Relatório de projeto
Laboratório de Desenvolvimento de Software
Licenciatura em Engenharia Informática

Aplicação demonstradora da API para aplicações com .NET Maui: Gestor de Tarefas

Equipa:

Alix Paulino, Nº 2200174

Ana Guerreiro, Nº 2103229

André Alface, Nº 2101601

Joel Ginga, Nº 2100116

Marisa Bernardo, Nº 2100446

Endereço do repositório Git:

Este relatório apresenta o projeto da aplicação demonstradora da API .Net Maui: Gestor de Tarefas, desenvolvida por esta equipa, composta pelos funcionários Alix Paulino, Ana Guerreiro, André Alface, José Ginga e Marisa Bernardo, no âmbito da empresa SimProgramming. O projeto está disponível no repositório git Bitbucket, no seguinte endereço:

https://github.com/AlixPaulino/MagicMobile

Descrição geral do projeto

O projeto propôs-se a demonstrar como desenvolver uma API .Net Maui, de acordo com o estilo arquitetónico MVC (Model-View-Controller). Foram utilizados os conceitos deste estilo arquitetónico, tendo em conta os seguintes princípios:

- Coesão forte;
- Acoplamento fraco, através de eventos e delegados;
- Reagir a problemas de forma prestável ao utilizador, através de exceções no contexto deste estilo arquitetónico;
- Separação de interesses e independência entre componentes, através do uso de uma interface, que implementamos na View.

O conceito-base da aplicação demonstradora é a gestão de tarefas, permitindo inserir uma tarefa nova ou visualizar a lista de tarefas atual, através de uma aplicação .Net Maui que permite a sua utilização em multiplataformas.

Para este projeto, a componente Model tem a responsabilidade de inicializar a lista de tarefas, quando o programa é iniciado, permitir a inserção de tarefas novas, por parte do utilizador, atualizar a lista de tarefas e ainda lançar uma exceção, que surge no caso de o utilizador pedir para visualizar a lista de tarefas e esta estar vazia.

O componente View tem a responsabilidade de exibir mensagens ao utilizador, através de uma interface (IView). As mensagens exibidas são uma mensagem de texto, o menu com as opções de escolha (interface do utilizador) e a mensagem de erro.

O componente Controller tem a responsabilidade de fazer a gestão dos fluxos de controlo da aplicação demonstradora, reagir aos inputs do utilizador (de acordo com a opção escolhida na View), reagir às exceções e ainda encerrar a aplicação.

Funcionamento esquemático

Para o desenvolvimento desta aplicação, optámos pela variante MVC proposta por Krasner & Pope (depois da revisão e análise de opções entre as duas variantes), que processa os inputs do utilizador no Controller e não prevê comunicação da View para o Controller.

O Diagrama 1 apresenta genericamente as responsabilidades de cada componente, explicitando as mensagens trocadas entre eles.

