

Criação de modelos de especificação de classes e componentes do sistema

Conceitos, técnicas, exemplos práticos e aplicação

1. Da análise ao projeto: por que especificar classes e componentes?

Depois que o modelo de classes de análise (modelo de domínio) foi construído e validado, o projeto entra na fase de design, também chamada de projeto de software. Nessa fase, os conceitos de negócio, ainda genéricos, precisam ser transformados em **especificações técnicas detalhadas**: cada classe passa a ter seus métodos definidos, tipos de dados escolhidos, visibilidade de atributos estabelecida e, em sistemas maiores, é preciso decidir como as classes se agrupam em componentes maiores e reutilizáveis.

Enquanto o modelo de análise responde "o que" o sistema precisa representar, o modelo de especificação de classes e componentes responde "como" isso será efetivamente construído, servindo como um guia detalhado para a equipe de desenvolvimento.

2. Especificação detalhada de classes de projeto

2.1 Definição de atributos com tipos e visibilidade

Cada atributo identificado na análise recebe um tipo de dado específico (String, Integer, Date, etc.) e um nível de visibilidade (público, privado ou protegido), definindo quem pode acessar diretamente aquela informação.

Exemplo prático: em um sistema de gestão hospitalar, a classe de análise Paciente evolui para uma classe de projeto com o atributo *-nome*: *String* (privado, acessível apenas por métodos da própria classe) e *+id*: *Integer* (público, usado por outras partes do sistema para referenciar o paciente).

2.2 Definição de métodos e assinaturas

Os comportamentos identificados de forma genérica na análise são detalhados como métodos, com nome, parâmetros de entrada, tipo de retorno e, quando necessário, tratamento de exceções.

Exemplo prático: o comportamento "verificar disponibilidade" da classe de análise Consulta torna-se, no modelo de projeto, o método *+verificarDisponibilidade(data: Date, medico: Medico): Boolean*, já especificando os parâmetros necessários e o tipo de retorno.

2.3 Aplicação de padrões e princípios de design

Nesta etapa, também são aplicados princípios como SOLID e padrões de projeto (Design Patterns) para melhorar a organização, flexibilidade e manutenibilidade das classes especificadas.

Exemplo prático: para permitir diferentes formas de notificação ao paciente (SMS, e-mail, push), a equipe especifica uma interface *NotificadorPaciente* e diferentes classes que a implementam, seguindo o padrão Strategy, em vez de criar uma única classe com múltiplos condicionais.

3. Exemplo simplificado de especificação de classe (notação UML)

Representação textual simplificada de uma classe já especificada para o projeto, no padrão UML (atributos e métodos com visibilidade):

```
Consulta
-----
- id: Integer
- dataHora: DateTime
- status: StatusConsulta
-----
+ verificarDisponibilidade(data: Date, medico: Medico): Boolean
+ confirmar(): void
+ cancelar(motivo: String): void
```

4. Modelagem de componentes do sistema

Em sistemas de médio e grande porte, especificar apenas classes individuais não é suficiente: é preciso organizar grupos de classes relacionadas em **componentes** — unidades maiores, com interfaces bem definidas, que podem ser desenvolvidas, testadas e até substituídas de forma mais independente.

4.1 Identificação de componentes por responsabilidade

Classes que trabalham juntas para entregar uma funcionalidade coesa são agrupadas em um mesmo componente, seguindo o princípio de alta coesão interna e baixo acoplamento entre componentes diferentes.

Exemplo prático: no sistema hospitalar, as classes Consulta, Agenda e Medico são agrupadas no componente "Agendamento", enquanto Paciente, Prontuário e Diagnóstico são agrupadas no componente "Prontuário Eletrônico", cada componente podendo evoluir de forma relativamente independente.

4.2 Definição de interfaces entre componentes

Cada componente expõe uma interface pública, definindo quais operações estão disponíveis para outros componentes utilizarem, escondendo os detalhes internos de implementação.

Exemplo prático: o componente "Prontuário Eletrônico" expõe a interface *IConsultaHistorico*, com o método *obterHistoricoPaciente(idPaciente)*, que o componente "Agendamento" utiliza para verificar consultas anteriores sem precisar conhecer como o prontuário é armazenado internamente.

4.3 Documentação de dependências entre componentes

É importante registrar quais componentes dependem de quais, para entender o impacto de futuras mudanças e planejar corretamente a ordem de desenvolvimento e implantação.

Exemplo prático: a equipe documenta que o componente "Agendamento" depende do componente "Notificações" para avisar pacientes sobre consultas marcadas, o que significa que qualquer alteração na interface de notificações deve ser cuidadosamente avaliada quanto ao impacto no agendamento.

5. Resumo comparativo: análise x projeto de classes

Aspecto	Modelo de análise (domínio)	Modelo de especificação (projeto)
Foco	Entender o negócio	Definir como implementar
Atributos	Nome do conceito, sem tipo técnico	Tipo de dado e visibilidade definidos
Comportamentos	Descrição geral da regra de negócio	Métodos com assinatura completa
Organização	Classes isoladas ou pouco agrupadas	Classes agrupadas em componentes
Padrões técnicos	Não se aplicam ainda	Design patterns e princípios SOLID

6. Boas práticas na especificação de classes e componentes

- Manter alta coesão dentro de cada classe e componente (cada um com uma responsabilidade clara) e baixo acoplamento entre eles, facilitando manutenção e testes independentes.
- Especificar interfaces antes de detalhar a implementação interna de um componente, permitindo que diferentes equipes trabalhem em paralelo a partir de um contrato bem definido.
- Evitar over-engineering: nem todo sistema pequeno precisa ser dividido em muitos componentes separados; a granularidade deve ser proporcional à complexidade e ao tamanho da equipe.
- Manter a especificação atualizada conforme o sistema evolui, especialmente as interfaces entre componentes, que costumam ser o ponto mais sensível a mudanças.

7. Aplicação prática no dia a dia dos projetos

A criação de modelos de especificação de classes e componentes é aplicada por diferentes profissionais ao longo do ciclo de vida do projeto:

- **Arquitetos de software** definem a divisão em componentes e as interfaces entre eles, estabelecendo as fronteiras técnicas do sistema antes do início da codificação.
- **Desenvolvedores** utilizam as especificações de classes (atributos, métodos e assinaturas) como referência direta para implementar o código, reduzindo ambiguidades durante a programação.

- **Equipes que trabalham em paralelo** em partes diferentes de um sistema grande dependem das interfaces entre componentes já especificadas para integrar seu trabalho corretamente, mesmo sem conhecer os detalhes internos de outras equipes.
- **Times de manutenção** utilizam a documentação de dependências entre componentes para avaliar o impacto de mudanças futuras antes de alterá-las em produção.

8. Considerações finais

A criação de modelos de especificação de classes e componentes representa a transição da compreensão do negócio para a construção efetiva do software. Ao detalhar atributos, métodos, padrões de projeto e a organização em componentes com interfaces bem definidas, a equipe reduz ambiguidades na implementação, facilita o trabalho em paralelo entre diferentes desenvolvedores e cria uma base técnica mais sólida, flexível e preparada para evoluir ao longo do tempo.