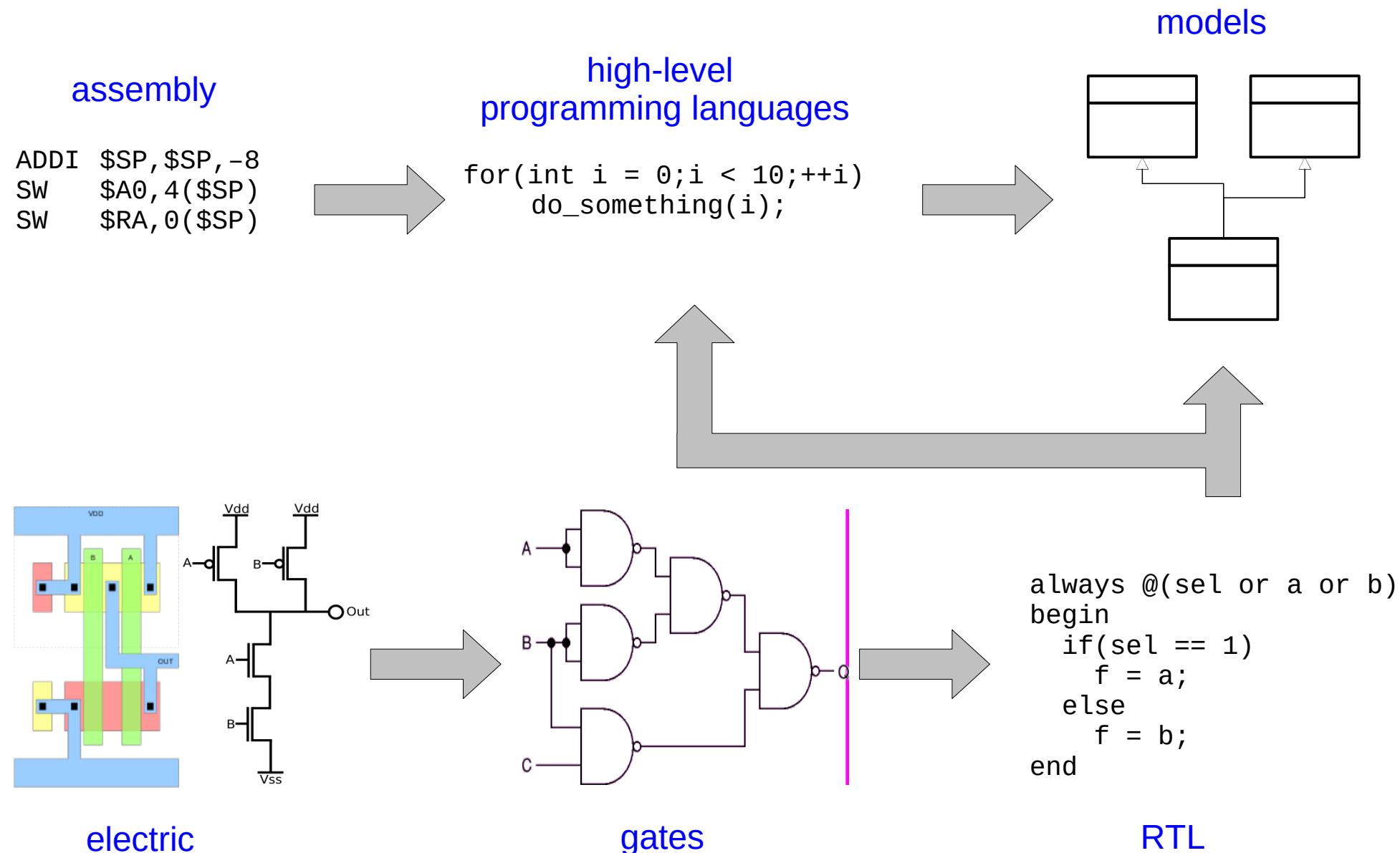


Evolução do Nível de Abstração



```
always @(sel or a or b)  
begin  
    if(sel == 1)  
        f = a;  
    else  
        f = b;  
end
```

Evolução do Nível de Abstração



- Criador do C++, Bjarne Stroustrup, fala porque ele criou o C++
 - Youtube: Bjarne Stroustrup: Why I Created C++ | Big Think
 - <https://www.youtube.com/watch?v=JBjjnqG0BP8&feature=youtu.be>

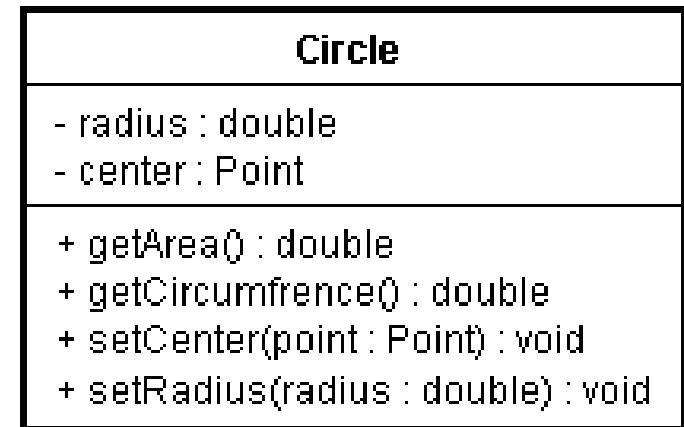
C++ tem pior desempenho que C?

- É comum o pensamento que C++ tem pior desempenho que C
- Mas essa afirmação é verdadeira?
 - <http://hildstrom.com/projects/langcomp/index.html>

- A ideia é seguir o mesmo fluxo de modelagem apresentado com a linguagem C
 - Diagramas de casos de uso
 - Diagramas de máquina de estados
 - Diagramas de classes
 - Diagramas de sequência
- Iremos ver alguns exemplos/exercícios relacionando a linguagem com o diagrama de classes

- Como seria o diagrama de classes do código abaixo?

