Resumo – Qualidade de Software

[Sistemas Criticos]

* *Sistemas Criticos*

Sistemas criticos ou **sociotécnicos** são sistemas dos quais as pessoas ou os negócios são extremamente dependentes. Caso esses sistemas falhem, os problemas serão gravíssimos.

> Existem 3 tipos de sistemas criticos que são sistemas criticos de:
1. **Segurança**: Sistemas cuja falha pode resultar em prejuízo, danos sérios ou ao ambiente.
2. **Missão**: Sistemas cuja falha pode causar problemas em objetivos, missões e etc.
3. **Negócio**: Sistemas em cuja falha pode acarretar perdas financeiras em algum negócio.

A propriedade mais importante de um sistemas crítico é a **confiança** no qual contem os seguintes atributos:
1. **Disponibilidade**: Probabilidade de que o sistema esteja pronto e em execução, capaz de fornecer serviços úteis a qualquer instante.
2. **Confiabilidade**: Probabilidade de que o sistema forneça corretamente os serviços, conforme esperado pelo usuário.
3. **Segurança**: Análise da probabilidade de um sistema causar danos para pessoas ou para o ambiente.
4. **Proteção**: Análise da probabilidade de que um sistema possa resistir a intrusões acidentais ou deliberadas.

Porque a confiança é a maior prioridade de sistemas criticos?
> **Sistemas não confiáveis são frequentemente rejeitados por seus usuários e podem causar perda de informações.**

* *Falha do Sistemas*

Existem 3 tipos de falhas que podem ocorrer em sistemas criticos:
1. O hardware pode falhar.
2. O software pode falhar.
3. Os operadores podem falhar.

> Uma pessoa comente um **erro**, que cria um **defeito** no software, que pode causar uma **falha** na operação.

Essa distinção entre erro, defeito e falha ajudando a identificar três abordagens complementares usadas para melhorar a confiabilidade de um sistema:
1. **Prevenção** de defeitos: Técnicas de desenvolvimento são usadas para minimizar a possibilidade de erros humanos antes que eles resultem na introdução de defeitos de sistema.
2. **Detecção** e remoção de defeitos: O uso de técnicas de verificação e validação aumentam as chances de detecção e remoção de defeitos antes de o sistema ser usado.
3. **Tolerância** a defeitos: São as técnicas que asseguram que os defeitos em um sistema não resultam em falhas de sistemas.

** Segurança*

O software crítico de segurança divide-se em duas classes:
1. Software crítico de **segurança primária**: Esse é um software embutido como um controlador em um sistema. O mau funcionamento do software pode causar mau funcionamento do hardware, o que resulta em danos as pessoas ou ao ambiente.
2. Software crítico de **segurança secundaria**: Esse é um software cuja falha resulta em defeitos em outros sistemas que podem ameaçar as pessoas.

Existem 4 razões pelas quais os sistemas de software que são confiáveis não são necessariamente seguros:
1. **Nós nunca podemos estar 100% certos** de que um sistema de software seja livre de defeitos ou tolerante a defeitos.
2. A especificação pode ser incompleta, sem a descrição do comportamento requerido do sistema em algumas situações críticas.
3. Maus funcionamentos de hardware podem levar o sistema a se comportar de forma imprevisível, bem como apresentar o software com um ambiente imprevisto.
4. Os operadores de sistema podem gerar entradas que por si só não são erradas, mas em algumas situações podem causar um mau funcionamento de sistema.

A chave para garantir segurança é assegurar que os acidentes não ocorram. Isso pode ser alcançado de três maneiras complementares:
1. **Prevenção de perigos**: O sistema é projetado de modo que os riscos sejam evitados.
2. **Detecção e remoção de perigos**: O sistema é projetado de modo que os perigos sejam detectados e removidos antes que resultem em um acidente.
3. **Limitação de danos**: O sistema pode incluir recursos de proteção que minimizem os danos que possam resultar em acidente.