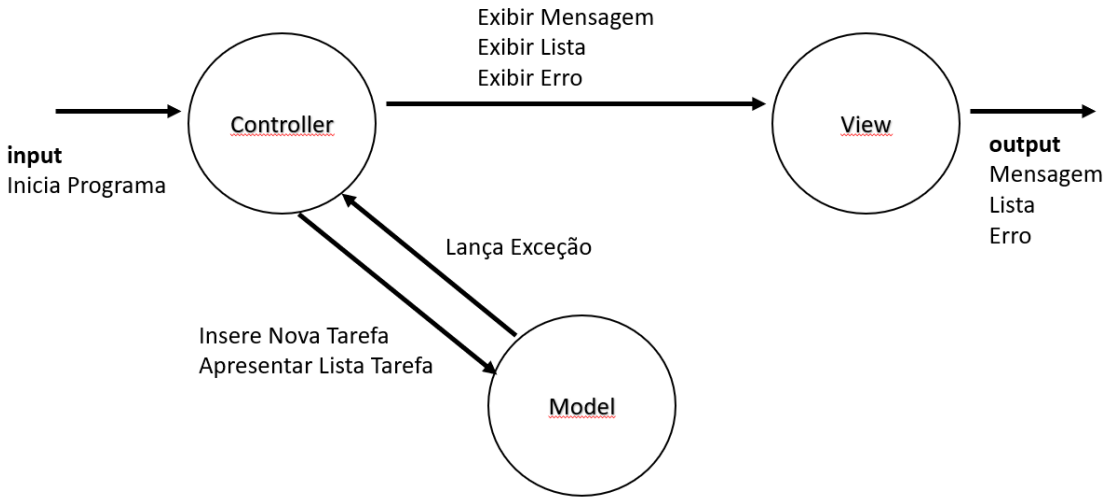

Diagrama 1: Responsabilidade de cada componente e as mensagens trocadas

Este processo inicia-se no Controller, pelo que todo o controlo da execução é gerido por este componente, logo após a inicialização geral da aplicação (atribuição de delegados a eventos, etc.).

Assim, o componente Controller toma as decisões-chave de controlo de fluxo e processa o input do utilizador (Diagrama 1).

O Controller inicia o programa e envia mensagens à View. A exibição da lista é feita através de uma interface de utilizador, que permite escolher a ação a desenvolver, a sua escolha é recebida no Controller (por ser um input) e é enviada ao Model, para a processar. O Model dá a lista de tarefas ao Controller (sem que isso seja uma ordem/mensagem e por isso não está marcado no diagrama 1, mas pode ser visto no Diagrama 2) e lança uma exceção (quando necessário, pode ser verificado no Diagrama 3). Posteriormente o Controller envia a informação atualizada da lista de tarefas e eventual erro, para que a View possa exibir a informação correta ao utilizador.

Ao desenvolver o programa procuramos promover a independência entre componentes e acabamos por não estabelecer ligações entre o Model e View. Apesar de sabermos que elas são possíveis no modelo de MVC usado, não sentimos a necessidade de usá-las.

