Engenharia de Software - Guia de Estudo

📚 Visão Geral dos Tópicos Principais

1. Fundamentos Agile & Scrum

Valores do Manifesto Agile

- **Indivíduos e interações** > processos e ferramentas
- **Software funcional** > documentação abrangente
- **Colaboração com o cliente** > negociação de contratos
- **Responder à mudança** > seguir um plano

Framework Scrum (Estrutura 3-5-3)

3 Papéis:

- **Product Owner**: Maximiza valor do produto, gere backlog, define Product Goal
- **Scrum Master**: Garante que Scrum é compreendido, remove impedimentos, facilita eventos
- **Developers**: Equipa auto-organizada que entrega Incrementos

5 Eventos:

- **Sprint** (máx 1 mês): Contentor para todos os outros eventos
- **Sprint Planning**: Define Sprint Goal, seleciona itens do backlog, planeia trabalho
- **Daily Scrum** (15 min): Sincroniza trabalho, planeia próximas 24 horas
- **Sprint Review**: Inspeciona Incremento, adapta Product Backlog
- **Sprint Retrospective**: Planeia melhorias para próxima Sprint

3 Artefactos:

- **Product Backlog** → Compromisso: Product Goal
- **Sprint Backlog** → Compromisso: Sprint Goal
- **Increment** → Compromisso: Definition of Done

2. Requisitos & User Stories

Critérios INVEST

- **I**ndependent (Independente)
- **N**egotiable (Negociável)
- **V**aluable (Valioso)
- **E**stimable (Estimável)
- **S**mall (Pequeno)
- **T**estable (Testável)

Formato de User Story

Como <tipo de utilizador>
Quero <alguma funcionalidade>
Para que <razão/benefício>

Critérios de Aceitação (Given-When-Then)

Dado <contexto inicial>
Quando <ação ocorre>
Então <resultado esperado>

Estimação

- **Story Points**: Medida relativa (Fibonacci: 1, 2, 3, 5, 8, 13...)
- **Planning Poker**: Estimativa baseada em consenso da equipa
- **Vantagens sobre horas**: Independente do nível de competência, acompanha velocidade, sem necessidade de re-estimação

3. Componentes de Software & Arquitetura

Horizontal vs Vertical Slicing

- **Horizontal**: Dividido por camadas técnicas (frontend/backend/BD) → Sem valor para utilizador até estar completo
- **Vertical**: Fatias funcionais cross-layer → Cada uma entrega valor deployável

Princípios SOLID

- **S**RP: Single Responsibility Principle (Princípio da Responsabilidade Única)
- **O**CP: Open-Closed Principle (Princípio Aberto-Fechado)
- **L**SP: Liskov Substitution Principle (Princípio da Substituição de Liskov)
- **I**SP: Interface Segregation Principle (Princípio da Segregação de Interface)
- **D**IP: Dependency Inversion Principle (Princípio da Inversão de Dependência)

Princípios de Componentes

- **REP**: Reuse/Release Equivalence (Equivalência Reutilização/Release)
- **CCP**: Common Closure Principle (mudam juntos)
- **CRP**: Common Reuse Principle (não forçar dependências não usadas)

4. CI/CD & Deployment

Continuous Integration (CI)

- Integração frequente no branch principal
- Build, teste e validação automatizados
- Feedback rápido sobre qualidade do código

Continuous Delivery vs Deployment

- **Delivery**: Automatizado até produção, aprovação manual necessária
- **Deployment**: Totalmente automatizado para produção (sem gate manual)

Conceitos-Chave GitHub Actions

- **Workflows**: Ficheiros YAML em .github/workflows/
- **Triggers**: push, pull_request, schedule (cron), workflow_dispatch, repository_dispatch
- **Jobs**: Executam em runners (ubuntu-latest, etc.)
- **Steps**: Ações sequenciais ou comandos shell
- **Matrix Strategy**: Testa em múltiplas configurações

5. Estratégias de Progressive Delivery

Estratégia	Descrição	Rollback	Caso de Uso
Canary Release	Deploy para subconjunto pequeno primeiro	Parcial	Validar com utilizadores reais gradualmente
Blue/Green	Dois ambientes, switch instantâneo	Instantâneo	Releases sem downtime
A/B Testing	Comparar variantes	N/A	Medir comportamento/preferências utilizador
Dark Launch	Deploy mas escondido dos utilizadores	N/A	Testar performance antes de exposição

Feature Toggles (4 Tipos)