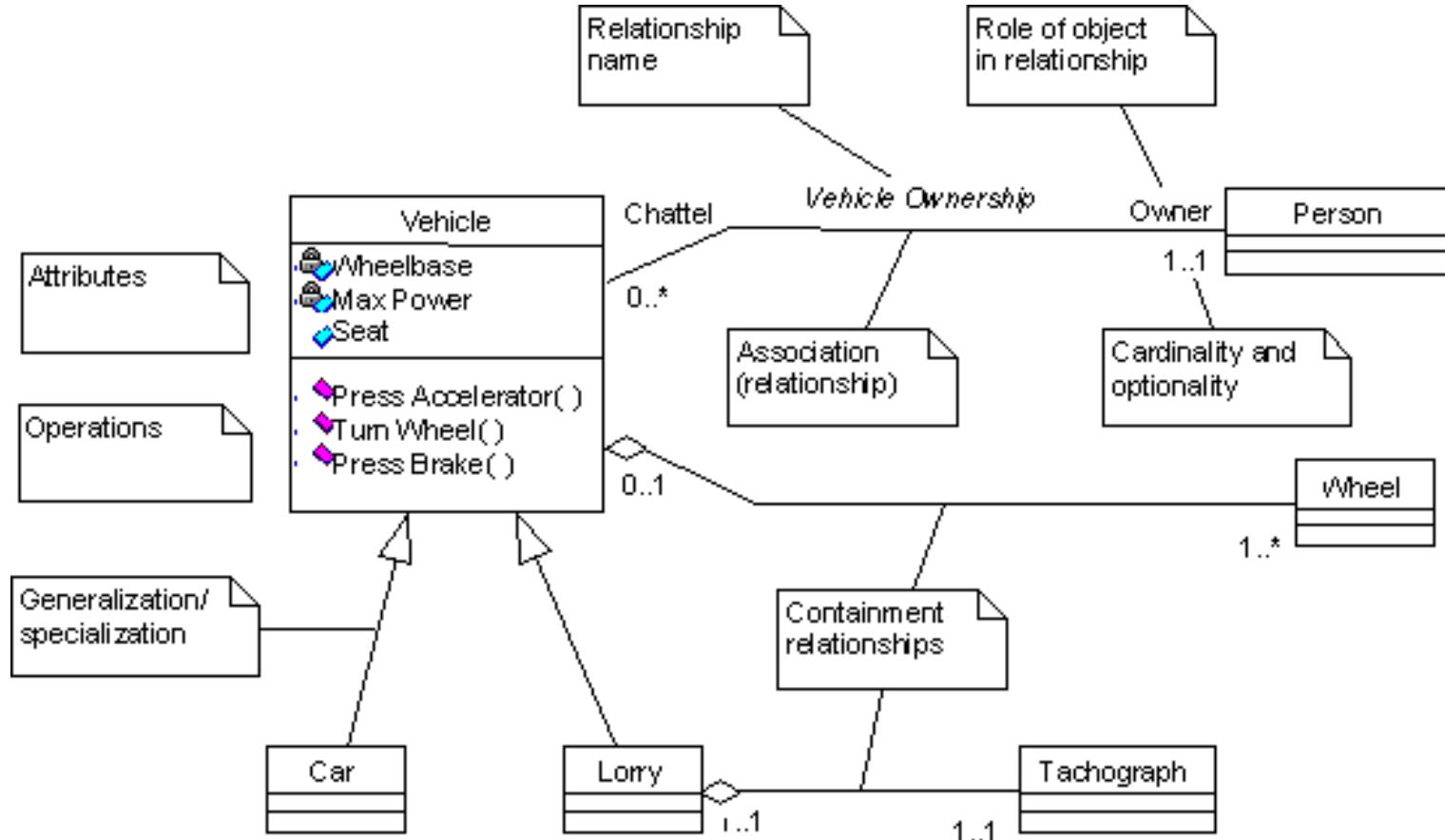
```
class Circle {  
private:  
    double radius;  
    Point center;  
public:  
    double getArea();  
    double getCircumference();  
    void setCenter(Point center);  
    void setRadius(double radius);  
};
```