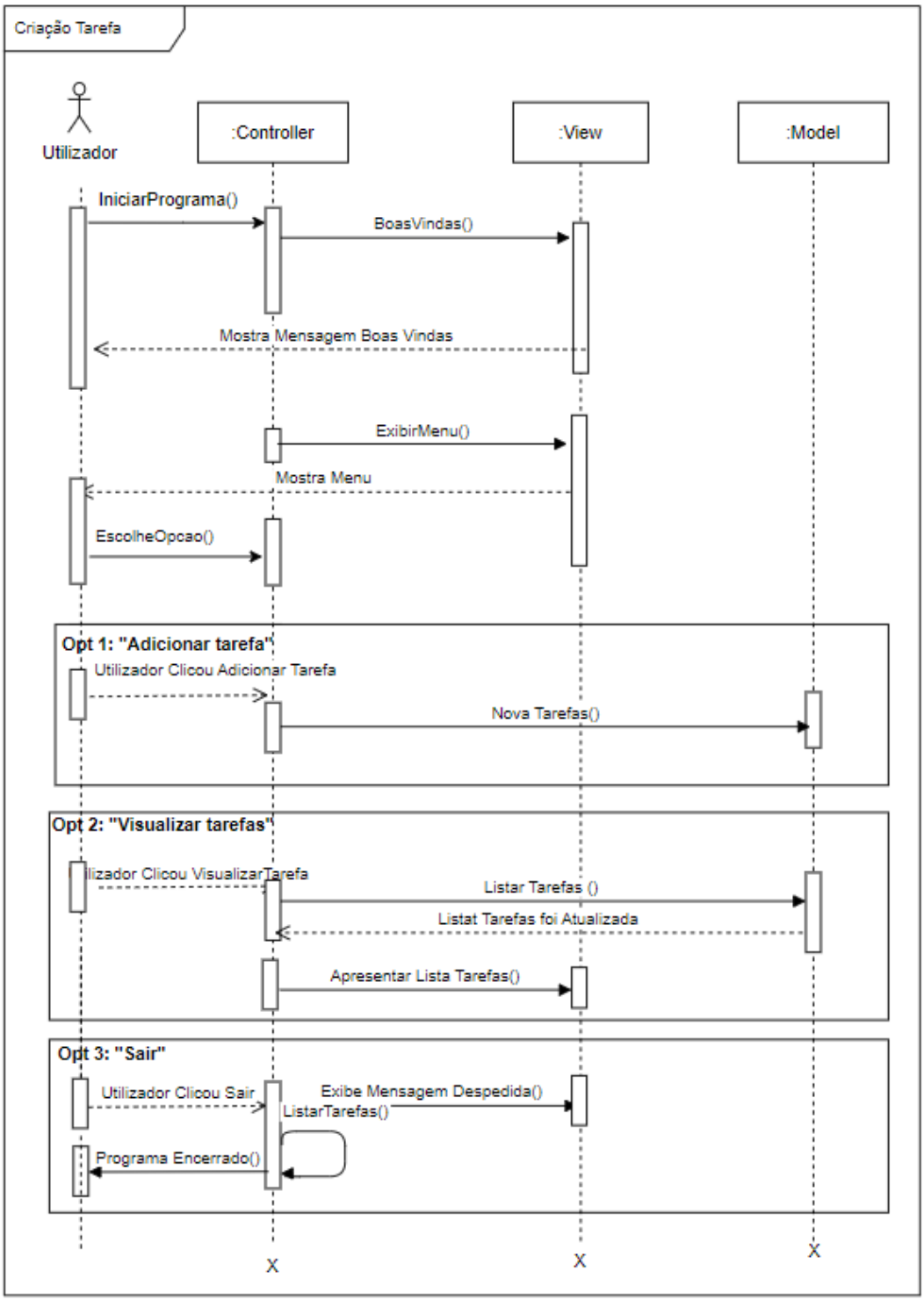

Diagrama 2: Sequência de execução para a gestão de tarefas

Com a análise do diagrama 2 é possível visualizar, de forma clara, os fluxos de comunicação entre os componentes e a forma como estruturámos o nosso código:

1. Utilizador inicia o programa, e esse comando é recebido no Controller
2. O Controller envia uma mensagem à View para exibir uma mensagem de Boas-Vindas, seguida do Menu com as opções (interface do utilizador).
3. O utilizador faz a sua escolha (1. Adicionar tarefa; 2. Visualizar tarefas; 3. Sair). A escolha é recebida pelo Controller e caso a opção seja "adicionar tarefa", tal opção também é recebida de imediato. Conforme a escolha selecionada as ações desenvolvidas são:
    a. Opção "1" (Adicionar tarefa): Controller envia os dados da tarefa inserida ao Model.
    b. Opção "2" (Visualizar tarefas): Controller solicita lista de tarefas ao Model que retorna a lista de tarefas existente ao Controller e esse faz o seu envio para que seja apresentada na View.
    c. Opção 3 (Sair): Controller envia a informação à View para apresentar a mensagem de despedida e procede ao encerramento do programa.

Divisão de responsabilidades pelos componentes

Para o desenvolvimento da nossa aplicação, uma das etapas foi a divisão das responsabilidades pelos componentes do estilo arquitetónico MVC, segundo a variante que escolhemos: Krasner & Pope, resultando na Figura 1.