1. **Release**: Controlar visibilidade de funcionalidade (temporário)
2. **Experiment**: Testes A/B
3. **Ops**: Afinação de performance
4. **Permission**: Acesso específico por utilizador

6. Estratégias de Testes

Test Pyramid (Preferir) vs Test Ice Cream (Evitar)

Tipos de Testes

- **Static**: Linters, type checkers (compile-time)
- **Unit**: Componentes isolados, rápido, mocks
- **Integration**: Interações entre componentes, bases de dados
- **E2E**: Sistema completo, perspetiva do utilizador (Cypress, Playwright)
- **System**: Black-box, deployment completo

Testing em Produção

- Usar **feature toggles** para exploração segura

- **Dados de teste sintéticos** (sinalizados e escondidos)
- **Source maps** para debugging
- Ambientes preprod lean

7. DevSecOps

Princípio Central

"Segurança é responsabilidade de todos"

- De "Segurança como Gate" → "Segurança como Guardrail"

Security Shift

- **Shift Left**: Início do ciclo de vida (threat modeling, SAST, pre-commit hooks)
- **Shift Right**: Segurança em produção (RASP, monitoring, incident response)

Práticas-Chave por Fase

Planning:

- Threat Modeling (STRIDE)
- Abuse Cases ("Evil User Stories")
- Requisitos de segurança como código

Coding:

- IDE Security Plugins (SonarLint)
- SCA (Software Composition Analysis) - verificar dependências
- Pre-commit hooks (scanning de secrets)

Building:

- SAST (Análise Estática) - scanning código fonte
- Container/base image scanning
- Breaking the build (alta severidade)
- Artifact signing

Testing:

- DAST (Análise Dinâmica) - app em execução
- IAST (Interactive) - combina SAST + DAST

- Fuzz testing (inputs aleatórios)

Release:

- IaC Scanning (Checkov, tfsec)
- Secrets management (padrão Vault)
- Policy as Code (Gatekeeper)

Deploy:

- RASP (Proteção Runtime)
- Infraestrutura imutável

Monitor:

- SIEM + UEBA (analytics comportamental)
- Scanning contínuo de compliance

8. Infrastructure as Code (IaC)

CAPEX vs OPEX

- **CAPEX**: Infraestrutura física inicial, ativos fixos
- **OPEX**: Pay-as-you-go, sem depreciação, scaling flexível

Pets vs Cattle

- **Pets**: Nomeados, mantidos manualmente, insubstituíveis
- **Cattle**: Numerados, descartáveis, auto-provisionados, efémeros

Ferramentas IaC

- **Provisioning**: Terraform, OpenTofu, Pulumi
- **Configuration Management**: Ansible, Chef, Puppet
- **Containers**: Docker, Packer
- **Orchestration**: Kubernetes, Docker Swarm

Workflow Terraform

1. **Write**: Definir infraestrutura como código
2. **Plan**: Pré-visualizar mudanças

3. **Apply**: Provisionar recursos

9. Monitoring & Observability

SLI, SLO, SLA

- **SLI** (Indicator): Métrica quantitativa (ex: % pedidos bem-sucedidos)
- **SLO** (Objective): Target para SLI (ex: 99.9% disponibilidade)
- **SLA** (Agreement): Contrato com consequências

Monitoring vs Observability

- **Monitoring**: Alerta **quando** algo acontece
- **Observability**: Explica **porquê** acontece

Três Pilares de Observability

1. **Logs**: Registos de eventos
2. **Metrics**: Dados numéricos agregados (time-series)
3. **Traces**: Fluxo de pedidos entre serviços (spans)

Ferramentas Observability

- **Metrics**: Prometheus, VictoriaMetrics
- **Visualization**: Grafana
- **Logging**: Elastic Stack (ELK), Fluentd
- **Tracing**: Jaeger, OpenTelemetry
- **All-in-one**: Elastic Observability, Datadog

10. Gestão de Código Legacy

Definição

"Código legacy é código que tens medo de mudar" (porque não tem testes)

Nove Práticas para Estender Vida do Software

1. **Dizer O Quê, Porquê, Para Quem antes do Como**
2. **Build em Small Batches**

3. **Integrar Continuamente**
4. **Colaborar** (pair/mob programming, propriedade coletiva)
5. **Criar Código CLEAN** (Boy Scout Rule)
6. **Escrever o Teste Primeiro** (capturar comportamento)
7. **Especificar Comportamentos com Testes** (estilo BDD)
8. **Implementar Design Por Último** (TDD: Red → Green → Refactor)
9. **Refatorar Código Legacy** (com rede de segurança)

