**Módulo 1: Ciclo de Vida de Desenvolvimento de Software - Diagramas de Interação**

◦

Os **diagramas de interação** na UML são usados para mostrar o comportamento interativo de um sistema1. Eles explicitam as comunicações, descrevendo o fluxo de mensagens e fornecendo contexto para as linhas da vida de objetos1.

◦

Podem ser usados para modelar um sistema como uma sequência de eventos organizada por tempo, fazer engenharia reversa/avançada, organizar eventos interativos, mostrar comportamento de mensagens/linhas da vida e identificar conexões entre elementos2.

◦

**Principal objetivo**: refinar o diagrama de classes na etapa de projeto, detalhando métodos3.

◦

**Tipos Principais**: Diagrama de Sequência e Diagrama de Comunicação3....

◦

**Diagrama de Sequência**: Mostra a interação focando na ordem cronológica das mensagens4.... Não é necessário construir para todos os casos de uso; foca nos mais complexos e relacionados ao negócio3.... Pode ser trabalhoso para o sistema inteiro4. Pode ter um diagrama principal e complementares para fluxos alternativos/erros4.

◦

**Diagrama de Comunicação**: Mostra como os objetos interagem para executar um caso de uso, determinando interfaces e responsabilidades e definindo papéis4.... Apresenta o fluxo de mensagens entre objetos, similar ao diagrama de sequência, mas com uma forma de exibição diferente5.

•

**Módulo 2: Diagrama de Classes de Projeto**

◦

É um **refinamento do diagrama de classes** construído na fase de análise6. Modifica propriedades, notações adicionais, atributos, operações e associações6.

◦

**Especificação de Associações**: Mecanismos que permitem aos objetos se comunicarem, descrevendo a conexão entre classes7.... Podem ter regras, ser unidirecionais ou bidirecionais, e possuem multiplicidade em cada ponta8.

◦

**Dependência**: Um tipo relevante de associação na etapa de projeto8. Representa relacionamentos onde **uma classe depende da outra**; modificações na classe independente afetam a dependente9. Indica que uma classe necessita dos serviços da outra9. É melhor definida na fase de projeto por influenciar a implementação9.

◦

**Notação de Dependência**: Seta tracejada da classe dependente para a classe da qual depende10.

◦

**Transformação de Associações em Dependências**: Na passagem da análise para o projeto, associações são estudadas para identificar se podem ser transformadas em dependências10.... A razão é **definir melhor o encapsulamento**11. Quanto menos dependências estruturais, maior a qualidade do projeto (encapsulamento, acoplamento)11. Associações entre classes de entidade geralmente permanecem como associações11.

◦

**Classes de Interface**: Coleção de operações com um nome que especifica um tipo de serviço **sem detalhar a implementação**11.... Não têm métodos concretos, apenas declaram para outra classe implementar12. Permitem a colaboração de objetos externos12.

◦

**Navegabilidade de Associações**: Podem ser bidirecionais (conhecimento mútuo) ou unidirecionais12. No diagrama, uma seta indica a direção: a classe para onde a seta aponta é a que *não* tem visibilidade dos objetos da outra classe13. No modelo de análise, associações são usualmente bidirecionais; no projeto, a navegabilidade deve ser refinada13.

•

**Módulo 3: Diagramas Dinâmicos (Diagrama de Transição de Estados e Diagrama de Atividades)**

◦

**Diagrama de Transição de Estados (DTE)**: Diagrama da UML usado para analisar a **mudança de estado** de um objeto14. Um objeto muda de estado quando ocorre um evento14.

◦

**Estado**: Uma situação na vida de um objeto, determinada pelos valores de seus atributos ou ligações15.

◦

**Transição**: Linha conectando estados com uma seta do estado origem para o destino15. Disparada por um evento16. O estado subsequente pode ser o mesmo que o original16. Regras de negócio podem ajudar a identificar estados e transições16.

◦

**Construção de DTEs**: Para sistemas complexos, a quantidade de estados é grande17. Para evitar a "explosão de estados", os diagramas são desenhados **por classe**17. Isso torna o diagrama por classe compreensível, mas dificulta a visualização do estado geral do sistema17.... Essa desvantagem é parcialmente compensada por diagramas de interação18. Nem todas as classes precisam de DTE; apenas aquelas com **comportamento dinâmico relevante**18. Classes cujo histórico precisa ser rastreado são típicas18.

◦

**Diagrama de Atividades**: Um tipo especial de diagrama de estados que representa **estados de uma atividade**19. É **orientado a fluxos de controle**, ao contrário dos diagramas de estados (orientados a eventos)19.

◦

Possui notação para representar **ações concorrentes (paralelas)** e sua sincronização19.... Barras de sincronização são usadas para isso20.

◦

**Raias ou Partições**: Usadas para dividir a execução de processos, frequentemente em processos de negócio por pessoas ou departamentos20....

◦

Pode ser usado para **modelar a lógica de uma operação complexa** ou regra de negócio, assemelhando-se a um fluxograma21....

•

**Módulo 4: Diagramas de Estrutura Física (Diagrama de Componentes e Diagrama de Implantação)**

◦

**Diagrama de Componentes**: Apresenta o relacionamento entre os diferentes **componentes** de um sistema de software22.

◦

**Componente**: Um módulo (artefato do sistema) que representa sistemas/subsistemas com capacidade de interagir22. Funciona como uma **caixa preta** que oferece/requer serviços via interfaces, sem conhecer seu conteúdo interno23. Modelado ao longo do ciclo de vida23. Pode ter uma estrutura interna ("caixa branca") com partes e conectores24.

◦

**Interfaces**: Coleções de operações que **especificam serviços** de um componente25. É por onde os componentes se comunicam (oferecem ou recebem serviços)25. Descrevem comportamento visível externamente, sem estrutura (sem atributos) ou implementação25.

◦

**Camadas**: Arquitetura que permite isolar camadas (apresentação, regras de negócio, armazenamento de dados), melhorando desempenho e facilitando manutenção/atualização25....

◦

**Diagrama de Implantação**: Especifica artefatos de infraestrutura para definir a **arquitetura física de execução** do sistema26. Foca na organização sobre a qual o software será implantado e executado (hardware, software básico, redes)26.... Representa a estrutura física do sistema27.

◦

**Nós (Nodes)**: Representam dispositivos de hardware e ambientes de execução de software (servidores, computadores, SOs, bancos de dados, navegadores)27.... Podem conter outros nós e são conectados por caminhos de comunicação28.

◦

**Estereótipos principais**: <<device>> (dispositivos com processamento), <<artifact>> (artefatos físicos como executáveis, BDs), <<execution environment>> (SO, gerenciadores de BD)28....

◦

**Utilidade**: Útil para comunicação entre equipes de desenvolvimento e infraestrutura, configuração/instalação29. Mais relevante para **sistemas distribuídos e complexos**29.