** Proteção*

A segurança é um atributo do sistema que reflete sua capacidade de se proteger de ataques externos, sejam acidentais ou deliberados.
Esses ataques são possíveis porque a maioria dos computadores de uso geral está em rede e é, portanto, acessível a estranhos.

Existem alguns termos importantes referentes a proteção:

Termo	Definição
Ativo	Algo de valor que deve ser protegido. O ativo pode ser o próprio sistema de software ou dados usados por esse sistema.
Exposição	Possíveis perdas ou danos a um sistema de computação. Pode ser perda ou dano aos dados, ou uma perda de tempo e esforço, caso seja necessária a recuperação após uma brecha de proteção.
Vulnerabilidade	A fraqueza em um sistema computacional, que pode ser explorada para causar perdas ou danos.
Ataque	Uma exploração da vulnerabilidade de um sistema. Geralmente, vem de fora do sistema e é uma tentativa deliberada de causar algum dano.
Ameaças	Circunstâncias que têm potencial para causar perdas ou danos. Você pode pensar nisso como uma vulnerabilidade de um sistema submetido a um ataque.
Controle	Uma medida de proteção que reduz a vulnerabilidade do sistema. A criptografia é um exemplo de controle que reduz a vulnerabilidade de um sistema de controle de acesso fraco.

Em qualquer sistema de rede, existem três principais tipos de ameaças à proteção, são essas ameaças à:
1. **Confidencialidade** do sistema e seus dados: Essas ameaças podem divulgar informações para pessoas ou programas não autorizados a acessarem-nas.
2. **Integridade** do sistema e seus dados: Essas ameaças podem danificar o software ou corromper seus dados.
3. **Disponibilidade** do sistema e seus dados: Essas ameaças podem restringir, para usuários autorizados, acesso ao software ou a seus dados.

> Na prática, a maioria das vulnerabilidades em sistemas sociotécnicos resultam de falhas humanas e não de problemas técnicos.

Os controles que você pode colocar em prática para melhorar a proteção de sistema são comparáveis aqueles de confiabilidade e segurança:
1. **Prevenção de vulnerabilidade**: controles que se destinam a assegurar que os ataques não sejam bem-sucedidos.
2. **Detecção e neutralização de ataques**: controles que visam detectar e repelir os ataques.
3. **Limitação de exposição e recuperação**: controles que apoiam a recuperação de problemas.

**COCOMO*

> Hierarquia do modelo COCOMO

1. **Básico**: modelo estático de valor simples que computa o esforço (e custo) de desenvolvimento de software com uma função do tamanho de programa expresso em linhas de código estimadas.
2. **Intermediário**: computa o esforço de desenvolvimento de software como uma função do tamanho do programa e de um conjunto de "direcionadores de custo" que incluem avaliações subjetivas do produto, do hardware, do pessoal e dos atributos do projeto.

3. **Avançado**: incorpora todas as características da versão intermediária, com uma avaliação do impacto dos direcionadores de custo sobre cada passo (análise, projeto, etc.) do processo de engenharia de software.

Modos de Desenvolvimento
1. **Modo orgânico**: projetos simples, relativamente pequenos, nos quais pequenas equipes com boa experiência em aplicações trabalham em um conjunto de requisitos não tão rígidos.
2. **Modo semidestacado**: um projeto intermediário (em tamanho e complexidade) no qual equipes com níveis de experiência mistos devem atingir uma combinação de requisitos rígidos e não tão rígidos.
3. **Modo embutido**: um projeto de software que deve ser desenvolvido dentro de um conjunto rígido de restrições operacionais, de hardware e de software.

[Sistemas Embarcados e de Tempo Real]

** Contexto*

Os computadores são usados para controlar uma vasta gama de sistemas, desde máquinas domésticas simples, controladores de jogos, até plantas inteiras de fabricação. Esses computadores interagem diretamente com dispositivos de hardware. O software é embutido no hardware do sistema, muitas vezes em memória do tipo apenas leitura, e geralmente responde em tempo real a eventos no ambiente do sistema.
O software embutido é muito importante economicamente porque quase todos os dispositivos elétricos incluem software.

