1. **Características das histórias (INVEST):**

- **Independente (Independent):** A história deve ser independente de outras histórias, de forma que possa ser implementada e testada isoladamente.
- **Negociável (Negotiable):** A história deve ser aberta a discussões e negociações entre o time de desenvolvimento e o cliente, de forma que possa ser adaptada conforme as necessidades e prioridades do projeto.
- **Valiosa (Valuable):** A história deve ter valor para o usuário ou para o negócio, de forma que traga benefícios ou soluções para um problema real.
- **Estimável (Estimable):** A história deve ser clara o suficiente para que o time de desenvolvimento possa estimar o tempo e recursos necessários para implementá-la.
- **Pequena (Small):** A história deve ser pequena o suficiente para que possa ser implementada em um curto período de tempo, permitindo um fluxo constante de entregas.
- **Testável (Testable):** A história deve ser testável, de forma que o time de desenvolvimento possa verificar se ela foi implementada corretamente e atende aos requisitos do usuário.

2. **Acionar validação do PO:**

Para acionar a validação das histórias de usuário pelo Product Owner (PO), é importante que o PO realize uma revisão criteriosa das histórias antes de aceitá-las para o backlog do projeto. Durante essa revisão, o PO deve verificar se as histórias atendem aos critérios INVEST e se estão alinhadas com os objetivos e requisitos do projeto.

Uma forma de acionar a validação das histórias pelo PO é definir um processo formal de revisão, no qual o PO deve analisar cada história submetida pelo time de desenvolvimento antes de adicioná-la ao backlog do projeto. Durante essa revisão, o PO pode realizar perguntas para entender melhor as necessidades do usuário e verificar se as histórias atendem aos critérios INVEST. Além disso, o PO pode definir um conjunto de critérios adicionais específicos para o projeto, de forma a garantir que as histórias atendam aos requisitos do negócio.

Outra forma de acionar a validação das histórias pelo PO é estabelecer um fluxo de comunicação constante entre o PO e o time de desenvolvimento. Dessa forma, o PO pode acompanhar o progresso do

desenvolvimento das histórias, fornecer feedbacks e verificar se as histórias estão sendo implementadas de acordo com as expectativas. O PO também pode aproveitar esse fluxo de comunicação para acionar a validação de histórias específicas, solicitando ao time de desenvolvimento que forneça mais informações ou evidências de que a história foi implementada corretamente.

Em resumo, para acionar a validação das histórias de usuário pelo PO, é importante estabelecer um processo formal de revisão e um fluxo de comunicação constante entre o PO e o time de desenvolvimento. Isso ajuda a garantir que as histórias atendam aos critérios INVEST e aos requisitos do projeto, maximizando o valor entregue ao usuário ou ao negócio.

3. Características de protótipos de baixa fidelidade

Os protótipos de baixa fidelidade são modelos simplificados e rápidos de serem criados, geralmente feitos à mão ou usando ferramentas simples de design, como papel, canetas, lápis, post-its, entre outros. Esses protótipos são utilizados para testar conceitos, fluxos de interação e funcionalidades de uma interface de forma rápida e econômica. Algumas características comuns de protótipos de baixa fidelidade incluem:

1. Baixo nível de detalhamento: os protótipos de baixa fidelidade são geralmente simples e apresentam poucos detalhes, com o objetivo de testar rapidamente ideias e conceitos.

2. Materiais de baixo custo: materiais como papel, canetas, post-its e outros materiais de baixo custo são frequentemente utilizados na criação de protótipos de baixa fidelidade.

3. Design esboçado: os protótipos de baixa fidelidade geralmente têm um design esboçado, com desenhos rápidos e informais que representam as ideias e conceitos da interface.

4. Interface limitada: os protótipos de baixa fidelidade são geralmente limitados em termos de interface e funcionalidades, apresentando apenas as principais interações e fluxos da interface.