Padrões de Refactoring

System Strangling (Macro):

1. Inserir proxy/façade
2. Redirecionar 100% para sistema antigo
3. Implementar um endpoint no novo
4. Atualizar routing do proxy
5. Repetir até sistema antigo receber 0% tráfego

Branch by Abstraction (Micro):

1. Criar camada de abstração
2. Refatorar clientes para usar abstração
3. Implementar novo supplier por trás da abstração
4. Mudar clientes para novo supplier
5. Remover supplier antigo

11. MLOps

MLOps vs DevOps

Aspeto	DevOps	MLOps
Versioning	Código	Código + Dados + Modelo
CI/CD	Build/Test/Deploy	+ Continuous Training (CT)
Monitoring	Infraestrutura	+ Qualidade/Drift do Modelo

Ciclo de Vida MLOps

1. **Data Management**: Pipelines, versioning (DVC), lineage
2. **Model Development**: EDA, training, experiment tracking (MLflow)
3. **Model Deployment**: Packaging, infraestrutura, integração

4. **Monitoring**: KPIs, deteção de drift, triggers de retreino
5. **Retiring**: Avaliar obsolescência, substituir por nova versão

Níveis de Maturidade (5 Níveis)

- **Nível 1**: Manual, siloed, sem versioning
- **Nível 2**: DevOps para infra, pipeline de dados automatizado
- **Nível 3**: Training automatizado, metadata store, model registry
- **Nível 4**: Pipelines CI/CD, testes A/B
- **Nível 5**: Retreino automatizado em drift, sistemas adaptativos

12. LLMOps

LLMOps vs MLOps

Aspeto	MLOps	LLMOps
Main Asset	Modelo treinado	Modelo + Prompts + Contexto
Training	Epochs, dados versionados	Fine-tuning ou In-context learning
Monitoring	Precisão de previsão	Qualidade de geração (toxicidade, alucinações)

Fases-Chave

1. **Prompt Engineering**: Versionar prompts como código
2. **Context Management (RAG)**: Base de dados vetorial para info relevante
3. **Deployment**: Resiliência API, otimização latência
4. **A/B Testing**: Testar variantes de prompt/modelo
5. **Monitoring**: Métricas de qualidade, model-as-a-judge
6. **Security**: Filtragem input/output, guardrails

LLMs no SDLC

- **Casos de Uso**: Geração de código, debugging, documentação, testes
- **Ferramentas**: GitHub Copilot, Cursor, Windsurf, ChatGPT
- **Estruturação de Prompt**: Simple, Specific, Short (3S)
- **Context Window**: Tokens máximos (ex: GPT-4: 128k, Claude: 200k)
- **Agents**: Sistemas autónomos usando ferramentas (protocolo MCP)

Melhores Práticas

- Testar/verificar todos os outputs
- Documentar prompts & decisões
- Evitar expor dados sensíveis
- Usar para assistência, não lógica crítica
- Melhoria contínua de prompts

13. Fórmulas & Métricas-Chave

Disponibilidade

Disponibilidade (%) = (Uptime / (Uptime + Downtime)) × 100

- 99.9% = 8.76 horas downtime/ano
- 99.99% = 52 minutos downtime/ano

Load Balancing

Carga por servidor = Carga Total / Número de Servidores Ativos

Escalabilidade

Escalabilidade = (Carga Máxima / Carga Inicial)
OU (Performance em Escala / Performance Base)

MTTR & MTTD

- **MTTD** (Mean Time To Detect): Quanto tempo ameaças vivem não detetadas
- **MTTR** (Mean Time To Recover): Quão rápido patches são aplicados

🎯 Dicas de Preparação para Exame

Deve Saber de Cor

1. **Scrum 3-5-3**: Papéis, Eventos, Artefactos + Compromissos
2. **Critérios INVEST** para user stories
3. **Horizontal vs Vertical slicing**
4. **CI vs CD vs Continuous Deployment**

5. **Hierarquia Test Pyramid**
6. **SLI vs SLO vs SLA**
7. **Monitoring vs Observability**
8. **Canary vs Blue/Green deployment**
9. **SAST vs DAST**
10. **Diferenças MLOps vs LLMOps**