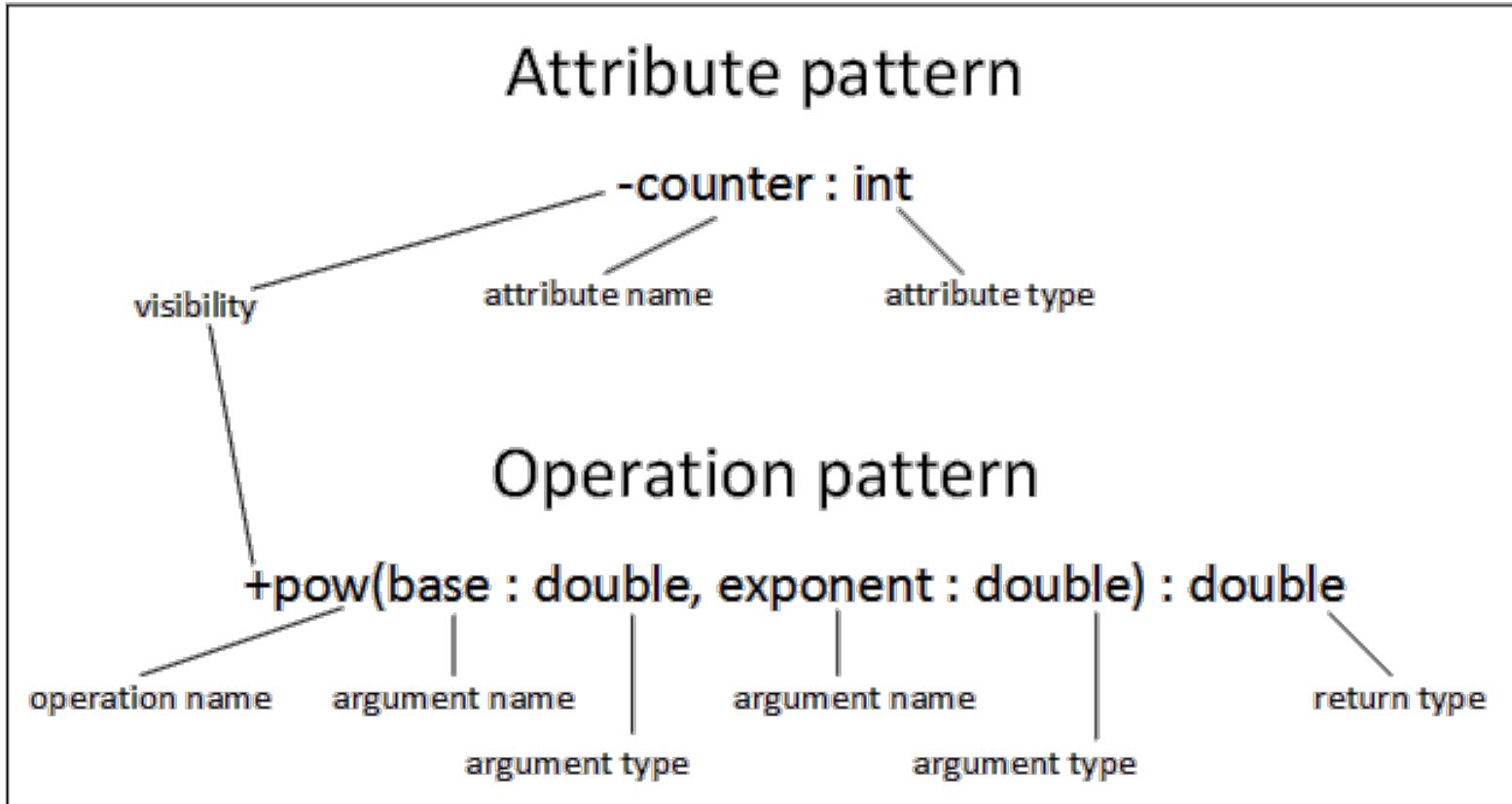


■ Relacionamento entre classes

- Associação (conhece)
- Dependência (usa)
- Composição (tem)
- Agregação (tem)
- Herança (é) e Realização
- Template

Revisão dos Relacionamentos





■ Associação

- Um objeto tem ciência de outro. Contém um ponteiro ou referência para outro objeto

```
class X {  
    X(Y *y) : y_ptr(y) {}  
    void SetY(Y *y) { y_ptr = y; }  
    void f() { y_ptr->Foo(); }  
    ---  
    Y *y_ptr; // pointer  
};
```



■ Dependência

- Uma classe depende da outra se a classe independente é um parâmetro ou variável local de um método da classe dependente

```
class X {  
    ...  
    void f1(Y y) {...; y.Foo(); }  
    void f2(Y *y) {...; y->Foo(); }  
    void f3(Y &y) {...; y.Foo(); }  
    void f4() { Y y; y.Foo(); ...}  
    void f5() {...; Y::StaticFoo(); }  
};
```



■ Agregação

- Ocorre quando uma classe possui uma coleção de uma ou mais classes
- Associação difere da agregação pois não tem a relação de “conter”

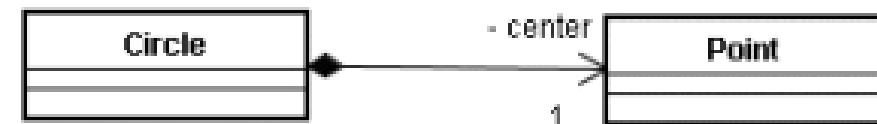
```
class Window {  
public:  
    //...  
private:  
    vector itsShapes;  
};
```



■ Composição

- É uma forma mais forte de agregação. Ocorre quando uma classe possui uma coleção de uma ou mais classes, mas existe uma dependência de vida
- Se a classe que contém é deletada, as classes contidas também o são

```
class Circle {  
private:  
    ...  
    Point center;  
    ...  
};
```

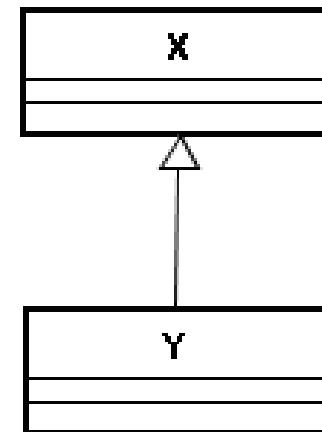


■ Outros exemplos de composição

```
class X {  
    ...  
    Y a; // 1; Composition  
    Y b[10]; // 0..10; Composition  
};  
  
class X {  
    X() { a = new Y[10]; }  
    ~X(){ delete [] a; }  
    ...  
    Y *a; // 0..10; Composition  
};  
  
class X {  
    ...  
    vector a;  
};
```