5. Fácil de modificar: como são protótipos rápidos e simples, é fácil modificá-los conforme necessário. Isso permite que o designer faça alterações rápidas e fáceis para testar diferentes conceitos e ideias.

6. Testes de usabilidade: os protótipos de baixa fidelidade são frequentemente usados em testes de usabilidade, onde os usuários interagem com o protótipo para fornecer feedback sobre a interface e suas funcionalidades.

Em resumo, as características dos protótipos de baixa fidelidade incluem um baixo nível de detalhamento, materiais de baixo custo, design esboçado, interface limitada, facilidade de modificação e utilização em testes de usabilidade. O objetivo desses protótipos é permitir que os designers testem rapidamente ideias e conceitos de interface de forma econômica e fácil de modificar.

4. Diferença entre protótipo de baixa e protótipo de média

Os protótipos de baixa e média fidelidade se diferenciam principalmente em relação ao nível de detalhamento e realismo que apresentam. Enquanto os protótipos de baixa fidelidade são modelos simplificados e rápidos de serem criados, os protótipos de média fidelidade apresentam um nível de detalhamento maior e uma representação mais próxima da interface final. Algumas das principais diferenças entre esses dois tipos de protótipos são:

1. Nível de detalhamento: os protótipos de média fidelidade apresentam um nível de detalhamento maior do que os protótipos de baixa fidelidade, com elementos mais elaborados e próximos da interface final.

2. Materiais e ferramentas: enquanto os protótipos de baixa fidelidade geralmente são criados com materiais de baixo custo e ferramentas simples de design, como papel e canetas, os protótipos de média fidelidade utilizam ferramentas de design mais avançadas, como softwares de prototipagem e design gráfico.

3. Realismo: os protótipos de média fidelidade apresentam um nível de realismo maior do que os protótipos de baixa fidelidade, com elementos mais próximos da interface final em termos de tamanho, cor e proporção.

4. Interação: os protótipos de média fidelidade podem incluir interações mais complexas do que os protótipos de baixa fidelidade, permitindo que o usuário teste fluxos de interação mais elaborados.

5. Finalidade: os protótipos de média fidelidade são geralmente usados em testes mais avançados de usabilidade e validação de conceitos, enquanto os protótipos de baixa fidelidade são usados principalmente para testar ideias e conceitos iniciais.

Em resumo, os protótipos de média fidelidade apresentam um nível de detalhamento, realismo e interatividade maior do que os protótipos de baixa fidelidade, além de serem criados com ferramentas mais avançadas e serem usados em testes mais avançados de usabilidade e validação de conceitos. Por outro lado, os protótipos de baixa fidelidade são mais econômicos e fáceis de modificar, sendo ideais para testar ideias e conceitos iniciais.

