ATIVIDADE:: OUTROS CASOS NOTÁVEIS...

- Pesquise sobre outros casos de grande repercussão:
  - Bug do Milênio
  - Queda do sistema de informática da British Airways
  - Interrupção do Serviço AWS S3 da Amazon
  - CrowdStrike e Microsoft: entenda a interrupção cibernética que deu 'tela azul' em vários países
- Responda as questões no slide anterior considerando estes casos.

Ciberataque WannaCry de 2017 -> Esse ataque cibernético global afetou mais de 200.000 computadores em cerca de 150 países, causando grandes prejuízos financeiros e operacionais em diversas organizações, incluindo hospitais, bancos e empresas de infraestrutura crítica. O WannaCry explorou uma vulnerabilidade no sistema operacional Windows, conhecida como EternalBlue, que foi desenvolvida pela NSA e vazada por hackers.

Liberdade Prematura de Prisioneiros nos EUA -> Em dezembro de 2015, um erro no software que calculava a sentença dos prisioneiros levou à libertação antecipada de mais de 3.200 prisioneiros nos Estados Unidos. O software, introduzido em 2002, foi responsável por calcular a liberdade condicional com base no comportamento dos detentos.

TSB Bank (Reino Unido) -> Em 2018, uma atualização de TI no TSB Bank resultou em uma interrupção prolongada dos serviços bancários online e móveis. Clientes tiveram suas contas bloqueadas e, em alguns casos, puderam visualizar detalhes das contas de outras pessoas. A falha ocorreu durante a migração para uma nova plataforma bancária após a divisão do Lloyds Banking Group.

1. Quais são os principais vilões nessas histórias?

- WannaCry:
- Vulnerabilidade EternalBlue (desenvolvida pela NSA e vazada).
- Cibercriminosos (criação e disseminação do ransomware).
- Falta de atualizações de segurança nos sistemas Windows.
- TSB Bank:
- Complexidade da migração de sistemas.
- Falhas na gestão de projetos de TI.

- Testes insuficientes.
- Falta de preparação para problemas inesperados.
- Libertação prematura nos EUA:
- Erro no software de cálculo de sentenças.
- Falta de validação dos resultados.
- Ausência de revisão humana.

2. O que poderia ser feito para evitar tais problemas?

- WannaCry:
- Cultura de segurança cibernética.
- Testes de invasão regulares.
- Planos de resposta a incidentes.
- Conscientização sobre phishing.
- TSB Bank:
- Planejamento cuidadoso da migração.
- Testes abrangentes e simulações.
- Envolvimento de especialistas em gestão de projetos de TI.
- Planos de contingência.
- Libertação prematura nos EUA:
- Validação dos resultados do software.
- Testes rigorosos antes da implantação.
- Revisão humana dos cálculos.
- Monitoramento contínuo.

3. Como você avalia a qualidade desses softwares?

- WannaCry: Não se aplica (ataque explorando vulnerabilidade).
- TSB Bank: Qualidade deficiente do software e processo de migração.
- Libertação prematura nos EUA: Baixa qualidade (erro crítico no cálculo de sentenças).

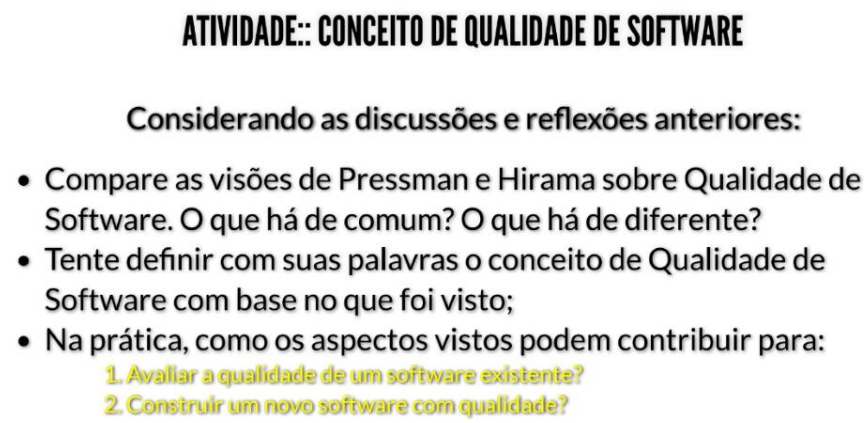
4. Qual é a relação entre os casos anteriores?