■ Herança

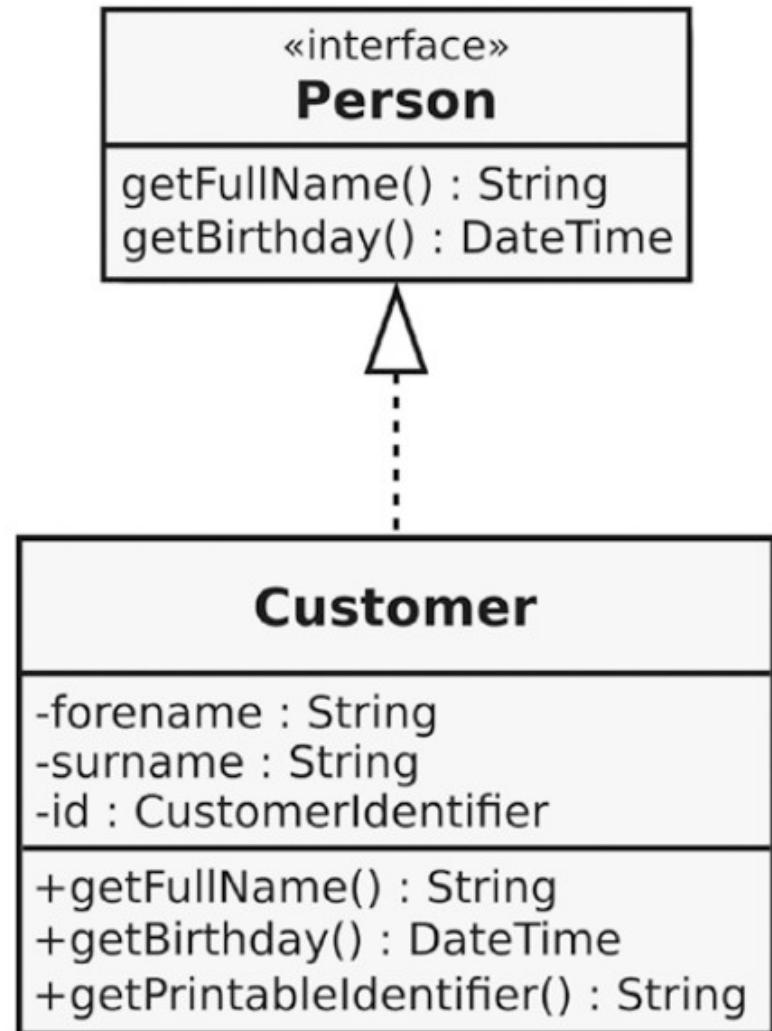
```
class X {  
    ....  
};  
  
class Y : public X {  
private:  
    ...  
};
```