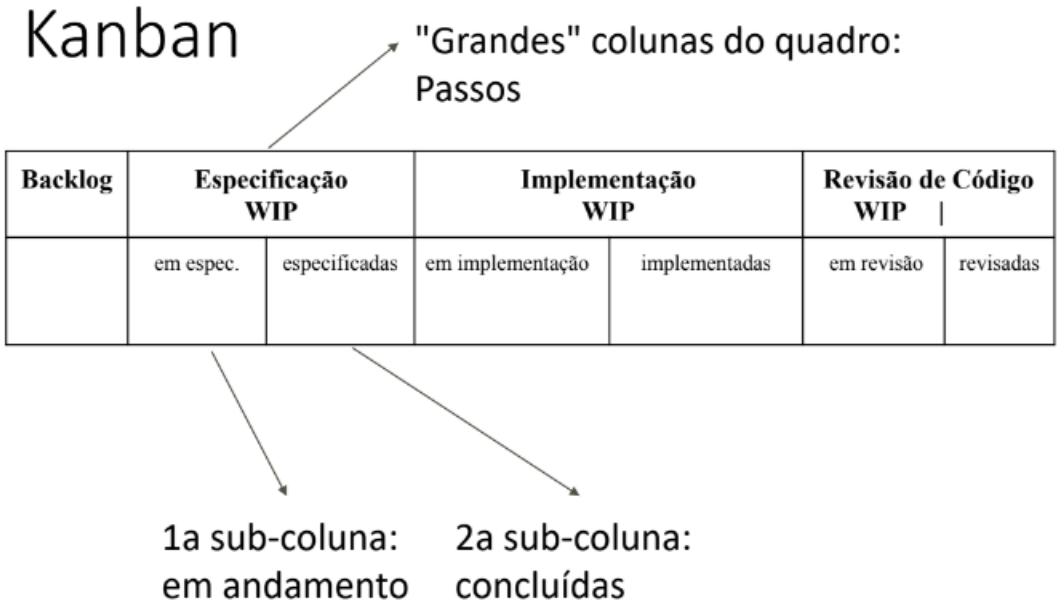
5. Kanban

WIP

WIP (Work In Progress) é uma sigla usada em metodologias ágeis, como o Kanban, para indicar o número de tarefas em andamento em um determinado momento. Em um quadro Kanban, o WIP limita o número de tarefas que podem ser trabalhadas simultaneamente em uma determinada etapa do fluxo de trabalho.

O objetivo do WIP limitado é evitar o acúmulo excessivo de tarefas em uma etapa do fluxo de trabalho, o que pode levar a atrasos e gargalos no processo. Com um WIP limitado, a equipe é incentivada a concluir as tarefas em andamento antes de iniciar novas, o que aumenta a eficiência e o fluxo do trabalho.

O WIP é uma medida importante no quadro Kanban, pois permite que a equipe visualize a quantidade de trabalho em andamento e identifique possíveis gargalos no processo. Limitar o WIP também ajuda a manter o foco e a produtividade da equipe, evitando sobrecarga de trabalho e estresse desnecessário.



BDD

BDD (Behavior-Driven Development)[1] é uma metodologia de desenvolvimento de software que se concentra na linguagem natural e nas interações entre as pessoas envolvidas no projeto, incluindo desenvolvedores, testadores, gerentes de produto e outros stakeholders. O objetivo do BDD é melhorar a colaboração e a comunicação entre as equipes, fornecendo um processo claro e compreensível para a criação de software.

O BDD utiliza uma linguagem de domínio específico (DSL) para descrever o comportamento esperado do software em termos de cenários de uso realistas. Esses cenários são escritos em uma linguagem simples e acessível para todos os membros da equipe e são usados para definir os requisitos do software. Essa abordagem ajuda a garantir que todos os membros da equipe tenham uma compreensão clara dos requisitos e do comportamento esperado do software.

[1] BDD significa "Desenvolvimento Guiado por Comportamento" em português. Trata-se de uma metodologia de desenvolvimento de software que se concentra no comportamento esperado do software em termos de cenários de uso realistas descritos em linguagem natural. Esses cenários são usados para definir os requisitos do software e para criar testes automatizados baseados em linguagem natural. O objetivo do BDD é melhorar a colaboração e a comunicação entre as equipes de desenvolvimento, testes e negócios, garantindo que todos os membros da equipe tenham uma compreensão clara dos requisitos e do comportamento esperado do software.

O BDD enfatiza a criação de testes automatizados baseados nos cenários descritos em linguagem natural, que são executados continuamente durante o processo de desenvolvimento para garantir que o software esteja funcionando conforme o esperado. Isso ajuda a detectar problemas e erros de maneira precoce, reduzindo a probabilidade de defeitos mais graves surgirem posteriormente no processo de desenvolvimento.

Em resumo, o BDD é uma metodologia ágil que enfatiza a colaboração e a comunicação entre as equipes e a criação de testes automatizados baseados em cenários de uso realistas descritos em linguagem natural. O objetivo é fornecer um processo claro e compreensível para a criação de software que ajuda a garantir que todos os membros da equipe tenham uma compreensão clara dos requisitos e do comportamento esperado do software.

