Excelente! Vamos mergulhar no "Eixo 2: Desenvolvimento de Software e Engenharia de Sistemas" do CNU Bloco 3 com foco total na preparação para a FGV. Este eixo é a espinha dorsal da Tecnologia da Informação, tratando de como construímos sistemas eficientes, robustos e que realmente resolvem problemas.

A FGV gosta de cobrar \*\*conceitos\*\*, \*\*aplicações práticas\*\* (embora em questões teóricas, o "cenário" é a prática), \*\*comparações entre metodologias\*\* e a \*\*razão por trás\*\* de cada fase ou artefato. Foco na compreensão, não apenas na memorização.

Prepare seu material de revisão. Aqui estão os pontos essenciais:

---

\*\*Eixo 2: Desenvolvimento de Software e Engenharia de Sistemas\*\*

\*\*O que é?\*\*

\* Área que trata do \*\*processo\*\* de criação de sistemas de software e sistemas complexos (incluindo hardware, software, processos, pessoas).

\* Vai da \*\*ideia inicial\*\* (requisitos) até a \*\*entrega\*\* e \*\*manutenção\*\*.

\* Envolve \*\*metodologias, ferramentas e técnicas\*\* para garantir qualidade e eficiência.

---

\*\*1. Ciclo de Vida de Desenvolvimento de Software (SDLC - Software Development Life Cycle)\*\*

\* \*\*Conceito:\*\* Estrutura que define as fases do desenvolvimento de um sistema, do início ao fim.

\* \*\*Propósito:\*\* Organizar, planejar, controlar e gerenciar o projeto.

\* \*\*Fases Típicas (varia conforme a metodologia):\*\*

\* \*\*Requisitos:\*\* Entender o que o cliente/usuário precisa.

\* \*\*Projeto/Design:\*\* Definir como o sistema será construído (arquitetura, estrutura, interfaces).

\* \*\*Implementação/Codificação:\*\* Escrever o código.

\* \*\*Testes:\*\* Verificar se o sistema funciona como esperado e atende aos requisitos.

\* \*\*Implantação/Deploy:\*\* Colocar o sistema em produção.

\* \*\*Manutenção:\*\* Corrigir erros, adaptar, evoluir o sistema após a implantação.

---

\*\*2. Metodologias de Desenvolvimento de Software\*\*

\* \*\*Conceito:\*\* Diferentes abordagens para seguir o SDLC.

\* \*\*Importância:\*\* Guia o time, define papéis, ritos, artefatos e o fluxo de trabalho.

\* \*\*Principais Tipos:\*\*

\* \*\*Metodologias Tradicionais/Cascata (Waterfall):\*\*

\* \*\*Característica:\*\* Linear, sequencial, uma fase só começa após a conclusão da anterior.

\* \*\*Vantagens:\*\* Simples de entender e gerenciar para projetos com requisitos MUITO estáveis.

\* \*\*Desvantagens (Foco FGV):\*\* Pouca flexibilidade a mudanças, detecção tardia de erros, cliente só vê o produto no final. \*\*Não ideal para a maioria dos projetos modernos.\*\*

\* \*\*Metodologias Iterativas e Incrementais:\*\*

\* \*\*Característica:\*\* O ciclo de vida é percorrido em várias iterações (ciclos), entregando incrementos funcionais a cada ciclo. Ex: Modelo Espiral, Modelo V.

\* \*\*Vantagens:\*\* Feedback mais cedo, riscos mitigados ao longo do tempo.

\* \*\*Metodologias Ágeis (Agile) - FOCO FORTÍSSIMO DA FGV:\*\*

\* \*\*Filosofia:\*\* Baseada no Manifesto Ágil (4 valores e 12 princípios).

\* \*\*Valores do Manifesto Ágil (FGV PODE COBRAR):\*\*

\* Indivíduos e interações \*\*mais que\*\* processos e ferramentas.

\* Software funcionando \*\*mais que\*\* documentação abrangente.