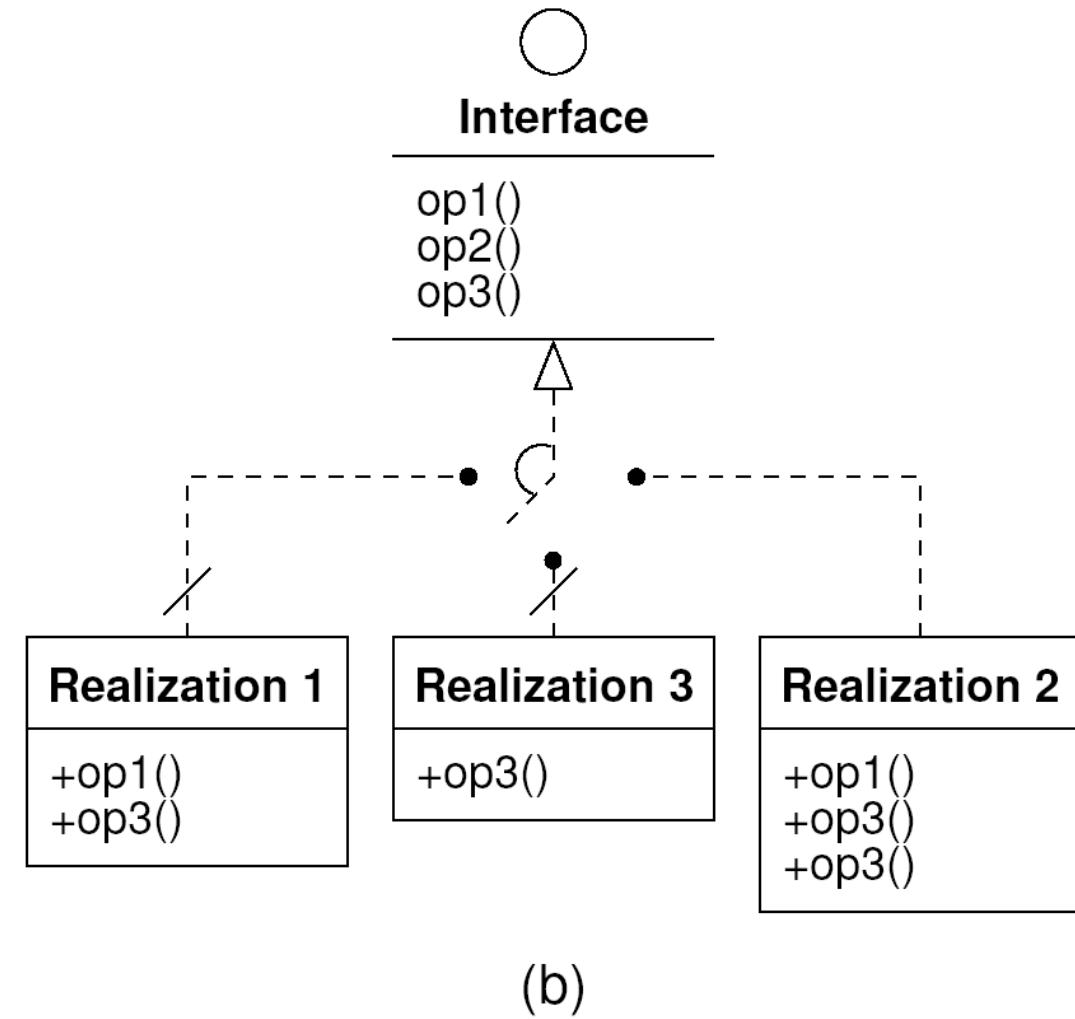
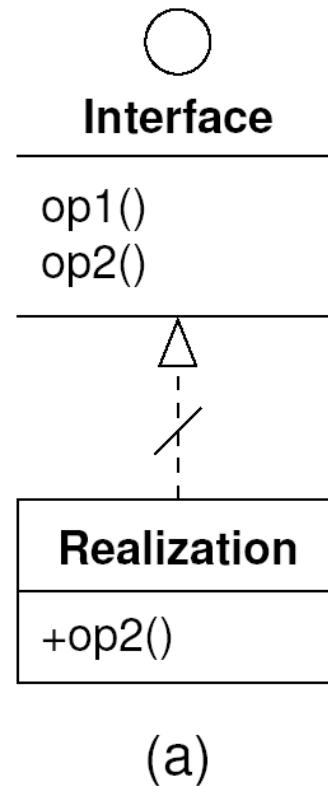
■ Realização e polimorfismo

```
class Person {
public:
    virtual ~Person() { }
    virtual std::string getFullName() const = 0;
    virtual DateTime getBirthday() const = 0;
};

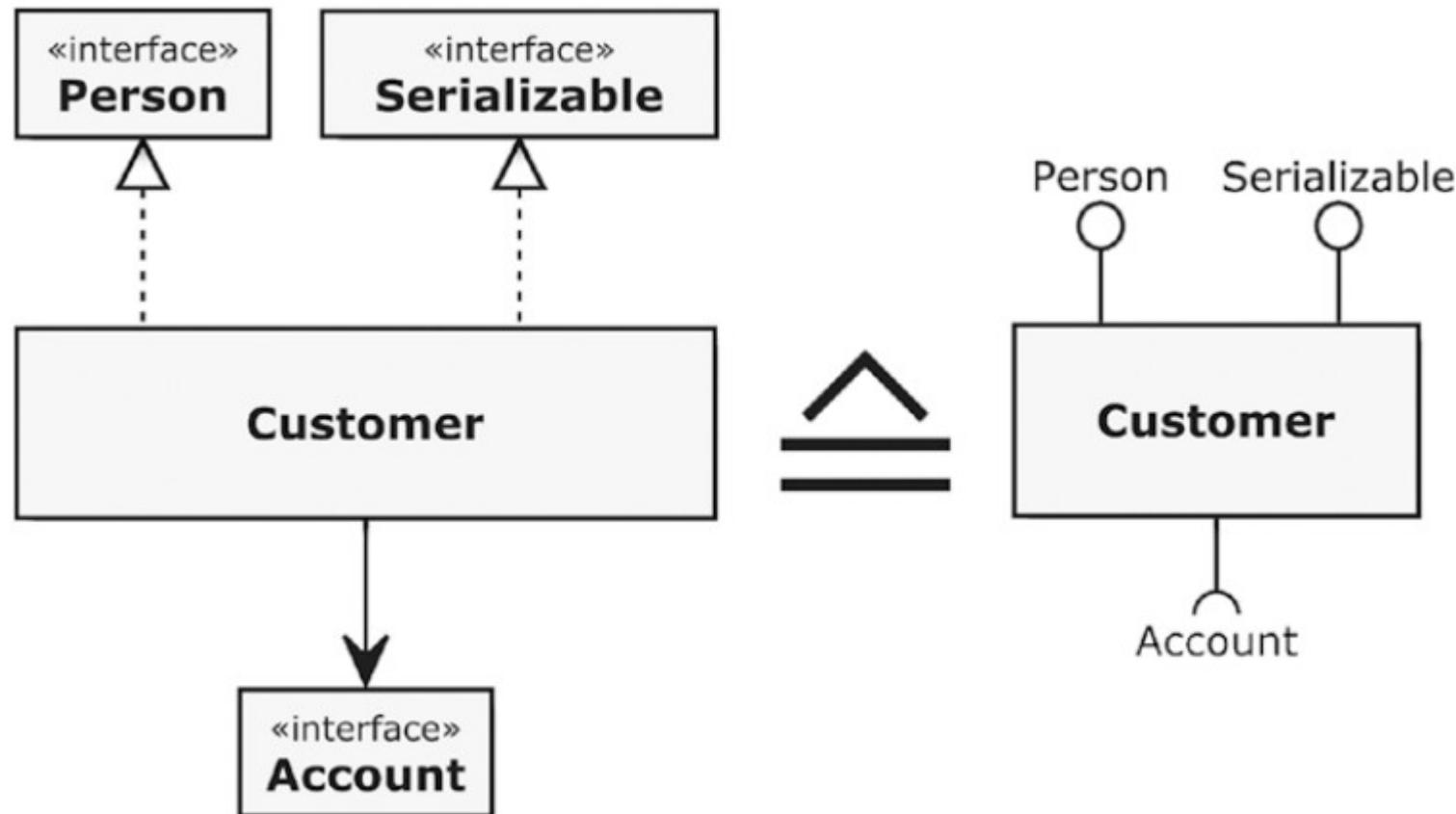
class Customer : public Person {
public:
    Customer();
    ~Customer();
    std::string getFullName();
    std::string getBirthday();
    std::string getPrintableIdentifier();
private:
    std::string forename, surname;
    int id;
};
```