QA

No contexto do Kanban, QA é a sigla para Quality Assurance, ou seja, Garantia da Qualidade em português. O papel do QA no Kanban é assegurar que o produto ou serviço entregue atenda aos requisitos de qualidade previamente estabelecidos. Isso envolve o teste e a validação do produto ou serviço em todas as etapas do processo de produção, desde o início do desenvolvimento até a entrega final.

O QA no Kanban é responsável por garantir que os requisitos do cliente sejam cumpridos e que o produto ou serviço esteja livre de defeitos e erros. O QA trabalha em conjunto com os membros da equipe de desenvolvimento e ajuda a identificar possíveis problemas e a implementar melhorias para aumentar a qualidade do produto ou serviço.

Em resumo, o QA no Kanban é um importante papel que ajuda a manter a qualidade do produto ou serviço entregue, assegurando que o mesmo esteja em conformidade com as expectativas do cliente e com os requisitos de qualidade estabelecidos.

E2E

"E2E" significa "end-to-end" e se refere a uma abordagem de teste ou de solução de problemas que avalia todo o processo ou sistema em sua totalidade, desde o início até o fim, em vez de avaliar apenas partes individuais do processo ou sistema.

No contexto do Kanban, a abordagem E2E pode ser aplicada para garantir que todo o fluxo de trabalho, desde a criação até a entrega do produto final, seja avaliado e otimizado para garantir a qualidade e a eficiência em cada etapa. Dessa forma, a equipe de Kanban pode identificar e solucionar problemas em qualquer ponto do processo que possa afetar a qualidade do produto ou serviço entregue ao cliente.

A abordagem E2E também pode ser aplicada em testes de software, onde testes são realizados em todo o sistema para verificar se os componentes individuais funcionam corretamente em conjunto. Isso ajuda a garantir que o software funcione corretamente em diferentes cenários de uso e em diferentes ambientes.

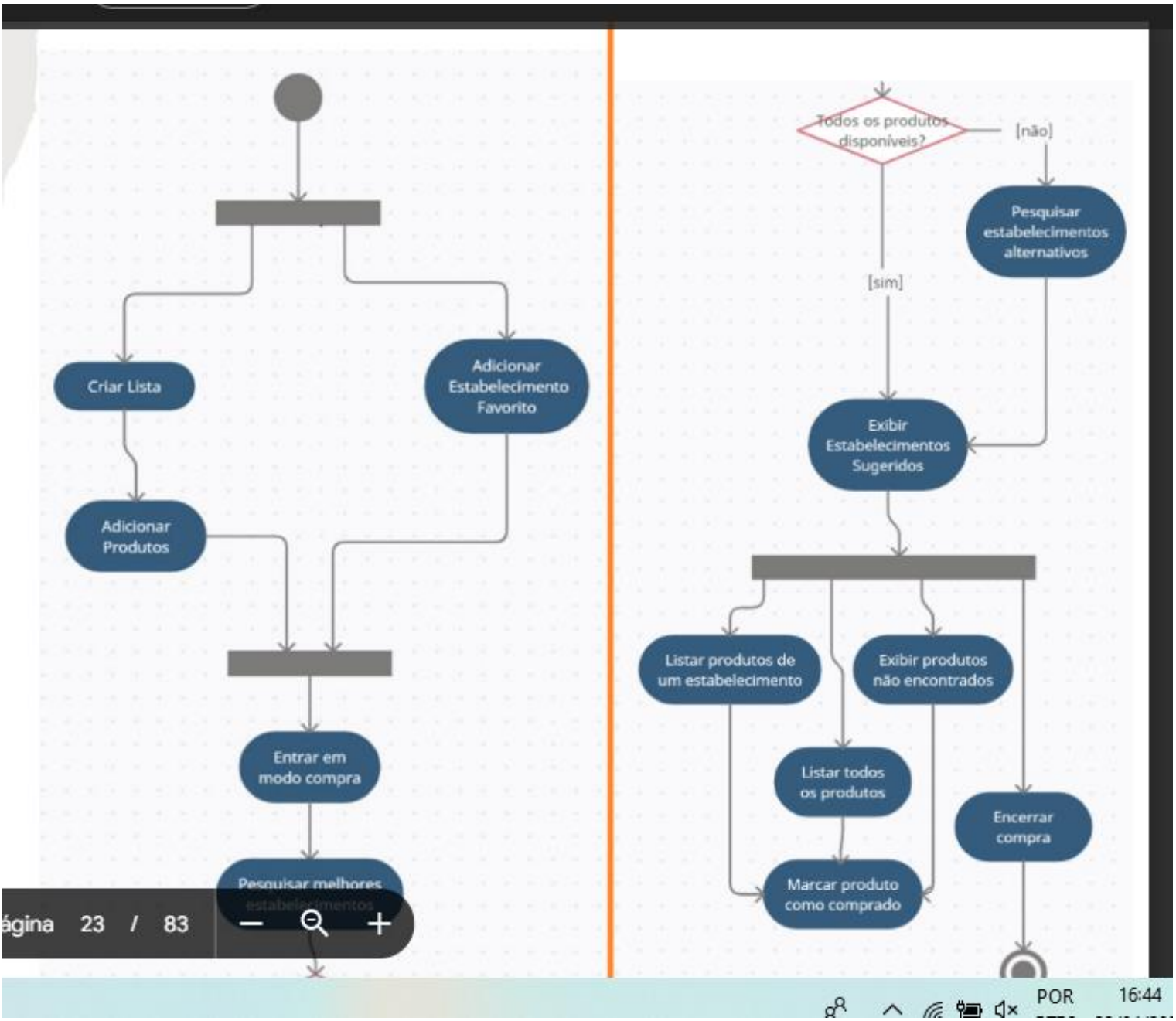
Backlog	Lo-Fi	BDD (especific)	Implement.	Testar(E2E)	Verifiq. (QA)	Done