\* Colaboração com o cliente \*\*mais que\*\* negociação de contratos.

\* Responder à mudança \*\*mais que\*\* seguir um plano.

\* \*\*Características Principais:\*\* Flexibilidade a mudanças, entregas frequentes de software funcionando, colaboração intensa (cliente e time), foco nas pessoas, auto-organização dos times.

\* \*\*Principais Frameworks Ágeis:\*\*

\* \*\*Scrum (MUITO Cobrado pela FGV):\*\*

\* \*\*É um Framework, não uma metodologia rígida.\*\*

\* \*\*Papéis:\*\* Product Owner (dono do produto, define o QUÊ), Scrum Master (servo-líder, remove impedimentos, garante o COMO - o processo), Time de Desenvolvimento (quem FAZ).

\* \*\*Eventos/Cerimônias:\*\* Sprint Planning (planejar a Sprint), Daily Scrum (reunião diária rápida, sincronização), Sprint Review (revisar o trabalho feito na Sprint com stakeholders), Sprint Retrospective (melhorar o processo na próxima Sprint).

\* \*\*Artefatos:\*\* Product Backlog (lista priorizada de funcionalidades), Sprint Backlog (itens do Product Backlog selecionados para a Sprint), Incremento (produto funcional gerado ao final da Sprint).

\* \*\*Kanban:\*\*

\* \*\*Foco:\*\* Visualizar o fluxo de trabalho, limitar o trabalho em progresso (WIP - Work in Progress), gerenciar o fluxo.

\* \*\*Não possui papéis/eventos fixos como Scrum.\*\* Adaptável.

\* \*\*Muito usado para manutenção, suporte ou projetos com fluxo contínuo.\*\*

---

\*\*3. Engenharia de Requisitos\*\*

\* \*\*Conceito:\*\* Processo de descobrir, analisar, especificar, validar e gerenciar os requisitos de um sistema.

\* \*\*Importância:\*\* É a base de todo o projeto. Requisitos mal definidos levam a projetos fracassados.

\* \*\*Tipos de Requisitos (FGV COBRA DIFERENÇA):\*\*

\* \*\*Requisitos Funcionais:\*\* O QUE o sistema deve fazer (funções, comportamentos específicos). Ex: "O sistema deve permitir que o usuário faça login".

\* \*\*Requisitos Não Funcionais:\*\* COMO o sistema deve se comportar (qualidades, restrições). Ex: Performance (tempo de resposta), Segurança (nível de criptografia), Usabilidade (facilidade de uso), Confiabilidade, Escalabilidade, Manutenibilidade, Portabilidade.

\* \*\*Atividades:\*\* Elicitação (coleta), Análise (entendimento), Especificação (documentação), Validação (confirmar se atendem às necessidades), Gerenciamento (controle de mudanças).

---

\*\*4. Projeto e Arquitetura de Software\*\*

\* \*\*Conceito:\*\* Definir a estrutura interna do sistema, seus componentes, suas relações e restrições.

\* \*\*Arquitetura:\*\* Visão de alto nível, estrutura global do sistema.

\* \*\*Projeto:\*\* Detalhes de como os componentes individuais serão implementados.

\* \*\*Princípios de Projeto Comuns:\*\*

\* \*\*SOLID:\*\* Conjunto de 5 princípios (SRP, OCP, LSP, ISP, DIP) para design de software orientado a objetos mais flexível e manutenível. \*Conhecer o conceito geral ajuda.\*

\* \*\*DRY (Don't Repeat Yourself):\*\* Evitar duplicação de código/lógica.

\* \*\*KISS (Keep It Simple, Stupid):\*\* Manter o design o mais simples possível.

\* \*\*Padrões de Projeto (Design Patterns):\*\* Soluções reutilizáveis para problemas comuns de design de software (Ex: MVC - Model-View-Controller, muito comum em aplicações web).

\* \*\*Estilos Arquiteturais (FGV PODE COBRAR COMPARAÇÃO):\*\*

\* \*\*Monolítico:\*\* Todo o sistema é construído como uma única unidade de implantação.