- Todos são exemplos de falhas de software com impacto significativo.
- Demonstram a importância da qualidade do software, segurança cibernética e gestão de projetos de TI.

- Destacam a necessidade de testes, validação e planejamento de contingência.

5. Quais foram os impactos?

- WannaCry:
- Prejuízos financeiros globais.
- Interrupção de serviços essenciais.
- Exposição de dados confidenciais.
- TSB Bank:
- Interrupção dos serviços bancários.
- Clientes impossibilitados de acessar contas.
- Danos à reputação do banco.
- Libertação prematura nos EUA:
- Libertação antecipada de prisioneiros.
- Risco à segurança pública.
- Necessidade de recaptura.



1. Compare as visões de Pressman e Hirama sobre Qualidade de Software. O que há de comum? O que há de diferente?

- Comum: Ambos reconhecem a importância da qualidade para o sucesso do software e a necessidade de atender às necessidades do cliente.
- Diferente: Pressman foca em processos e métricas de engenharia, enquanto Hirama enfatiza a cultura organizacional e a satisfação do cliente.

2. Tente definir com suas palavras o conceito de Qualidade de Software com base no que foi visto.

Qualidade de software é o grau em que um software atende aos requisitos, satisfaz o usuário, funciona de forma confiável e se adapta a mudanças, dentro de um contexto de processos bem definidos e cultura de melhoria contínua.

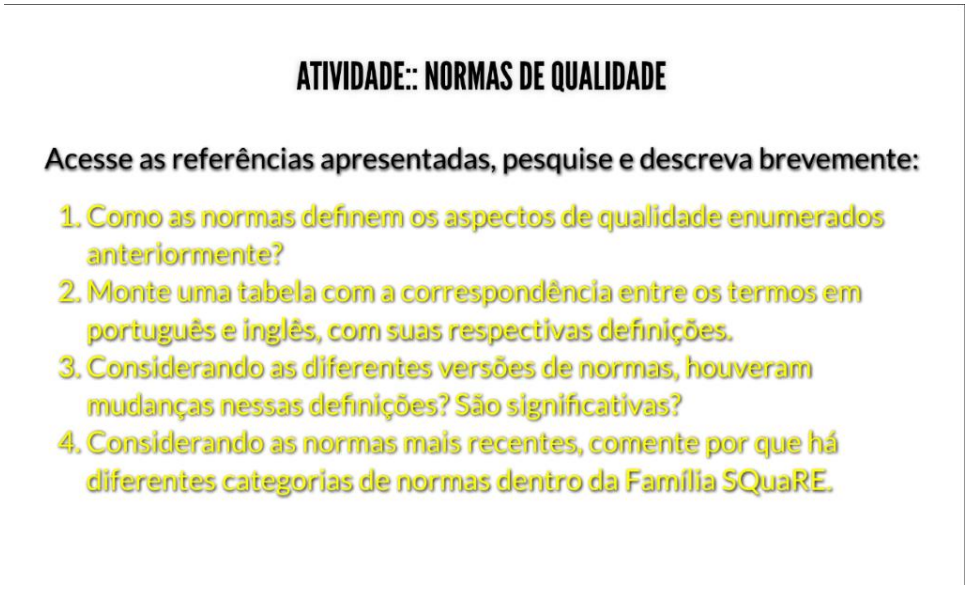
3. Na prática, como os aspectos vistos podem contribuir para:

3.1. Avaliar a qualidade de um software existente?

- Verificação de requisitos.
- Testes abrangentes.
- Coleta de métricas.
- Feedback do cliente.