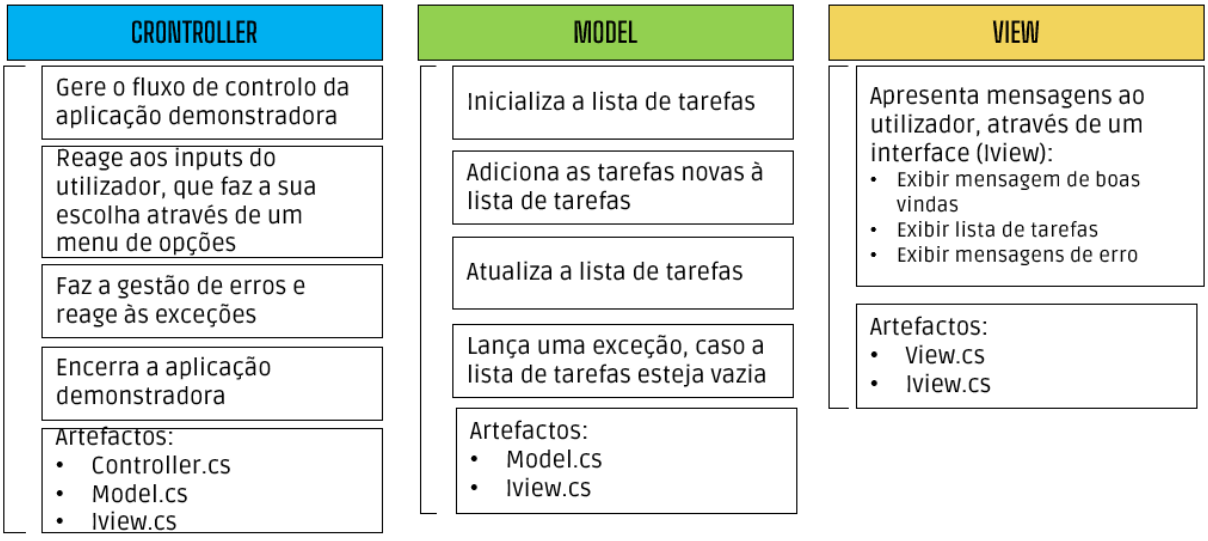

Figura 1: Divisão de responsabilidades entre componentes

Em termos de responsabilidades e classes é possível referir o seguinte:

Controller:

- Responsável por gerir o fluxo de dados e as interações entre a View e o Model.
- Iniciar o programa e controlar a sua lógica principal.
- Receber eventos da View e tomar as ações correspondentes.
- Interpretar as ações do utilizador e realizar as chamadas adequadas aos métodos do Model.
- Atualizar a View com as informações recebidas do Model.

Model:

- Responsável por armazenar os dados da aplicação demonstradora, neste caso, as tarefas.
- Fornece métodos para adicionar tarefas novas e apresentar a lista de tarefas atualizada.
- Mantém a integridade dos dados e aplica as regras de gestão das tarefas, nomeadamente o lançamento de uma exceção quando é pedida a visualização de uma lista que está vazia.
- Não interage diretamente com a View, já que ao desenvolver a nossa aplicação tomamos essa decisão por forma a simplificar o código.

View:

- Responsável por interagir com o utilizador e apresentar as informações necessárias.
- Apresentar a mensagem de boas-vindas, o menu de opções e solicitar a inserção de uma opção (inserir tarefas, visualizar tarefas ou sair) e também apresenta uma mensagem de erro (quando aplicável).
- Apresentar a lista de tarefas ao utilizador, sempre que esta é pedida
- Receber as tarefas novas do utilizador e enviá-las para o Controller.
- Não interage diretamente com o Model, já que ao desenvolver a nossa aplicação tomamos essa decisão por forma a simplificar o código.

Para além destes componentes, há outras classes relacionadas que complementam as suas responsabilidades:

- **Classe IView** é uma interface que define os métodos que a View implementa. Ela permite a abstração da implementação específica da View e possibilita a substituição fácil da implementação se necessário, promovendo uma maior independência do código e faz com que a sua manutenção e reutilização seja mais simples.

Recursos a eventos para acoplamento fraco

Para enfraquecer o acoplamento entre os componentes, foram definidos e atribuídos eventos e delegados entre o Model, a View e o Controller. A Tabela 1 detalha a interligação efetuada, indicando que componente é emissor do evento e que componente é receptor (ou seja, fornece o delegado).

	Model Emissor (ME)	Controller Emissor (CR)
Controller Receptor (CR)	(ME) Lança Exceção (CR) Encerrar	
Model Receptor (MR)		(CE) Insere Nova Tarefa (CE) Apresenta Lista Tarefas
View Receptor (VR)		(VR) Exibir Mensagem (VR) Exibir Lista (VR) Exibir Erro

Tabela 1: Componentes emissores e recetores de eventos.

Emprego de exceções, respeitando o estilo MVC

A situação de exceção considerada nesta aplicação demonstradora é a o utilizador pedir para visualizar as tarefas quando a lista de tarefas está vazia. O processamento desta exceção leva à apresentação de uma mensagem ao utilizador, que indica, de forma clara, que a lista de tarefas está vazia e que deve premir qualquer tecla para continuar, posteriormente volta a visualizar o menu com as suas opções (1. Adicionar tarefa; 2. Visualizar tarefas; 3. Sair).