\* Vantagens: Simples para projetos pequenos, fácil deploy inicial.

\* Desvantagens: Dificuldade de escalar partes específicas, manutenção complexa em sistemas grandes, dificuldade em adotar novas tecnologias em partes do sistema.

\* \*\*Microsserviços:\*\* O sistema é dividido em pequenos serviços independentes, que se comunicam (geralmente via APIs).

\* Vantagens: Escalabilidade independente, fácil adoção de novas tecnologias (em cada serviço), maior resiliência (falha em um serviço não derruba tudo).

\* Desvantagens: Complexidade de gerenciar múltiplos serviços, comunicação distribuída, necessidade de orquestração.

\* \*\*SOA (Service-Oriented Architecture):\*\* Abordagem onde os serviços são a unidade fundamental, que podem ser reutilizados e combinados para formar aplicações maiores. Microsserviços podem ser vistos como uma evolução de SOA.

---

\*\*5. Testes de Software\*\*

\* \*\*Conceito:\*\* Processo para avaliar se o software atende aos requisitos e identificar defeitos. \*\*Essencial para a qualidade.\*\*

\* \*\*Níveis de Teste (FGV COBRA OS NÍVEIS):\*\*

\* \*\*Teste de Unidade:\*\* Testa a menor parte isolada do código (função, método, classe). Geralmente feito pelo próprio desenvolvedor.

\* \*\*Teste de Integração:\*\* Testa a interação entre módulos ou componentes integrados.

\* \*\*Teste de Sistema:\*\* Testa o sistema completo e integrado para verificar se atende aos requisitos funcionais e não funcionais.

\* \*\*Teste de Aceitação:\*\* Testa o sistema sob a perspectiva do usuário final ou cliente para validar se atende às suas necessidades e expectativas. Pode ser Alfa (interno) ou Beta (externo, com usuários reais).

\* \*\*Tipos de Teste (FGV COBRA A CLASSIFICAÇÃO):\*\*

\* \*\*Funcionais:\*\* Verificam o QUE o sistema faz (com base nos requisitos funcionais).

\* \*\*Caixa Preta (Black-box):\*\* Não se conhece a estrutura interna do código. Testa apenas a entrada e a saída (comportamento).

\* \*\*Não Funcionais:\*\* Verificam o COMO o sistema se comporta (performance, segurança, usabilidade, etc. - com base nos requisitos não funcionais).

\* \*\*Caixa Branca (White-box):\*\* Conhece-se a estrutura interna do código. Testa caminhos de execução, loops, condições.

\* \*\*Regressão:\*\* Garante que as novas mudanças (correções ou novas funcionalidades) não quebraram funcionalidades existentes.

---

\*\*6. Qualidade de Software\*\*

\* \*\*Conceito:\*\* Grau em que um software atende a requisitos explícitos e implícitos, satisfazendo as necessidades dos usuários.

\* \*\*Garantia da Qualidade (QA - Quality Assurance):\*\* Processos preventivos e sistêmicos para garantir que a qualidade seja construída no software \*desde o início\*. Foco no PROCESSO.

\* \*\*Controle de Qualidade (QC - Quality Control):\*\* Atividades para identificar defeitos no produto \*após\* ele ter sido construído (ex: Testes, Inspeções). Foco no PRODUTO.

\* \*\*Modelos de Qualidade (FGV PODE COBRAR):\*\*

\* \*\*ISO/IEC 25010 (Evolução da ISO 9126):\*\* Define características da qualidade de software:

\* \*\*Qualidade do Produto:\*\* Funcionalidade, Confiabilidade, Usabilidade, Eficiência de Desempenho, Segurança, Manutenibilidade, Compatibilidade, Portabilidade. \*Decorar ou entender o significado de cada uma é útil.\*

---

\*\*7. Gerenciamento de Configuração de Software (GCS) / SCM\*\*

\* \*\*Conceito:\*\* Disciplina para controlar as mudanças nos artefatos do projeto (código-fonte, documentação, scripts, etc.).