Armadilhas Comuns

- ❌ Product Owner remove impedimentos → ✅ Scrum Master faz isso
- ❌ Horizontal slicing entrega valor → ✅ Vertical slicing entrega
- ❌ SAST scans apps em execução → ✅ DAST faz isso (SAST = static/source)
- ❌ Story points são tempo → ✅ Medida relativa
- ❌ Blue/Green divide tráfego → ✅ Canary divide (Blue/Green faz switch)
- ❌ Feature toggles precisam deployment → ✅ Não precisa deployment

Foco em Questões Práticas

- Escrever user stories com critérios de aceitação (Given-When-Then)
- Desenhar pipelines CI/CD com fases e gates de segurança
- Explicar abordagens de refactoring para código legacy
- Comparar estratégias de deployment com casos de uso
- Descrever práticas DevSecOps por fase SDLC

📝 Referência Rápida de Acrónimos

- **SAST**: Static Application Security Testing
- **DAST**: Dynamic Application Security Testing
- **IAST**: Interactive Application Security Testing
- **RASP**: Runtime Application Self-Protection
- **SCA**: Software Composition Analysis
- **IaC**: Infrastructure as Code
- **SIEM**: Security Information & Event Management
- **UEBA**: User & Entity Behavior Analytics
- **SOAR**: Security Orchestration, Automation & Response
- **RAG**: Retrieval Augmented Generation
- **MCP**: Model Context Protocol
- **DORA**: DevOps Research & Assessment

- **MTTR**: Mean Time To Recover
- **MTTD**: Mean Time To Detect

14. Conceitos Adicionais Importantes

Threat Modeling (STRIDE)

- **S**poofing (Falsificação) → Autenticação (MFA)
- **T**ampering (Adulteração) → Integridade (Assinaturas Digitais)
- **R**epudiation (Repúdio) → Não-Repúdio (Audit Logs)
- **I**nfo Disclosure (Divulgação Info) → Confidencialidade (Encriptação)
- **D**oS (Denial of Service) → Disponibilidade (Rate Limiting)
- **E**levation of Privilege → Autorização (Least Privilege)

Definition of Done (DoD)

- Conjunto de condições acordadas com stakeholders
- Funcionalidade não está "terminada" até passar todos critérios
- Exemplo: código revisto, scans passados, testes executados

Velocity (Velocidade)

- Soma dos Story Points completados por Sprint
- Usada para previsão de capacidade futura
- Melhora com tempo à medida que equipa amadurece

Epic vs Initiative

- **Epic**: Grande user story, não cabe numa Sprint
- **Initiative**: Objetivo de alto nível, maior que épico, alinha com estratégia de negócio

15. Padrões Anti-Pattern a Evitar

Monitoring Anti-Patterns

1. **Tool Obsession**: Perseguir ferramentas sem outcomes definidos
2. **Monitoring-as-a-Job**: Uma equipa "dona" de todo monitoring
3. **Checkbox Monitoring**: Métricas superficiais para compliance

4. **Monitoring as a Crutch**: Adicionar alertas para compensar sistemas frágeis
5. **Manual Configuration**: Configuração manual de checks/dashboards

DevOps Anti-Patterns

- **"Throwing over the wall"**: Dev atira release para Ops sem colaboração
- **Snowflake servers**: Servidores que divergem ao longo do tempo
- **Manual deployments**: Deployments feitos à mão, propensos a erro
- **No version control**: Configurações fora de controlo de versão

16. Práticas de Código & Qualidade

Boy Scout Rule

"Deixa sempre o código mais limpo do que o encontraste"

- Não reescrever tudo, fazer melhorias incrementais
- Ao corrigir bug, melhorar também área circundante
- Aplicar refactoring seguro com testes

Pre-commit Hooks - Casos de Uso

1. **Secret Scanning**: Bloquear commits com API keys, passwords
2. **Linting**: Forçar standards de código automaticamente
3. **Formatting**: Garantir formatação consistente
4. **Test Execution**: Executar testes unitários antes de commit

Code Review - Checklist de Segurança

- ✓ Este código trata input não confiável?
- ✓ Estamos a fazer log de dados sensíveis (PII)?
- ✓ O utilizador está autorizado para esta ação específica?
- ✓ Há validação de input adequada?
- ✓ Há gestão correta de erros?