3.2. Construir um novo software com qualidade?

- Planejamento claro.
- Design modular.
- Codificação limpa.
- Testes contínuos.
- Revisões de código.
- Integração contínua.
- Cultura de qualidade.



ATIVIDADE:: NORMAS DE QUALIDADE

Acesse as referências apresentadas, pesquise e descreva brevemente:

1. Como as normas definem os aspectos de qualidade enumerados anteriormente?
2. Monte uma tabela com a correspondência entre os termos em português e inglês, com suas respectivas definições.
3. Considerando as diferentes versões de normas, houveram mudanças nessas definições? São significativas?
4. Considerando as normas mais recentes, comente por que há diferentes categorias de normas dentro da Família SQuaRE.

1. Definição dos Aspectos de Qualidade pelas Normas

As normas ISO definem os aspectos de qualidade de software de forma estruturada e abrangente, utilizando modelos e métricas para avaliar e garantir a qualidade em diferentes fases do ciclo de vida do software. Elas estabelecem critérios para:

- Funcionalidade: Capacidade do software de atender às necessidades do usuário e realizar as funções especificadas.
- Confiabilidade: Capacidade do software de manter o desempenho especificado sob condições estabelecidas por um período de tempo determinado.
- Usabilidade: Facilidade com que os usuários podem aprender a operar, utilizar e se sentir atraídos pelo software.
- Eficiência: Capacidade do software de utilizar recursos de forma otimizada, como tempo de processamento, memória e largura de banda.
- Manutenibilidade: Facilidade com que o software pode ser modificado, corrigido, adaptado ou aprimorado.
- Portabilidade: Capacidade do software de ser transferido de um ambiente para outro.
- Segurança: Capacidade do software de proteger informações e dados contra acesso não autorizado, uso indevido ou divulgação.

2. Tabela de Correspondência de Termos (Português/Inglês)

Termo em Português	Termo em Inglês	Definição
Qualidade de Software	Software Quality	Grau com que um software satisfaz as necessidades especificadas e implícitas, incluindo conformidade com requisitos funcionais, desempenho, confiabilidade, usabilidade e segurança.
Requisito	Requirement	Condição ou capacidade que um software deve atender ou possuir.
Teste	Testing	Processo de avaliação de um software para detectar erros, falhas ou não conformidades com os requisitos.
Manutenção	Maintenance	Modificação de um software após a entrega para corrigir defeitos, melhorar o desempenho ou adaptar a novas necessidades.
Métrica	Metric	Medida quantitativa utilizada para avaliar um aspecto específico do software ou do processo de desenvolvimento.
Conformidade	Conformity/Compliance	Grau em que um software atende aos requisitos e normas estabelecidas.

3. Mudanças nas Definições ao Longo das Versões das Normas

Sim, as definições de qualidade de software nas normas ISO sofreram mudanças ao longo das diferentes versões. Essas mudanças geralmente refletem:

- A evolução das tecnologias e metodologias de desenvolvimento de software.
- A crescente importância da experiência do usuário (UX) e da segurança da informação.
- A necessidade de abordar novos desafios, como a complexidade dos sistemas distribuídos e a proliferação de dispositivos móveis.

As mudanças nas definições podem ser significativas, pois afetam a forma como a qualidade do software é avaliada, medida e gerenciada.