■ Realização completa ou parcial



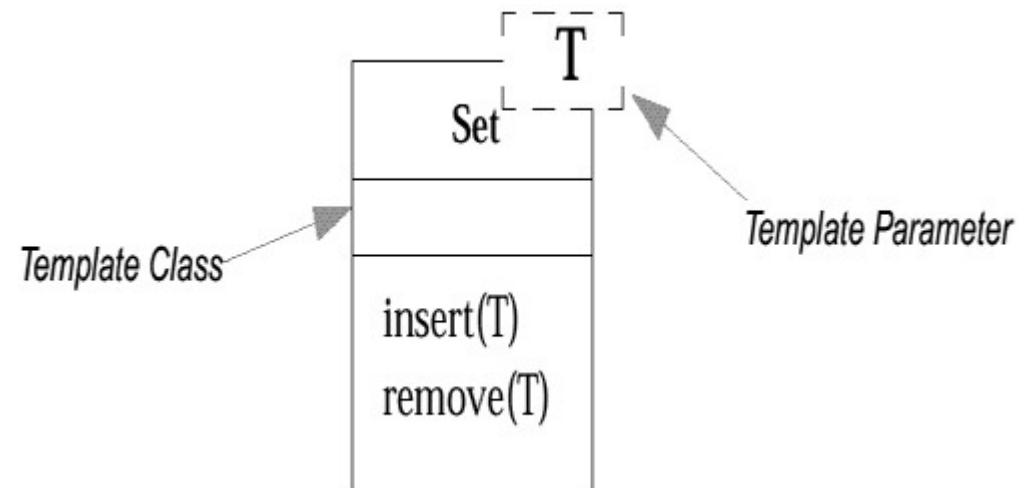
- Representação das interfaces em um diagrama de componentes



■ Classe que recebe template

```
template <class T>
class Set {
    ...
};

class X : public Y {
private:
    ...
};
```



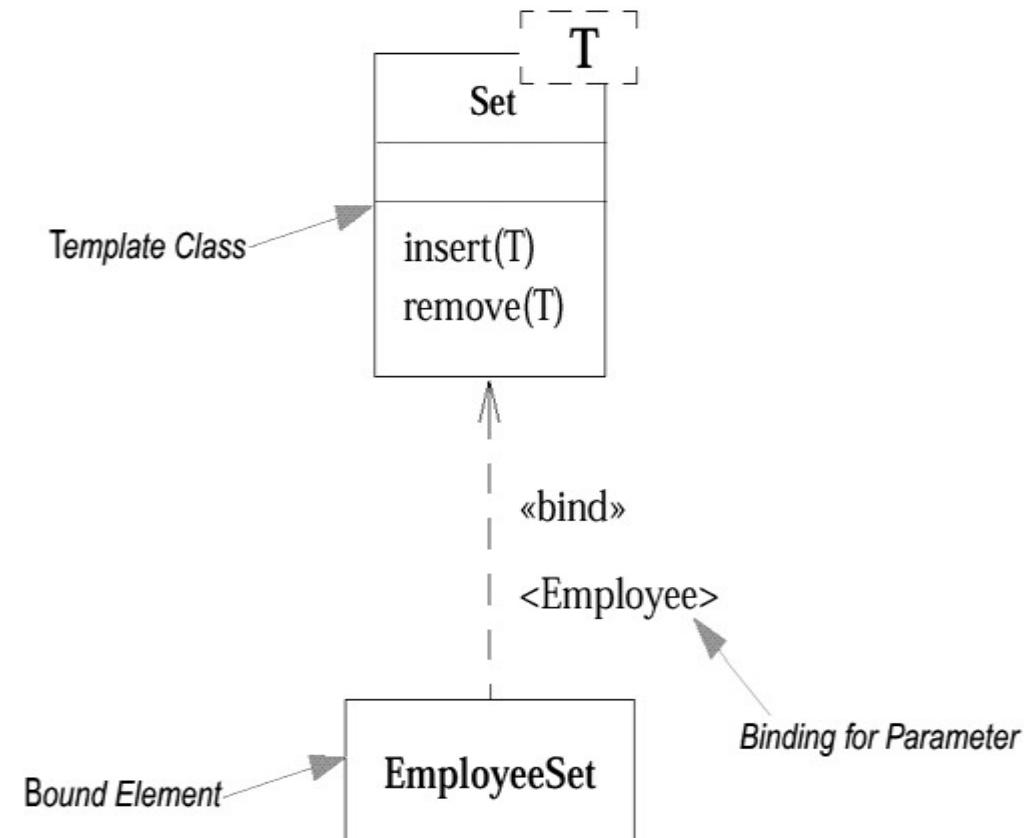
■ Representar o uso da classe parametrizada

```
Set <Employee> employeeSet;
```

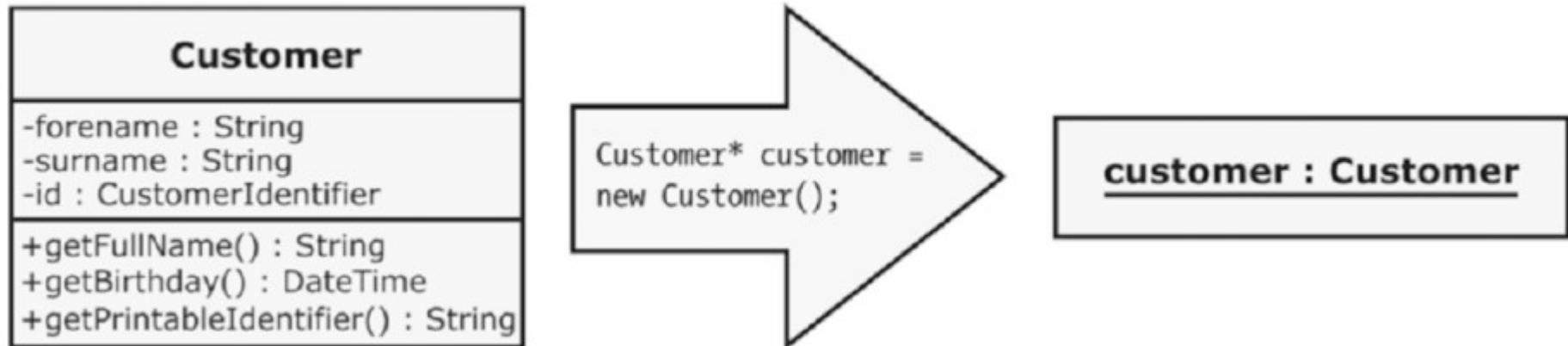
Primeira forma:

