

Levantamento e análise de requisitos

a) Quais diagramas UML serão essenciais (nesta fase)

Nesta fase, usaremos **apenas**:

1. **Diagrama de Casos de Uso** — mapa de atores e funcionalidades.
2. **Diagrama de Atividade** — fluxos de negócio detalhados por caso de uso crítico.

b) Justificativa detalhada de cada diagrama

1) Diagrama de Casos de Uso

- **Comunicação com stakeholders:** apresenta de forma visual e intuitiva quem usa o sistema e o que o sistema deve fazer (visão funcional), facilitando alinhamento com donos de negócio, gerentes e clientes.
- **Captura de requisitos funcionais:** cada caso de uso é um requisito funcional (ex.: Agendar Serviço), que pode virar história/entrada no backlog.
- **Priorização e escopo:** permite priorizar casos de uso (ex.: MVP = Agendamento, Check-in, Lavagem, Pagamento).
- **Base para testes de aceitação:** casos de uso servem como cenários de aceitação.
- **Baixo custo de criação:** fácil de entender por não-técnicos — ideal no começo do projeto.

Como ajuda no sistema de lavação de carros:

- Mostra ações essenciais: Agendar Serviço, Registrar Novo Veículo, Check-in do Veículo, Executar Lavagem, Finalizar Serviço, Pagar Serviço, Emitir Comprovante/Nota, Gerar Relatórios, Gerenciar Funcionários.
- Relaciona atores: Cliente (mobile/web), Repcionista/Operador, Administrador, Caixa, Sistema de Pagamento (externo), Aplicativo de Notificações (SMS/Email).
- Ex.: Caso de uso Agendar Serviço — mostra que Cliente interage; pode ter extensão Confirmar Agendamento (SMS) e inclusão Autenticar Cliente.

Nível de detalhamento recomendado nesta fase:

- Diagrama de alto nível + descrição textual curta por caso de uso (precondição, fluxo principal, fluxos alternativos e pós-condição). Não modelar atributos de classes.

2) Diagrama de Atividade

- **Visualizar fluxos de trabalho** (passo a passo) — ideal para traduzir casos de uso em sequências de ações de negócio.
- **Identificar decisões e exceções** (ramos de decisão, tratamentos de erro, cancelamentos).
- **Apoiar desenvolvedores e analistas na compreensão do comportamento** sem entrar ainda em estrutura de código.

- **Servir de base para estimativa e automação de processos** (onde automatizar e onde permanece manual).

Como ajuda no sistema de lavação de carros:

- **Fluxo “Agendar Serviço”**: selecionar tipo (simples / completa), escolher data/hora, selecionar veículo, calcular tempo e preço, confirmar pagamento ou selecionar pagamento no local, enviar confirmação (SMS/Email). O diagrama mostra pontos de decisão (ex.: vaga disponível?), tempo limite para confirmação, alternativas (cancelamento).
- **Fluxo “Check-in / Recebimento”**: chegada do cliente → valida agendamento → registrar entrada do veículo → atribuir vaga → iniciar processo de lavagem. Inclui ações paralelas (preparar equipamento, abrir ordem de serviço).
- **Fluxo “Processo de Lavagem”**: sequência de etapas (pré-lavagem → lavagem externa → aspiração interna → secagem → inspeção) com possíveis ramificações (solicita revisão adicional, cliente autorizou pintura?).
- **Fluxo “Pagamento e Finalização”**: escolha do método de pagamento (Cartão, Dinheiro, PIX), processamento externo, emitir comprovante, liberar veículo.

Nível de detalhamento recomendado nesta fase:

- Diagramas de atividade **por cada caso de uso crítico** (Agendamento; Check-in/Recepção; Processo de Lavagem; Pagamento/Finalização). Incluir fluxos alternativos mais comuns (cancelamento, reagendamento, pagamento recusado). Evitar detalhes de implementação (não modelar threads, não mapear chamadas de API internas).

c) Diagramas UML provavelmente deixados de lado (nesta fase inicial)

Os diagramas que **não** serão criados agora (lista):