17. Observability em Detalhe

Spans (Distributed Tracing)

- **Span** = operação dentro de uma transação
- Contém:
    - Identificador do Span pai
    - Nome da operação
    - Timestamps início/fim
    - Atributos (key-value pairs)
    - Eventos com timestamps

OpenTelemetry (OTel)

- Framework vendor-neutral para telemetria
- Gera e exporta traces, metrics, logs
- Integração com Spring Boot via starter
- Protocolo OTLP para exportação

SIEM + UEBA

- **SIEM**: Centraliza logs de toda a stack
- **UEBA**: Machine Learning para estabelecer "baseline de normal"
- Deteta desvios comportamentais (ex: acesso a ficheiros HR às 2 AM domingo)

18. Deployment & Release Patterns

Immutable Infrastructure

- **Não fazer patch** a servidores live → **Substituí-los**
- Se vulnerabilidade encontrada, deploy imagem nova e patched
- Previne "Configuration Drift"
- Backdoors de hackers são destruídos no próximo deploy

Safe Deployment Strategies

- **Canary Deployments**: Release para 1% utilizadores primeiro
- **Blue/Green Deployment**: Manter dois ambientes para rollback instantâneo
- **Progressive Rollout**: Aumentar gradualmente percentagem de utilizadores

Secrets Management - Vault Pattern

1. App inicia e autentica com Vault (usando IAM role)
2. Vault gera credencial temporária (válida 1 hora)
3. App usa credencial
4. Credencial expira automaticamente

- Se hacker rouba password, é inútil uma hora depois

19. Testing Strategies - Detalhes

Fuzz Testing

- Bombardear APIs com dados aleatórios, malformados, inesperados
- Trigger edge cases que developers nunca testaram
- Encontra: crashes, memory leaks, exceções não tratadas

Mutation Testing

- Introduzir pequenas mudanças no código (mutações)
- Executar testes
- Se testes continuam a passar → testes inadequados
- Mede qualidade dos testes, não do código

Contract Testing

- Testar contratos entre serviços
- Garantir que consumer e provider concordam na API
- Ferramentas: Pact, Spring Cloud Contract

20. MLOps - Conceitos Adicionais

Data Drift vs Model Drift

- **Data Drift**: Distribuição dos dados de input muda
- **Model Drift**: Performance do modelo degrada ao longo do tempo
- **Concept Drift**: Relação entre input e output muda

Feature Store

- Repositório centralizado para features
- Gestão, serving e tracking de evolução
- Garante consistência entre training e inference

Model Registry

- Repositório para armazenar modelos ML
- Versões e artefactos
- Metadata sobre treino, performance, lineage

21. Security - Defesa em Profundidade

Layers de Segurança

1. **Network**: Firewalls, segmentação
2. **Application**: WAF, RASP, input validation
3. **Data**: Encriptação at rest e in transit
4. **Identity**: MFA, SSO, IAM
5. **Monitoring**: SIEM, alertas, incident response

Policy as Code

- Desacoplar lógica de decisão da lógica de aplicação
- Gatekeeper: corre antes de deployment
- Exemplos de regras:
    - Deployments só para 'EU-West' (GDPR)
    - Developer não pode lançar instância > t2.large
    - Container images de Trusted Registry

Breaking the Build - Quando Fazer?

- **Fazer**: High Confidence + High Severity
- **Não fazer muito**: False positives → developers fazem bypass
- **Apenas para**: Vulnerabilidades críticas validadas

22. Feedback Loops & Continuous Learning

The Feedback Loop (Deploy Phase)

- Incidentes em produção → automaticamente viram tickets no backlog "Plan"
- Dados de ataques reais informam próxima sessão Threat Modeling
- **Blameless Post-Mortems**: Aprender sem culpar
- Findings viram novas user stories (ex: "Fomos hackeados por X, adicionar check para X no CI")

Continuous Compliance

- Scanners automatizados verificam ambiente live 24/7
- Standards: GDPR, HIPAA, PCI-DSS
- Alerta imediatamente se configuração "drifts" out of compliance

23. Context Window & Agents (LLMs)

Context Window Management

- **Problema**: LLMs esquecem partes antigas quando limite atingido
- **Solução**:
    - Chunking de documentos
    - RAG (Retrieval Augmented Generation)
    - Summarização
    - Passar apenas partes relevantes

MCP Protocol (Model Context Protocol)

- Host AI cria conexões one-to-one com cada MCP Server
- Cada server fornece capacidades isoladas
- Servers podem ser locais ou remotos
- Exemplo: IDE conecta com GitHub, JIRA, Observability tools

Agents - Definition

- Sistema de software que age em nome de utilizadores
- Realiza tarefas de forma autónoma
- **Tools**: Recursos externos ou funções chamadas por agents

24. Advanced Testing Concepts

Testing in Production - Como Fazer Seguramente?