O erro é lançado pelo Model ao Controller, que solicita à View que apresente, no ecrã, as mensagens referidas. A sequência de execução é a apresentada no Diagrama 3.

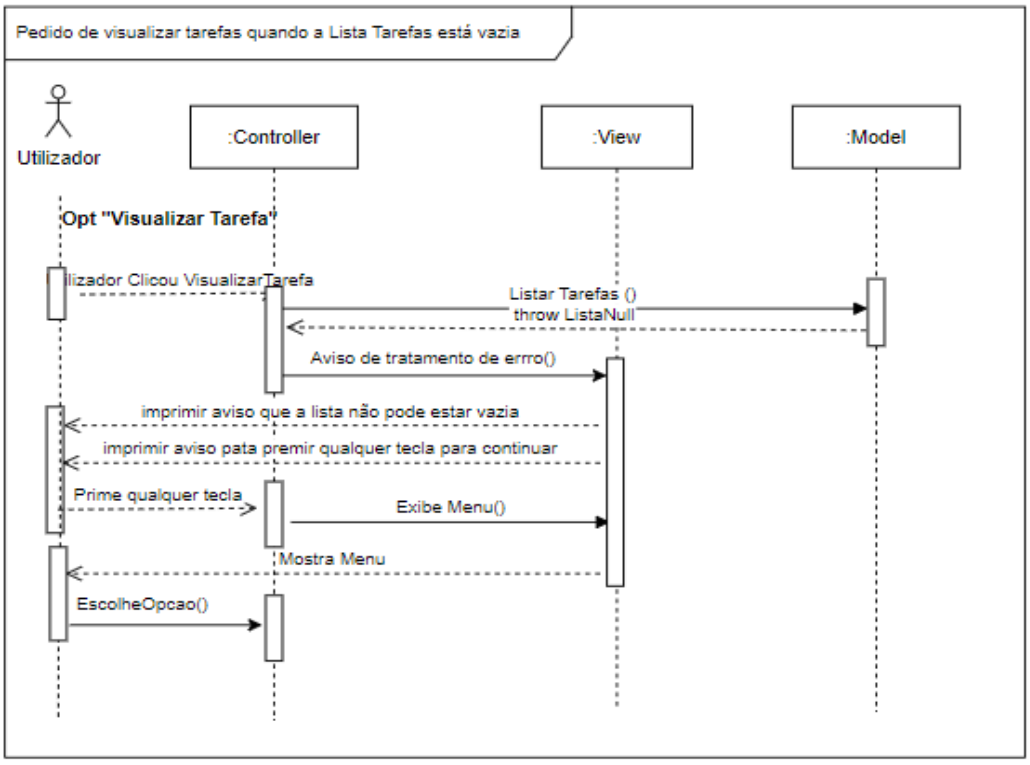

Diagrama 3: Sequência de deteção de tratamento de listas de tarefas vazias

O diagrama 3 apresenta a exceção que é lançada no Model mas apanhada no Controller, que toma por conseguinte a decisão de ativar a View para avisar o utilizador da situação. Quando o utilizador prime qualquer tecla para continuar volta a ter acesso ao Menu de opções e os passos seguintes seguem a estrutura normal do Diagrama 2 apresentado.

Nota, se o utilizador voltar a escolher a opção 2 (Visualizar tarefas) sem ter inserido nada antes, o programa volta a fazer o tratamento do erro.

Uso de interfaces para a independência entre componentes

Usamos uma interface para criar independência entre a forma de apresentar mensagens na View. Assim conseguimos aumentar a independência entre componentes e a View vai buscar a informação que necessita apenas à interface (IView) e caso no futuro venha a ser necessário acrescentar mais mensagens para serem apresentadas ao utilizador, isso poderá ser feito através da interface criada.

```
1   using System.Collections.Generic;
2
3   namespace MagicMobile
4   {
        2 referências
5       public interface IView
6       {
            2 referências
7           void ExibirMensagem(string texto);
            2 referências
8           void ExibirLista(List<string> lista);
            2 referências
9           void ExibirErro(string erro);
10      }
11  }
```

Código 1: Interface da IView

Reflexão final

De uma forma geral, consideramos que o projeto foi desafiante e que ajudou os elementos da equipa a evoluir e compreender, de uma forma abrangente, os vários passos que são necessários para o desenvolvimento de uma aplicação demonstradora com potencial real.