** Sistemas de Software x Embarcados*

A capacidade de resposta em tempo real é a diferença crítica entre sistemas embutidos e outros sistemas de software, como os sistemas de informações, os sistemas baseados em Web ou os sistemas de software pessoais, cuja principal finalidade é o processamento de dados.

** Sistemas de Tempo Real*

Para os sistemas de tempo não real, sua correção pode ser definida especificando-se como as entradas de sistema mapeiam as saídas correspondentes que devem ser produzidas pelo sistema.
Em resposta a uma entrada, deve ser gerada uma saída correspondente. Muitas vezes, alguns dados devem ser armazenados.
Em um sistema de tempo real, a correção depende tanto da resposta para uma entrada quanto do tempo necessário para gerar essa resposta.
Se o sistema demorar muito para responder, a resposta necessária poderá ser ineficaz.

> Um sistema de software de tempo real é um sistema cujo funcionamento correto depende tanto dos resultados produzidos pelo sistema quanto do tempo em que esses resultados são produzidos.

Um “Sistema de tempo real” é um sistema cuja operação é degradada se os resultados não forem produzidos em conformidade com os requisitos de tempo especificados.
A resposta no tempo certo é um fator importante em todos os sistemas embarcados, mas nem todos os sistemas embutidos exigem uma reposta muito rápida.

Existem outras diferenças importantes entre sistemas embutidos e outros tipos de sistema de software, além da necessidade de repostas em tempo real:

1. Geralmente, os sistemas embutidos executam continuamente e não param.
> Eles começam quando o hardware é ligado e devem executar até que o hardware seja desligado.
2. As interações com o ambiente do sistema são incontroláveis e imprevisíveis.
> Em sistemas interativos, o ritmo da interação é controlado pelo sistema e, ao limitar as opções de usuário, os eventos a serem processados são conhecidos antecipadamente.
> Devem ser capazes de responder a eventos inesperados a qualquer momento.

3. Podem haver limitações físicas que afetem o projeto de um sistema.
> Exemplos desse tipo incluem limitações sobre a energia disponível para o sistema e o espaço físico ocupado pelo hardware.
> Essas limitações podem gerar requisitos para o software embutido, como a necessidade de conservar a energia e, assim, prolongar a vida útil da bateria.

4. A interação direto com o hardware pode ser necessária.
> Em sistemas interativos e sistemas de informações, existe uma camada de software que esconde o hardware do sistema operacional.
> Isso é possível porque você só pode se conectar a alguns poucos tipos de dispositivos para esses sistemas, como teclados, mouse, monitores etc.

5. Questões de segurança e confiabilidade podem dominar o projeto de sistema.
> Muitos sistemas embutidos controlam dispositivos cuja falha pode ter custos humanos ou econômicos elevados.
> Nesse caso, a confiança é critica, e o projeto de sistema precisa garantir um comportamento critico de segurança em todos os momentos.