Set <Employees>

Segunda forma:



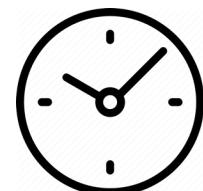
■ Representação de um objeto



- Um bom software para um sistema embarcado deve ser projetado levando em conta a possibilidade de troca de processador no futuro. Idealmente, o software de mais alto nível deve ser independente do software de mais baixo nível que lida com as questões específicas do processador.
- Quais funções ou atividades são específicas a cada processador?
 - Troca de contexto, o próprio contexto, instruções para habilitar/desabilitar interrupções, instruções atômicas, desligar o processador, etc

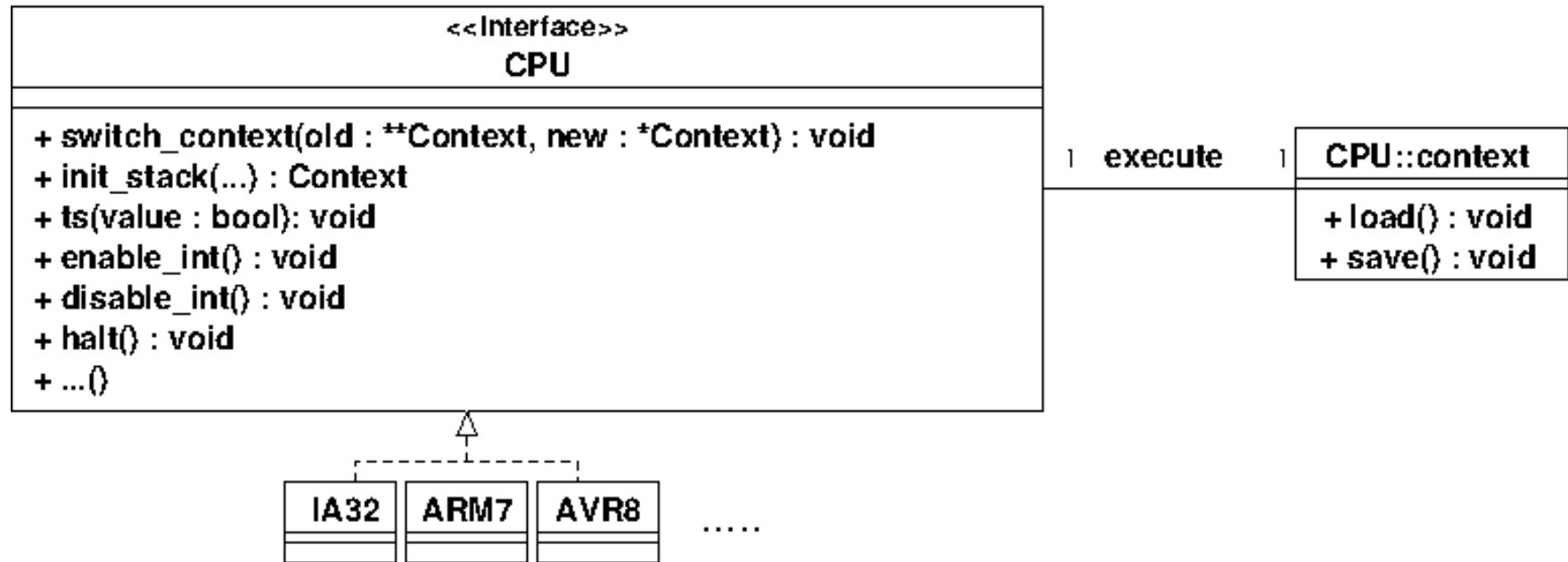
Exercício 1

- Projete o diagrama de classes para suportar diferentes processadores (CPU) de maneira que se houver a troca de CPU, o software que a usa não necessite ser reescrito.