\* \*\*Componentes:\*\* Identificação de itens de configuração, Controle de Mudanças, Auditoria de Configuração, Relatórios de Status.

\* \*\*Ferramentas de Versionamento (FOCO FGV: GIT):\*\* Essenciais para o GCS, permitem rastrear mudanças, colaborar em times e reverter para versões anteriores.

\* \*\*Git:\*\* Sistema de controle de versão distribuído. Conceitos como commit, branch, merge, clone, pull, push são fundamentais no dia a dia e podem aparecer.

---

\*\*8. Integração Contínua (CI) e Entrega/Implantação Contínua (CD) / DevOps\*\*

\* \*\*Contexto:\*\* Práticas modernas, fortemente ligadas às metodologias ágeis, que visam aumentar a velocidade e a confiabilidade das entregas.

\* \*\*Integração Contínua (CI):\*\* Prática de integrar o código dos desenvolvedores na linha principal (mainline) \*muitas vezes ao dia\*, seguida de \*\*construção (build) e testes automatizados\*\* para detectar problemas de integração o mais cedo possível.

\* \*\*Entrega Contínua (CD - Continuous Delivery):\*\* Extensão da CI onde o código que passou pelos testes automatizados está \*\*sempre em um estado pronto para ser implantado\*\* em produção. A implantação em si pode ser manual ou automatizada.

\* \*\*Implantação Contínua (CD - Continuous Deployment):\*\* Extensão da Entrega Contínua onde \*\*toda mudança que passa por todos os estágios do pipeline (build, testes) é automaticamente implantada em produção\*\*, sem intervenção manual.

\* \*\*DevOps:\*\* Cultura e conjunto de práticas que visa a \*\*colaboração e comunicação\*\* entre as equipes de Desenvolvimento (Dev) e Operações (Ops). Foco na \*\*automação do pipeline de entrega\*\*, monitoramento, infraestrutura como código, para agilizar o ciclo de vida e aumentar a confiabilidade. \*\*CI/CD são práticas centrais do DevOps.\*\*

---

\*\*9. Modelagem de Sistemas (UML - Unified Modeling Language)\*\*

\* \*\*Conceito:\*\* Linguagem gráfica padronizada para visualizar, especificar, construir e documentar artefatos de sistemas de software.

\* \*\*Importância:\*\* Ajuda a entender a estrutura e o comportamento do sistema antes (e durante) a implementação.

\* \*\*Diagramas Comuns (FGV COBRA OS PROPÓSITOS):\*\*

\* \*\*Diagramas de Casos de Uso:\*\* Descrevem as funcionalidades do sistema sob a perspectiva dos usuários (atores). \*O quê o sistema faz para cada tipo de usuário.\*

\* \*\*Diagramas de Classes:\*\* Descrevem a estrutura estática do sistema (classes, atributos, métodos e relacionamentos - herança, associação, agregação, composição). \*A estrutura de dados e comportamento.\*

\* \*\*Diagramas de Sequência:\*\* Descrevem a interação entre objetos ao longo do tempo, mostrando a ordem das mensagens trocadas. \*Como os objetos colaboram para realizar uma tarefa.\*

\* \*\*Diagramas de Atividade:\*\* Descrevem o fluxo de trabalho ou processo de negócio, mostrando a sequência de atividades e decisões. \*O fluxo de controle.\*

---

\*\*Dica FGV:\*\* A banca gosta de criar pequenas situações ("Em um projeto ágil...", "Um time de desenvolvimento detectou...", "Qual o melhor diagrama para representar...") e pedir para você aplicar o conceito correto. Entender o \*\*propósito\*\* de cada item (por que usar Scrum? Por que fazer teste de unidade? Qual a diferença entre QA e QC?) é mais importante do que apenas memorizar definições.

Revise estes pontos com atenção. Eles cobrem a maior parte do que a FGV costuma abordar neste eixo. Boa sorte nos estudos!