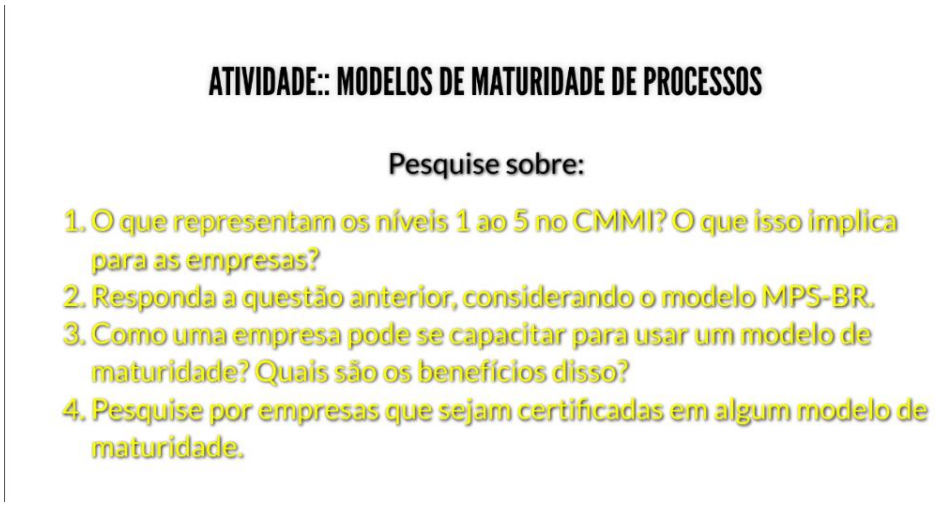
4. Categorias de Normas na Família SQuaRE

A família de normas SQuaRE (Software Quality Requirements and Evaluation) é dividida em diferentes categorias para abordar aspectos específicos da qualidade de software. As principais categorias incluem:

- Planejamento e Gerenciamento: Normas que estabelecem os princípios e processos para o planejamento, gerenciamento e controle da qualidade do software.
- Modelos de Qualidade: Normas que definem os modelos de qualidade a serem utilizados para avaliar as características e atributos do software.

- Medição da Qualidade: Normas que especificam as métricas e métodos para medir a qualidade do software.
- Requisitos de Qualidade: Normas que fornecem diretrizes para a definição e especificação dos requisitos de qualidade do software.
- Avaliação da Qualidade: Normas que descrevem os processos e métodos para avaliar a qualidade do software em diferentes fases do ciclo de vida.

A divisão em categorias permite que as normas SQuaRE sejam mais específicas e relevantes para diferentes contextos e necessidades.



1. **O que representam os níveis 1 ao 5 no CMMI? O que isso implica para as empresas?**

- Nível 1 - Inicial: Os processos são imprevisíveis, mal controlados e reativos. O sucesso depende do esforço individual e não é replicável. Implica que a empresa tem alta probabilidade de falhas, custos elevados e dificuldade em cumprir prazos e orçamentos.
- Nível 2 - Gerenciado: Os processos são gerenciados em nível de projeto, com planejamento, monitoramento e controle. Há um certo nível de disciplina, mas ainda com variações. Implica que a empresa consegue repetir sucessos em projetos similares, mas ainda não tem uma abordagem padronizada.
- Nível 3 - Definido: Os processos são padronizados e documentados em toda a organização. Há um conjunto de processos bem definidos que são adaptados para cada projeto. Implica que a empresa tem maior previsibilidade, qualidade e eficiência em seus projetos.

- Nível 4 - Quantitativamente Gerenciado: Os processos são medidos e controlados usando técnicas estatísticas. Há metas de qualidade e desempenho que são monitoradas e usadas para melhorar os processos. Implica que a empresa consegue tomar decisões baseadas em dados, otimizar seus processos e reduzir a variabilidade.
- Nível 5 - Otimizado: A empresa está focada na melhoria contínua dos processos, usando técnicas de inovação e otimização. Há uma cultura de aprendizado e adaptação. Implica que a empresa consegue responder rapidamente às mudanças do mercado, inovar e manter sua vantagem competitiva.

2. **Responda a questão anterior, considerando o modelo MPS-BR.**

O MPS-BR (Melhoria de Processo do Software Brasileiro) é um modelo similar ao CMMI, adaptado à realidade das empresas brasileiras. Os níveis de maturidade do MPS-BR também variam, mas podem ser estruturados de forma diferente, focando em aspectos específicos do contexto nacional. É importante consultar a documentação oficial do MPS-BR para obter detalhes precisos sobre cada nível e suas implicações.