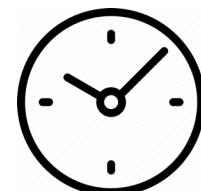


10 minutos

Exercício 1: solução

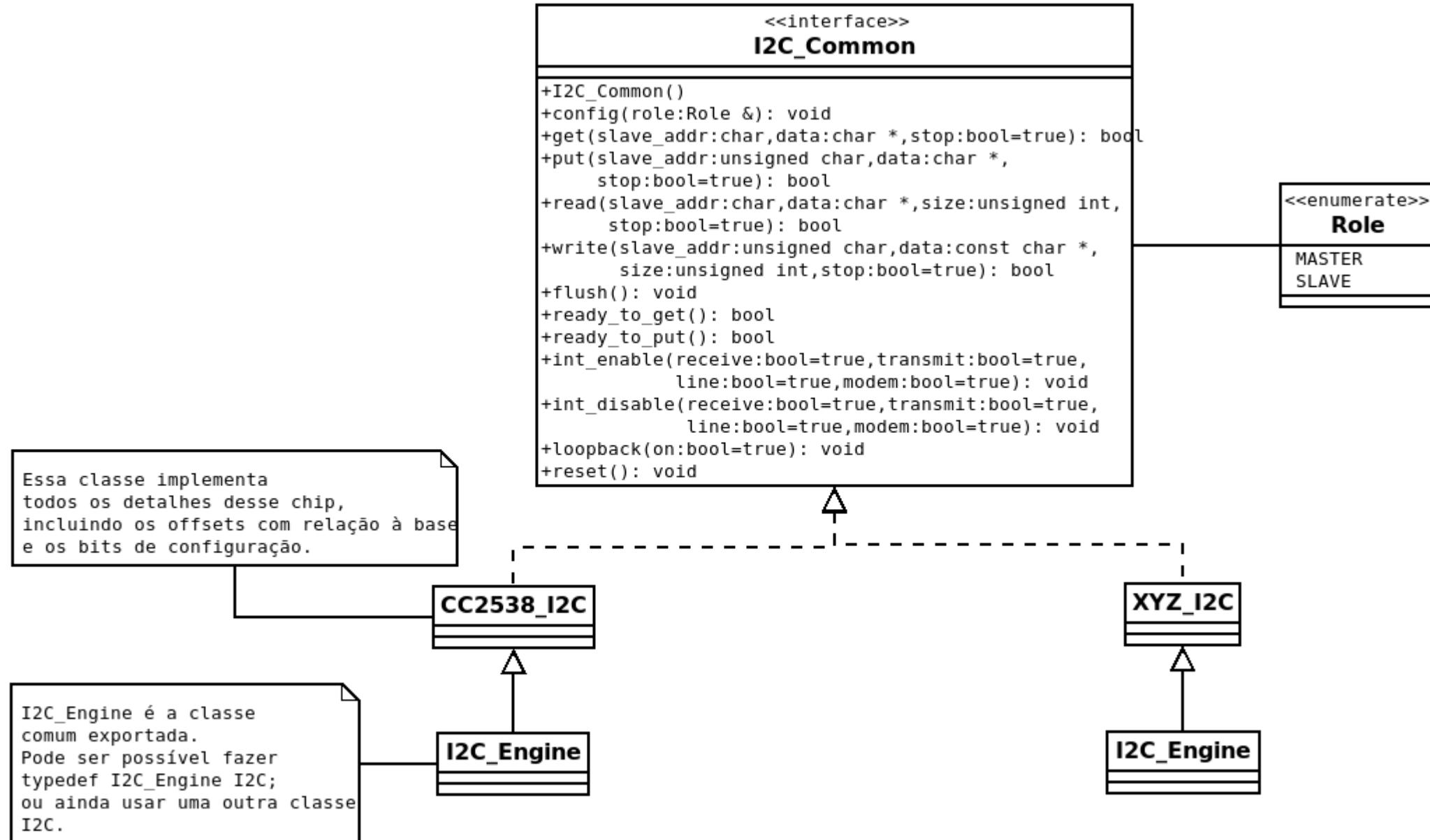


- É comum em sistemas embarcados termos diversos padrões de comunicação, como serial (UART), SPI e I2C
- Faça o projeto do software através do diagrama de classes para suportar UART, SPI e I2C
 - Quais são as operações suportadas por cada padrão de comunicação?
 - Como suportar diferentes microcontroladores?



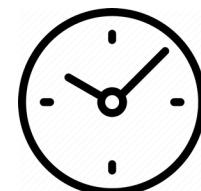
20 minutos

Exercício 2: solução



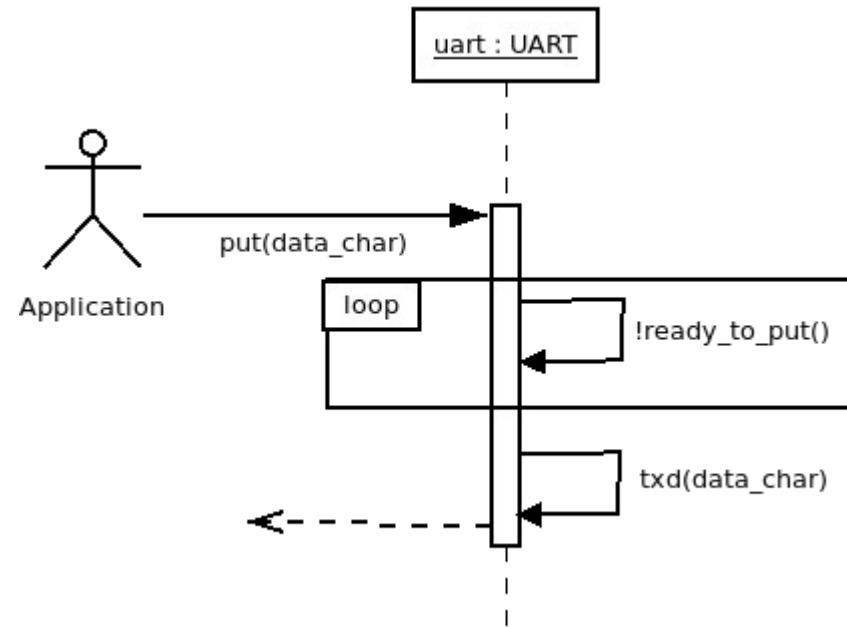
Exercício 3

- Faça o diagrama de sequência para a operação write/send/transmit da UART



15 minutos

Exercício 3: solução



Como evitar o uso de #ifdefs generalizado

- É comum quando se tem uma engenharia de software mal feita encontrar arquivos com vários #ifdefs
 - Qual o problema do #ifdef?
- Como evitar o uso generalizado de #ifdefs com C++?
 - Exemplo no código compartilhado

Referências

- <https://c4model.com/>
- <https://www.infoq.com.br/articles/C4-architecture-model/>
- <https://www.youtube.com/watch?v=x2-rSnhpw0g>
- https://www.embedded.com/wp-content/uploads/2019/11/EETimes_EMBEDDED_2019_EMBEDDED_MARKETS_STUDY.pdf
- UML for the C programming language. Bruce Powel Douglass, PhD, IBM
- <http://signal-processing.mil-embedded.com/articles/modeling-applications-uml-files-structures/>
- <https://www.drdobbs.com/cpp/uml-for-c-programmers/184401948>

Obrigado!