- Diagrama de Classes
- Diagrama de Sequência
- Diagrama de Colaboração
- Diagrama de Estados / Máquina de Estados
- Diagrama de Componentes
- Diagrama de Implantação
- Diagrama de Objetos
- Diagrama de Pacotes
- Diagrama de Estrutura Composta
- Diagrama de Perfil
- Diagrama de Tempo
- Diagrama de Visão de Interação

d) Justificativa da omissão (por que não são necessários nesta fase)

(Organizado por diagrama — motivo objetivo)

- **Diagrama de Classes**

- Motivo: modelagem de domínio e atributos detalhados pertence à fase de projeto de alto nível / detalhado. Criar classes agora força decisões de implementação prematuras e consome tempo sem valor imediato para stakeholders não-técnicos.

- **Diagrama de Sequência**

- Motivo: descreve chamadas entre objetos/partes do sistema ao nível de código/integração. Na fase de requisitos ainda não definimos objetos/serviços concretos nem interfaces de implementação. Melhor postergar até definir a arquitetura/domínio.

- **Diagrama de Colaboração**

- Motivo: é uma alternativa às sequências para mostrar interações entre objetos; mesma justificativa das Sequência — detalhe de design desnecessário neste momento.

- **Diagrama de Estados**

- Motivo: útil quando um objeto tem comportamento complexo baseado em estados (ex.: pedido de e-commerce). No sistema de lavação, estados existiriam (Agendado → Em Atendimento → Em Lavagem → Finalizado → Pago), mas para requisitos de negócio um **diagrama de atividade** cobre o fluxo suficiente. State machines serão criadas apenas para entidades cujo comportamento interno justificarem (ex.: OrdemDeServiço) em fases posteriores.

- **Diagrama de Componentes**

- Motivo: mostra organização de software em módulos/bibliotecas. Relevante após decidir arquitetura. Nesta fase a arquitetura é opcional e deve vir só após análise de requisitos e dimensionamento.

- **Diagrama de Implantação**

- Motivo: descreve nós físicos. Só necessário na fase de implantação quando infraestrutura concreta for escolhida.

- **Diagrama de Objetos**

- Motivo: pré-visualização de instâncias em determinado momento; é útil durante modelagem detalhada ou para debugging. No levantamento de requisitos pouco valor.

- **Diagrama de Pacotes**

- Motivo: organiza o modelo de classes em módulos; só relevante depois de modelar classes.

- **Diagrama de Estrutura Composta**

- Motivo: descreve colaboração interna de classes/partes no nível de implementação — desnecessário agora.

- **Diagrama de Perfil**

- Motivo: usado para extensões UML específicas (ex.: adaptar UML a domínio). Só se houver necessidade de estender metamodelo — raro em projetos padrão.
- **Diagrama de Tempo**
 - Motivo: focados em comportamentos temporais e interações complexas; custo/benefício baixo no levantamento de requisitos para um sistema de lavação de carros comum.

Projeto e arquitetura do sistema

a) Diagramas UML Essenciais nesta Fase

Durante a **fase de projeto e arquitetura**, os diagramas mais adequados são:

1. **Diagrama de Classes**
2. **Diagrama de Pacotes**

Esses dois diagramas são os principais responsáveis por **representar a estrutura estática** do sistema e a **organização modular** dos componentes.

b) Justificativa da Escolha de Cada Diagrama

1. Diagrama de Classes

O **Diagrama de Classes** é o núcleo do projeto de software orientado a objetos. Ele mostra **as classes, atributos, métodos e os relacionamentos** (associações, heranças, composições, dependências) entre elas.

Justificativa:

- **Identificação dos principais objetos do sistema:**

No contexto do sistema de lavação de carros, o diagrama permite identificar elementos como:

- Carro (atributos: placa, modelo, cor)
- Cliente (nome, telefone, histórico)
- Funcionário (nome, função, disponibilidade)
- Serviço (tipo de lavagem, preço, duração)
- Agendamento (data, hora, status)
- Pagamento (valor, forma de pagamento, data)

- **Facilita a manutenção e futuras alterações:**

Com a estrutura clara das classes e suas relações, torna-se simples adicionar novas funcionalidades (ex: novos tipos de serviço, promoções ou formas de pagamento) sem comprometer o restante do sistema.