À primeira vista o projeto parecia ser simples, mas ao avançar através das suas etapas houve vários passos de implementação, verificação, reavaliação do que estava feito e assim sentimos que

conseguimos desenvolver uma aplicação demonstradora sólida. O seu código final foi entregue através do tópico “Mãos à massa”.

Alguns dos aspetos que consideramos terem sido essenciais para chegar ao código final apresentado foram:

Estudo dos modelos de MCV. Inicialmente tínhamos decidido usar o modelo Curry & Grace, pois a discussão de grupo sobre o tema levou-nos a acreditar que a receber os inputs na View seria mais eficiente. No entanto, ao avançar para a fase de “Mudanças no fluxo de arquitetura" avançamos com o desenvolvimento de um diagrama para o modelo Krasner & Pope e percebemos que esse modelo iria promover uma maior independência entre os componentes e assim todo o código a desenvolver seria mais simples. Por esse motivo, fizemos a alteração e passámos a usar o modelo MVC de Krasner & Pope.

Análise das dependências e uso de interfaces. Esta fase do projeto foi muito importante, já que com ela conseguimos identificar elementos que tínhamos no código e não tinham uma função real, como foi o caso do botão de animação que foi eliminado, e também conseguimos aumentar a independência da aplicação demonstradora. O uso da interface que desenvolvemos para a IView foi especialmente útil, já que permitiu que a View ficasse menos acoplada ao resto do programa e assim aumentamos a independência entre componentes, como se deseja nas boas práticas de desenvolvimento de software.

Integração do código na API .NetMaui:

Por opção decidimos avançar com o desenvolvimento do código e só fazer a sua integração na API .NetMaui no final, pois consideramos que essa seria a melhor abordagem. No entanto, quando fomos fazer essa integração percebemos que tal não era tão simples de executar como haviamos avaliado inicialmente, especialmente devido ao facto da API escolhida não possui uma arquitetura MVC.

Dessa forma optamos por programar a interface da API que irá permitir tirar o máximo de partido da mesma, no entanto, não conseguimos fazer a sua integração total.

Sentimos que se tivéssemos trabalhado nessa integração, desde o primeiro momento do desenvolvimento do código, talvez o processo tivesse sido mais simples, ou então tínhamos percebido que esta API não era a melhor opção e teríamos escolhido outra, fica a aprendizagem para projetos futuros.

Aspetos que poderiam ser melhorados:

No que respeita ao código sabemos que existem alguns aspetos que poderiam ser melhorados, nomeadamente em relação à gestão das tarefas.

Ao implementar uma opção para o utilizador inserir a data prevista para a realização da tarefa, iria possibilitar uma melhor gestão das tarefas mais eficazes.

Implementar uma opção para eliminar tarefas já concluídas, iria permitir aceder a listas de tarefas sempre atualizadas.

Por fim, o tratamento de erro que identificámos também poderia ser melhorado já que quando é lançado o erro e o utilizador clica em qualquer tecla para continuar, e nesse momento é apresentado o menu de opções normal. Se optássemos por apresentar um menu especial, onde eram apresentadas apenas as opções de “Adicionar tarefa” e “Sair” seria possível evitar que o utilizador voltasse a despoletar o mesmo erro de novo.

Palavras finais:

No que respeita à integração da API em .NetMaui sentimos que, com uma maior disponibilidade de tempo, teríamos conseguido avançar mais neste ponto. No entanto, e tendo em conta que o objetivo deste projeto era apresentar uma aplicação demonstradora e não um produto final, pensamos que conseguimos apresentar um projeto que foi evoluindo, à medida que entranhávamos os novos conceitos e acreditamos que a versão final demonstra as boas práticas de desenvolvimento de software.