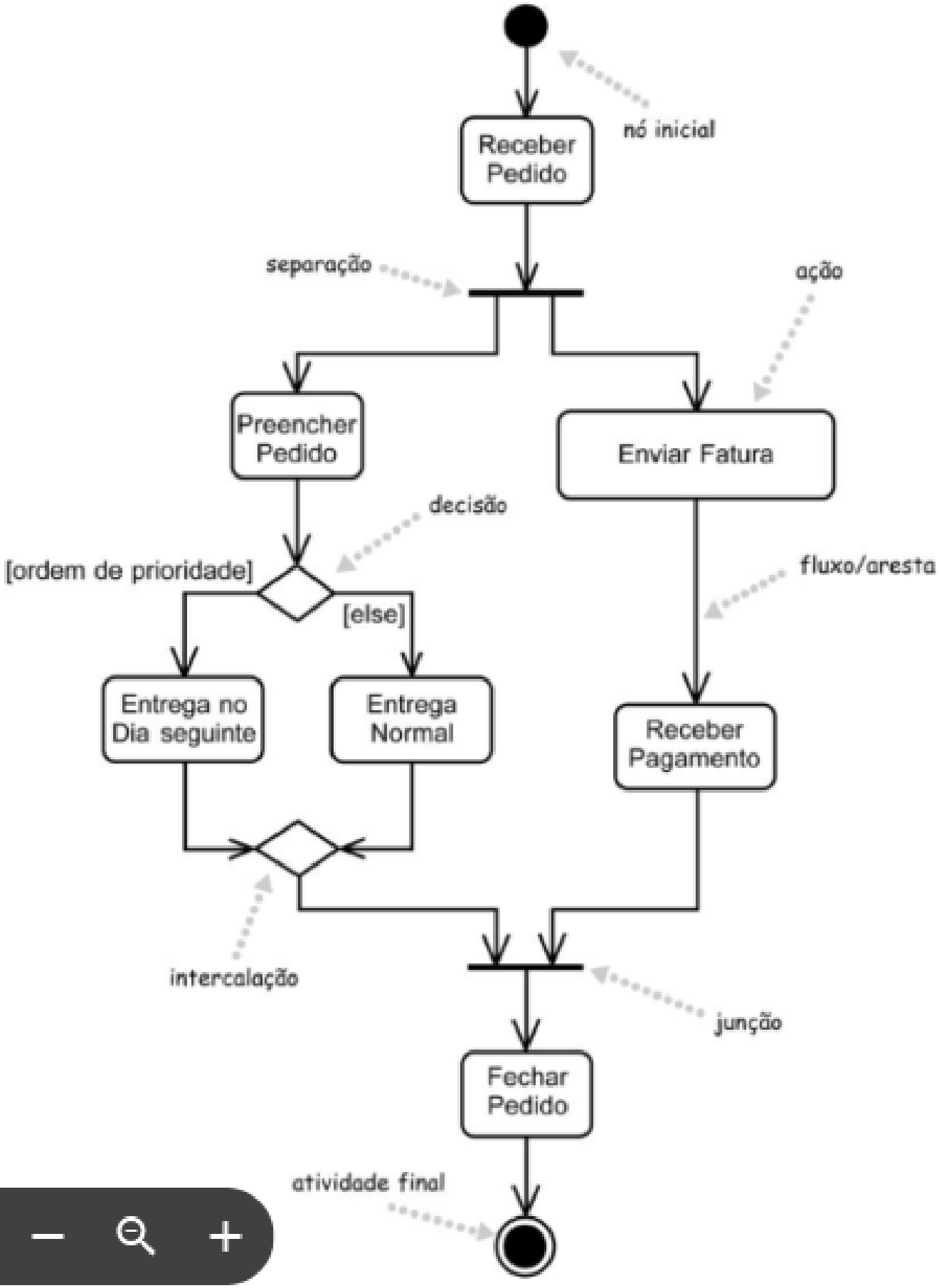
Diagrama de classes UML

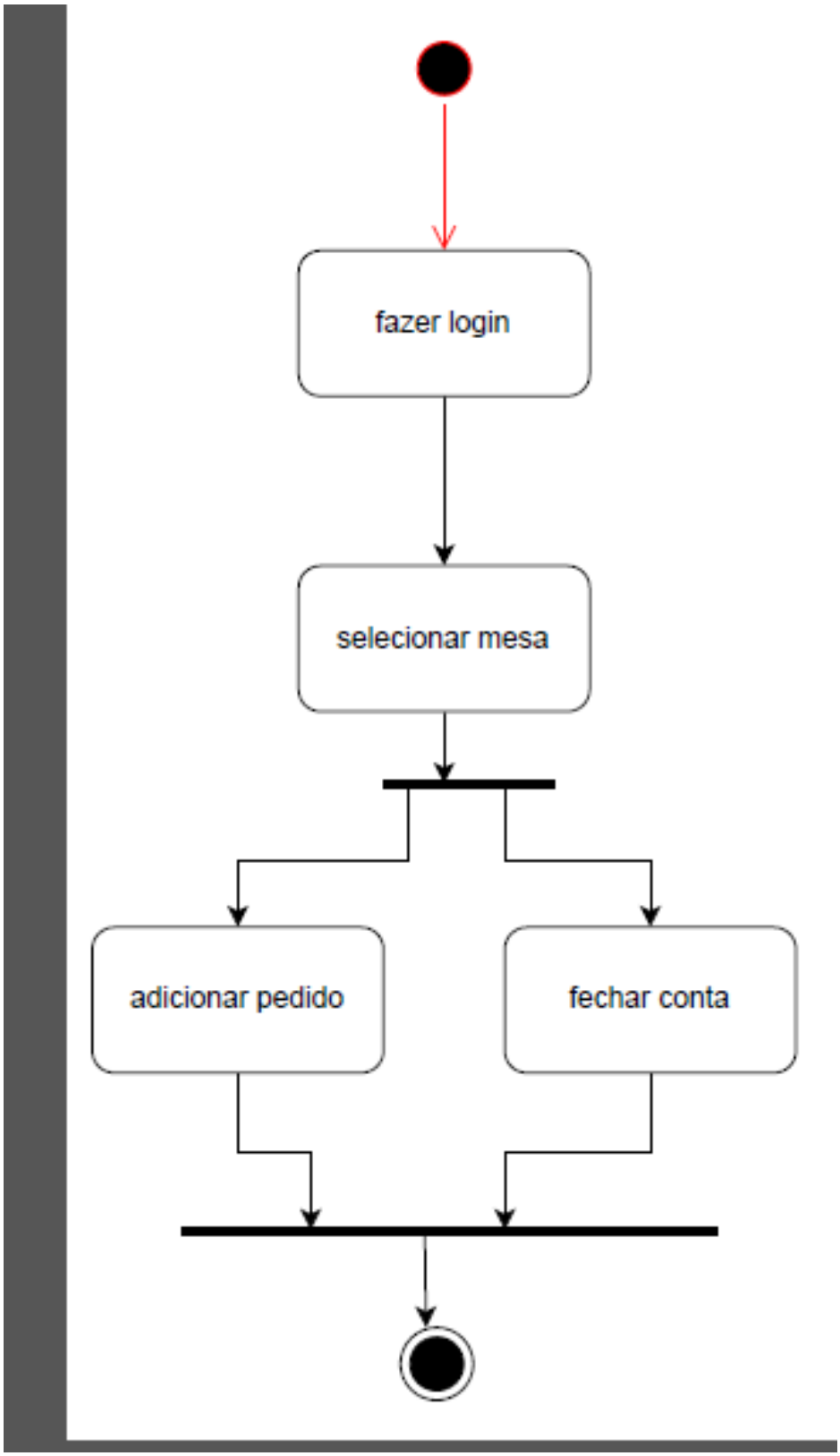
Diagrama de classes é uma das principais ferramentas da Linguagem de Modelagem Unificada (UML) que permite visualizar a estrutura de um sistema orientado a objetos. O diagrama de classes é utilizado para descrever as classes do sistema, seus atributos, métodos e relacionamentos com outras classes.

As classes são representadas por retângulos que contêm o nome da classe, seus atributos e métodos. Os atributos são representados por nomes e tipos de dados, enquanto os métodos são representados por seus nomes, parâmetros e tipo de retorno. As relações entre as classes são representadas por linhas que conectam os retângulos das classes. Existem vários tipos de relações, como associação, composição, agregação, herança, entre outras.

O diagrama de classes é uma ferramenta muito útil para entender a estrutura de um sistema orientado a objetos e também para projetar novos sistemas. Além disso, ele pode ser utilizado em conjunto com outras ferramentas da UML para descrever diferentes aspectos do sistema, como casos de uso, sequência de operações, entre outros.


Criar Lista
Adicionar Estabelecimento Favorito
Adicionar Produtos
Entrar em modo compra
Pesquisar melhores
Todos os produtos disponíveis?
[não]
[sim]
Pesquisar estabelecimentos alternativos
Exibir Estabelecimentos Sugeridos
Listar produtos de um estabelecimento
Exibir produtos não encontrados
Listar todos os produtos
Encerrar compra
Marcar produto como comprado


nó inicial
Receber Pedido
separação
ação
Preencher Pedido
Enviar Fatura
decisão
[ordem de prioridade]
[else]
fluxo/aresta
Entrega no Dia seguinte
Entrega Normal
Receber Pagamento
intercalação
junção
Fechar Pedido
atividade final



Junção (linha preta) != decisão (losango). Se não tiver decisão, não é losango.

Começa com uma bola preta. E termina com uma bola preta com um círculo em volta.