- **Apoia a comunicação entre equipe técnica e stakeholders técnicos:**

Desenvolvedores e arquitetos conseguem alinhar o entendimento sobre **a estrutura do código, regras de associação e responsabilidades de cada classe**.

- **Base para geração de código:**

Ferramentas CASE podem gerar automaticamente esboços de classes a partir do diagrama, acelerando a implementação.

2. Diagrama de Pacotes

O **Diagrama de Pacotes** organiza o sistema em **módulos lógicos ou camadas**, agrupando classes relacionadas em conjuntos coesos.

Justificativa:

- **Melhora a modularidade e a manutenibilidade:**

Caso uma parte do sistema precise ser alterada (ex: camada de persistência ou interface gráfica), as demais permanecem isoladas e estáveis.

- **Favorece a arquitetura em camadas:**

Esse diagrama ajuda a visualizar e validar a separação entre **camada de apresentação, negócio e dados**, reforçando boas práticas de arquitetura de software.

- **Apoia a integração entre equipes:**

Cada equipe (front-end, back-end, banco de dados) entende quais pacotes são de sua responsabilidade, evitando sobreposição de tarefas.

c) Diagramas UML Provavelmente Deixados de Lado

Durante a fase de **projeto e arquitetura, não serão utilizados** os seguintes diagramas UML:

1. Diagrama de Casos de Uso
2. Diagrama de Atividade
3. Diagrama de Sequência
4. Diagrama de Colaboração
5. Diagrama de Estados
6. Diagrama de Componentes
7. Diagrama de Implantação
8. Diagrama de Objetos
9. Diagrama de Estrutura Composta
10. Diagrama de Perfil
11. Diagrama de Tempo
12. Diagrama de Interação

d) Justificativa para a Omissão de Cada Um

Abaixo, a explicação do **porquê não serão utilizados nesta fase específica**:

Diagrama	Justificativa da Omissão
Casos de Uso	Já foi amplamente utilizado na fase de levantamento e análise de requisitos. Nesta fase, o foco não é mais o comportamento do usuário, mas a estrutura interna do sistema.
Atividade	Descreve fluxos de processos e decisões, mais útil para modelar regras de negócio e fluxos operacionais , não a estrutura estática.
Sequência	Mostra interações entre objetos ao longo do tempo; relevante para análise de comportamento dinâmico , não para a arquitetura estática .
Colaboração	Similar ao diagrama de sequência; enfatiza a comunicação entre objetos, irrelevante quando se define apenas a organização estrutural.
Estados	Foca na variação de estados de um objeto, como “lavando / concluído / cancelado”. Esse detalhe será tratado em etapas posteriores de implementação.
Componentes	Aplica-se melhor quando o sistema já está sendo dividido em módulos executáveis ou bibliotecas físicas , não apenas lógicos.
Implantação	Relevante na fase de implantação e infraestrutura , quando se define servidores e nós físicos. Não é necessário no projeto lógico.
Objetos	Representa instâncias de classes, útil para testes de modelagem dinâmica , não para definir estrutura global.
Estrutura Composta	Usado em sistemas complexos com componentes internos interconectados; o sistema de lavação é relativamente simples.
Perfil	Usado para criar extensões específicas da UML (pouco útil para este tipo de projeto).
Tempo	Útil em sistemas com comportamento temporal crítico (ex: embarcados), o que não se aplica aqui.
Interação	Resumo visual de interações — redundante em relação aos diagramas dinâmicos omitidos.

Implantação e evolução.

a) Diagramas UML essenciais para criar

Nesta fase, os **diagramas mais relevantes** são:

1. **Diagrama de Implantação**
2. **Diagrama de Componentes**

Esses dois diagramas fornecem uma visão clara da **infraestrutura de execução** e da **organização modular do sistema**, fundamentais para a instalação, manutenção e evolução do sistema de lavação de carros.

b) Justificativa detalhada da escolha de cada diagrama

1. Diagrama de Implantação

Função: Representar a **infraestrutura física e lógica** necessária para a operação do sistema no ambiente real.