1. **Feature Toggles**: Ativação precisa via query params, headers, cookies
2. **Test Data Strategy**: Criar e sinalizar dados sintéticos
3. **Prevent Leakage**: Definir política API para test data
4. **Debugging in Prod**: Source maps para diagnóstico frontend
5. **Lean Preprod**: Manter infra mínima para checks automatizados

Static Testing - Tipos

- **Linter**: ESLint (JavaScript), Checkstyle (Java)
- **Type Checker**: TypeScript, mypy (Python)
- **Formatter**: Prettier, Black
- **Security**: SonarQube, Snyk

25. Infrastructure Patterns

Pets vs Cattle - Em Detalhe

Pets:

- Nomes individuais (db-server-prod-01)
- Mantidos amorosamente, diagnosticar e corrigir manualmente
- Insubstituíveis, falha causa crise
- Configurados manualmente, tweaks acumulados
- Long-lived (anos)

Cattle:

- Numerados (web-server-001, web-server-002)
- Descartáveis, terminar e substituir
- Auto-provisionados de templates
- Efémeros, curta duração
- Escaláveis automaticamente
- Self-healing, sistema substitui falhas sem intervenção humana

Load Balancing - Métodos

- **Round-Robin**: Distribuição circular
- **Least Connections**: Servidor com menos conexões
- **IP Hash**: Mesmo cliente → mesmo servidor
- **Weighted**: Servidores com diferentes capacidades

26. CI/CD Pipeline - Exemplo Completo

Pipeline Spring Boot + React

```
Stages:
1. Build
  - Checkout código
  - Compilar backend (Maven/Gradle)
  - Build frontend (npm build)
  - Run unit tests

2. Test
  - Integration tests
  - SAST scan
  - Dependency check (SCA)

3. Security
  - Container image scan
  - IaC scan
  - Secret detection

4. Deploy (Staging)
  - Build Docker images
  - Push to registry
  - Deploy to staging
  - DAST scan
  - Smoke tests

5. Deploy (Production)
  - Manual approval gate
  - Blue/Green switch
  - Monitoring alerts
  - Rollback capability
```

27. Happiness Metric (Retrospective)

Perguntas a Fazer (Escala 1-5)

1. Como te sentes sobre o teu papel na empresa?
2. Como te sentes sobre a empresa como um todo?
3. Porquê sentes assim?
4. O que te faria mais feliz na próxima Sprint?

Porquê Importante:

- Equipas felizes são mais produtivas e criativas
- Medir happiness ajuda a identificar mudanças que aumentam moral
- Identificar problemas cedo antes de se tornarem críticos

28. Legacy Code - Black Boxes to Modern Blueprints

Multi-Lens Approach

Lens 1: UI Layer

- Artefactos: ASP files, JS, screenshots
- Info: User workflows, field names, validação client-side

Lens 2: Data Layer

- Artefactos: DB Schema, Stored Procedures
- Info: Entidades de negócio, regras encoded em SQL

Lens 3: Opaque App Tier

- Artefactos: Binaries compilados (.dll, .exe)
- Info: Core da black box, lógica de negócio

AI-Assisted Binary Archaeology

1. Não alimentar todo ASM ao LLM (overflow de contexto)
2. Fazer slice do DLL baseado em área funcional
3. Triangular: UI + DB + Binary
4. Preservar Lineage: Rastrear cada insight ao artefacto fonte

5. Human-in-the-Loop: Validar hipóteses da AI

29. Performance & Scalability

Scalability Types

- **Vertical Scaling (Scale Up)**: Adicionar CPU/RAM a servidor existente
- **Horizontal Scaling (Scale Out)**: Adicionar mais servidores
- **Auto-Scaling**: Ajustar recursos baseado em demand

Load Balancer - Health Checks

- Deteta e remove servidores failing de rotation
- Previne pedidos enviados para servidores unhealthy
- Pode verificar: HTTP status, response time, custom endpoints

Availability Calculation

- 99.9% (three nines) = 43.2 min downtime/mês
- 99.99% (four nines) = 4.32 min downtime/mês
- 99.999% (five nines) = 26 sec downtime/mês

30. Checklists Úteis

Checklist Acceptance Criteria Deve Cobrir

✓ Cenários negativos da funcionalidade ✓ Casos de uso funcionais e não-funcionais ✓ Preocupações e guidelines de performance ✓ O que sistema/feature pretende fazer ✓ Fluxo end-to-user ✓ Impacto da user story noutras features