3. **Como uma empresa pode se capacitar para usar um modelo de maturidade? Quais são os benefícios disso?**

- Capacitação.
- Treinamento: Invista em treinamento para seus funcionários sobre os princípios e práticas do modelo de maturidade escolhido (CMMI, MPS-BR, etc.).
- Avaliação: Realize uma avaliação inicial para identificar o nível de maturidade atual da empresa e as áreas que precisam de melhoria.
- Implementação: Desenvolva um plano de implementação gradual, com metas e prazos definidos.
- Monitoramento: Monitore o progresso da implementação e faça ajustes conforme necessário.
- Auditoria: Realize auditorias internas e externas para verificar a conformidade com o modelo de maturidade.
- Benefícios:
- Melhora na qualidade dos produtos e serviços.

- Aumento da eficiência e produtividade.
- Redução de custos.
- Melhora na satisfação do cliente.
- Maior previsibilidade e controle dos projetos.
- Melhora na imagem e reputação da empresa.
- Vantagem competitiva no mercado.

4. **Pesquise por empresas que sejam certificadas em algum modelo de maturidade.**

- Accenture: Uma empresa global de consultoria que frequentemente busca certificações CMMI para garantir a qualidade de seus processos de desenvolvimento de software e serviços.
- IBM: Outra grande empresa de tecnologia que pode ter certificações CMMI em várias de suas unidades de negócios, dependendo dos projetos e clientes.

> **Imagine que esteja iniciando um novo projeto de software:**
>
> 1. Qual seria sua estratégia para identificar as versões? Justifique.
> 2. Como nomearia a primeira versão para o público?
> 3. Após liberado o projeto, uma nova funcionalidade foi requisitada e implementada. Como nomearia esta nova versão?
> 4. Considerando a sequência anterior com o esquema de versionamento SemVer. Como ficaria o histórico de versões?

5. A partir da versão indicada antes, uma série de 3 correções, seguida por duas funcionalidades compatíveis com a versão atual e mais outras 2 correções foram publicadas em série. Qual seria a versão mais recente?
6. Uma versão nova exigiu mudanças críticas na API, foi lançada e na sequências houveram três novas versões: uma com adição de funcionalidades, seguidas de 2 com correções de bugs. Como fica o histórico?
7. Pesquise exemplos representativos de versões reais de software considerando cada um dos termos apontados.

1. **Estratégia para identificar versões:**

- Usaria o Versionamento Semântico (SemVer). Ele é claro, amplamente adotado e facilita a compreensão das mudanças entre as versões.
- A estrutura é `MAIOR.MENOR.PATCH`.
- `MAIOR`: Mudanças incompatíveis na API.
- `MENOR`: Adição de funcionalidades compatíveis com a versão anterior.
- `PATCH`: Correções de bugs.
- Justificativa: Permite que os usuários entendam o impacto da atualização antes de aplicá-la, evitando surpresas e facilitando a gestão de dependências.

2. **Nome da primeira versão para o público:**

- `1.0.0`.
- Indica que a API está estável e pronta para uso em produção.

3. **Nova versão após adicionar funcionalidade:**

- `1.1.0`.
- O número `MENOR` é incrementado, indicando a adição de uma funcionalidade compatível com a versão anterior.

4. **Histórico de versões com SemVer:**

- `1.0.0` (Versão inicial)
- `1.1.0` (Nova funcionalidade)

5. **Versão mais recente após correções e funcionalidades:**

- Partindo de `1.1.0`:
- 3 correções: `1.1.1`, `1.1.2`, `1.1.3`
- 2 funcionalidades: `1.2.0`, `1.3.0`
- 2 correções: `1.3.1`, `1.3.2`
- A versão mais recente seria `1.3.2`.

6. **Histórico após mudanças críticas na API:**

- `2.0.0` (Mudança crítica na API)
- `2.1.0` (Adição de funcionalidade)
- `2.1.1` (Correção de bug)
- `2.1.2` (Correção de bug)

7. **Exemplos de versões reais de software:**

- Linux Kernel: `5.15.0` (Grandes projetos de código aberto)
- Node.js: `16.13.0` (Plataformas de desenvolvimento)
- Angular: `13.0.0` (Frameworks web)