* *Projeto de Sistemas Embarcados*

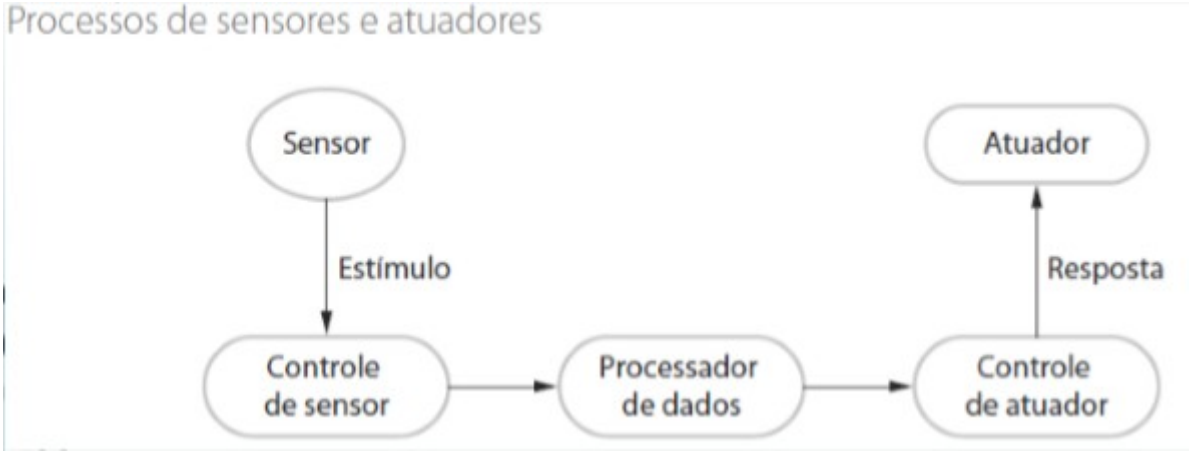
O processo de projeto para sistemas embutidos é um processo de engenharia de sistemas em que os projetistas de software devem considerar em detalhes o projeto e o desempenho do hardware de sistema.
Parte do processo de projeto do sistema pode envolver e decidir quais recursos de sistema serão implementados no software e no hardware.

Os sistemas embutidos são sistemas reativos que reagem a eventos em seu ambiente, e a abordagem geral de projeto de software embutido de tempo real é baseada em um modelo de estímulo-resposta.

O que é um estímulo?
Um estímulo é um evento que ocorre no ambiente do sistema de software que faz com que o sistema reaja de alguma forma. E é aguardada uma resposta no qual é um sinal ou mensagem enviada pelo software para seu ambiente.

Os estímulos são divididos em duas classes:
1. **Periódicos**: Estímulos que ocorrem em intervalos previsíveis.
> Por exemplo, o sistema pode examinar um sensor a cada 50 milissegundos
2. **Aperiódicos:** Estímulos que ocorrem de forma irregular e imprevisível e geralmente são sinalizados pelo mecanismo de interrupção do computador.

Os estímulos provêm de sensores no ambiente do sistema, e as respostas são enviadas aos atuadores.
Uma diretriz de projeto geral para sistemas de tempo real é ter processos separados para cada tipo de sensor e atuador.



1) Para cada tipo de sensor pode haver um processo de gerenciamento que trata coleta de dados dos sensores.
2) Os processos de processamento de dados calculam as respostas necessárias para os estímulos recebidos pelo sistema.
3) Os processos de controle de atuadores estão associados com cada atuador e gerenciam sua operação.
4) Esse modelo permite que os dados sejam rapidamente coletados do sensor, e que o processamento e a resposta do atuador associado sejam realizados mais tarde.

* Processo de Projeto de Sistemas de TR

Seleção de plataforma: nessa atividade, você escolhe uma plataforma de execução para o sistema (ou seja, o hardware e o sistema operacional de tempo real a ser usado).

Identificação de estímulos/resposta: Isso envolve a identificação dos estímulos que o sistema deve processar e a resposta ou respostas associadas a cada estímulo.

Análise de timing: para cada estímulo e resposta associada, identificam-se as restrições de tempo que se aplicam ao estímulo e ao processamento de resposta

Projeto de processo: nesse estágio, agrega-se o estímulo e a transformação da resposta em um número de processos concorrentes

Projeto de algoritmo: para cada estímulo e resposta, criam-se algoritmos para efetuar os processamentos necessários.

Projeto de dados: você especifica as informações que são trocadas por processos e os eventos que coordenam a troca de informações, e cria estruturas de dados para gerenciar essas trocas de informações.

Programação de processo: você projeta um sistema de programação que garantirá que os processos são iniciados no tempo certo para cumprirem seus deadlines.

A ordem dessas atividades de processo de projeto de software de tempo real depende do tipo de sistema a ser desenvolvido, bem como seus requisitos de processo e plataforma.