Justificativa:

O sistema de lavação de carros envolve **diversos pontos físicos e tecnológicos**, como:

- Servidores de aplicação e banco de dados centralizados (em nuvem ou local).
- Dispositivos móveis utilizados pelos funcionários para registrar serviços e pagamentos.
- Terminais administrativos no escritório do lava-jato para controle de agendamentos, relatórios e estoque.
- Comunicação entre dispositivos via rede local ou internet.

O **diagrama de implantação** mostra **como cada parte do sistema é distribuída fisicamente**, incluindo:

- Servidor de aplicação hospedando o backend.
- Banco de dados hospedado em um servidor dedicado ou serviço em nuvem.
- Aplicativo móvel Android/iOS instalado nos dispositivos dos atendentes.
- Interface web acessada por navegadores nos computadores administrativos.

Valor agregado:

- Facilita a **comunicação com equipes de TI e infraestrutura**, permitindo que todos compreendam os requisitos de hardware, rede e configuração.
- Garante que o sistema funcione corretamente em ambientes reais, evitando falhas de integração entre dispositivos e servidores.
- Suporta decisões sobre **escalabilidade** (ex: adicionar novos lava-jatos ou filiais).
- Permite planejar a **implantação segura** (backup, conexões seguras, autenticação).

2. Diagrama de Componentes

Função: Mostrar os **módulos lógicos e interfaces** que compõem o sistema, e como eles se relacionam.

Justificativa:

Durante a fase de evolução, é essencial entender como o sistema é estruturado em partes independentes, como:

- **Módulo de Agendamento** (gerencia reservas de horários).
- **Módulo de Pagamentos** (cartão, PIX, integração com gateways).
- **Módulo de Clientes** (cadastro, histórico de lavagens).
- **Módulo de Relatórios** (estatísticas de serviços e faturamento).
- **Módulo de Administração** (usuários, permissões, configurações).

O **diagrama de componentes** mostra **como esses módulos se conectam**, suas **interfaces públicas**, e **dependências** entre eles.

Valor agregado:

- Facilita **manutenções e correções de bugs**, permitindo modificar ou substituir um módulo sem afetar o restante do sistema.
- Auxilia na **evolução** do sistema, como a inclusão de novos serviços (ex: planos de assinatura, fidelidade, integração com apps externos).
- Melhora a **comunicação entre desenvolvedores**, pois cada equipe pode entender claramente a fronteira de responsabilidade de cada componente.
- Permite planejar **atualizações modulares** sem comprometer a estabilidade do sistema inteiro.

c) Diagramas UML não utilizados nesta fase

Os diagramas a seguir **não serão utilizados** na fase de implantação e evolução, pois seu propósito é mais adequado às fases de **análise, projeto ou modelagem comportamental** do sistema:

- Diagrama de Casos de Uso
- Diagrama de Atividade
- Diagrama de Classes
- Diagrama de Sequência
- Diagrama de Colaboração
- Diagrama de Estados
- Diagrama de Pacotes
- Diagrama de Objetos
- Diagrama de Estrutura Composta
- Diagrama de Perfil
- Diagrama de Tempo
- Diagrama de Visão Geral de Interação

d) Justificativa da omissão dos demais diagramas

Nesta fase, o foco está na **distribuição física do sistema, implantação real e manutenção contínua**. Os demais diagramas não contribuem diretamente para esses objetivos:

- **Casos de Uso e Atividades:**
Úteis para entender **requisitos funcionais e fluxos de negócio**, mas nesta fase o sistema já está implementado; o esforço de refazê-los não traz ganho operacional.
- **Classes, Objetos e Estrutura Composta:**
Servem para detalhar a **estrutura lógica e orientada a objetos**, essencial em fases anteriores. Na implantação, não há necessidade de visualizar classes — apenas componentes e módulos executáveis.
- **Sequência, Colaboração, Estados, Tempo e Interação:**
Diagramas **dinâmicos**, usados para representar o **comportamento e interações em tempo de**

execução. Durante a implantação, o foco é na **infraestrutura e módulos**, não em fluxos internos do código.

- **Pacotes e Perfis:**

Embora úteis em projetos grandes, nesta fase seu benefício é marginal, já que a modularização é melhor compreendida através do **diagrama de componentes** e a organização de deploy via **diagrama de implantação**.