

Sistemas Tolerantes a Falhas

Introdução a Sistemas Computacionais Tolerantes a Falhas

Prof. Jó Ueyama

Sistema Computacional

- Aumentou dramaticamente em escopo, complexidade e ambiente (*pervasive computing*).
- Alguns sistemas necessitam um alto grau de complexidade
 - Em aviões, controle de tráfego aéreo, dispositivos médicos, reatores nucleares, trens de alta velocidade, serviços bancários, sistemas náuticos e militares, etc.
- As consequências das falhas destes sistemas pode ser
 - Médio ou catastrófico com até perdas de vida

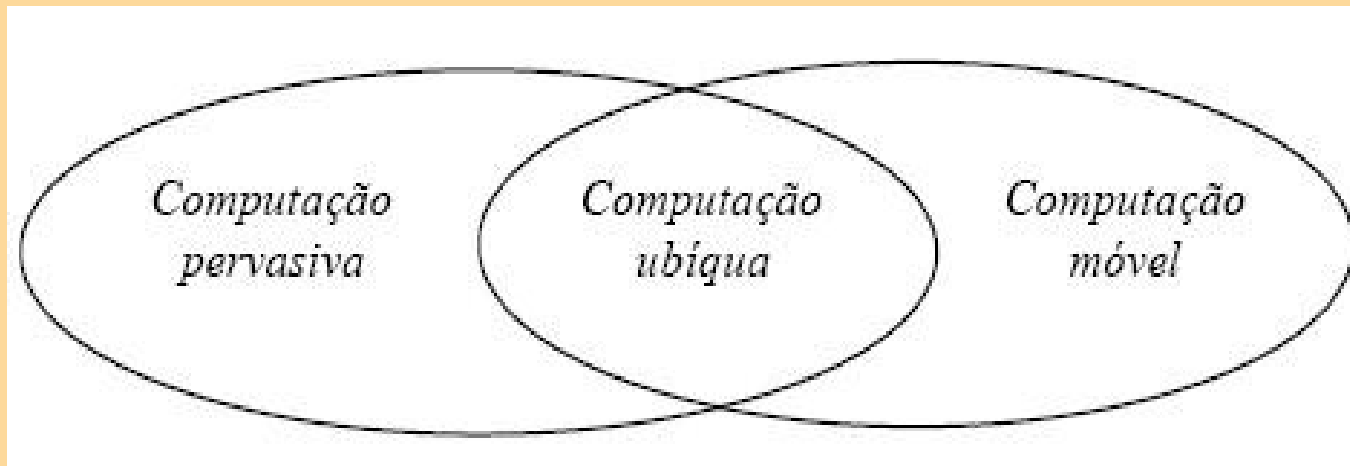


Computação Pervasiva e Ubíqua

- Computação Móvel
 - O usuário pode levar o equipamento para qualquer lugar
 - Sem perder o poder computacional
- Computação pervasiva
 - computação invisível ao usuário
 - Obter informações do ambiente no qual está inserido
 - Ajusta a aplicação para melhor atender as necessidades do usuário

Computação Pervasiva e Ubíqua

- Computação Ubíqua
 - Une a mobilidade com a onipresença da pervasiva
 - O sistema pode funcionar ajustando dinamicamente de acordo com o contexto em que ele se encontra
 - e.g. receber notícias de uma cidade em que o usuário está visitando



Sistema Computacional

- Softwares normalmente assumem a maior parte da funcionalidade destes sistemas pervasivos e ubíquos críticos
 - Erros neles podem ser catastróficos
 - Apesar dos avanços, nem todos os erros são identificados
- As IDEs ajudam a minimizar estes erros
- Erros sempre corrigidos inclusive em SOs conhecidos como o Windows através de *patches*

Exemplos de Incidentes

- Foguete Ariane V lançado pela Agência Espacial Europeia explodiu 40s após o lançamento
 - Prejuízo de meio bilhão de dólares
- Várias falhas tem sido encontradas em softwares dos aviões Air Bus A320
- Um problema de software causou um vazamento de radiação na unidade de Sellafield
- Uma falha de software no sistema do Patriot causou a perda de um míssil



Falhas em hardwares e softwares

- Existem desafios de falhas em softwares e nos hardwares
- Em hardwares
 - Normalmente, é mais previsível (durante o tempo de uso)
- Em softwares
 - É diferente, pois eles não 'deterioram' com o tempo
 - Os erros são mais de lógica, o que os tornam mais complexos
 - Não podemos simplesmente duplicar o sistema, pois isto duplicará os erros

Algumas Definições

- Falha
 - Uma causa identificada ou hipotetizada que leva a um erro (e.g. calcular errado o DV do CPF)
- Erro
 - É um estado de um sistema que pode levar a um fracasso na execução do sistema (e.g. aceitar CPFs errados)
- Considera-se que há falhas quando os erros são detectados no sistema
- Diz-se que um fracasso (*failure*) ocorre quando o serviço desvia-se do especificado



Software Confiável (*Dependable*)

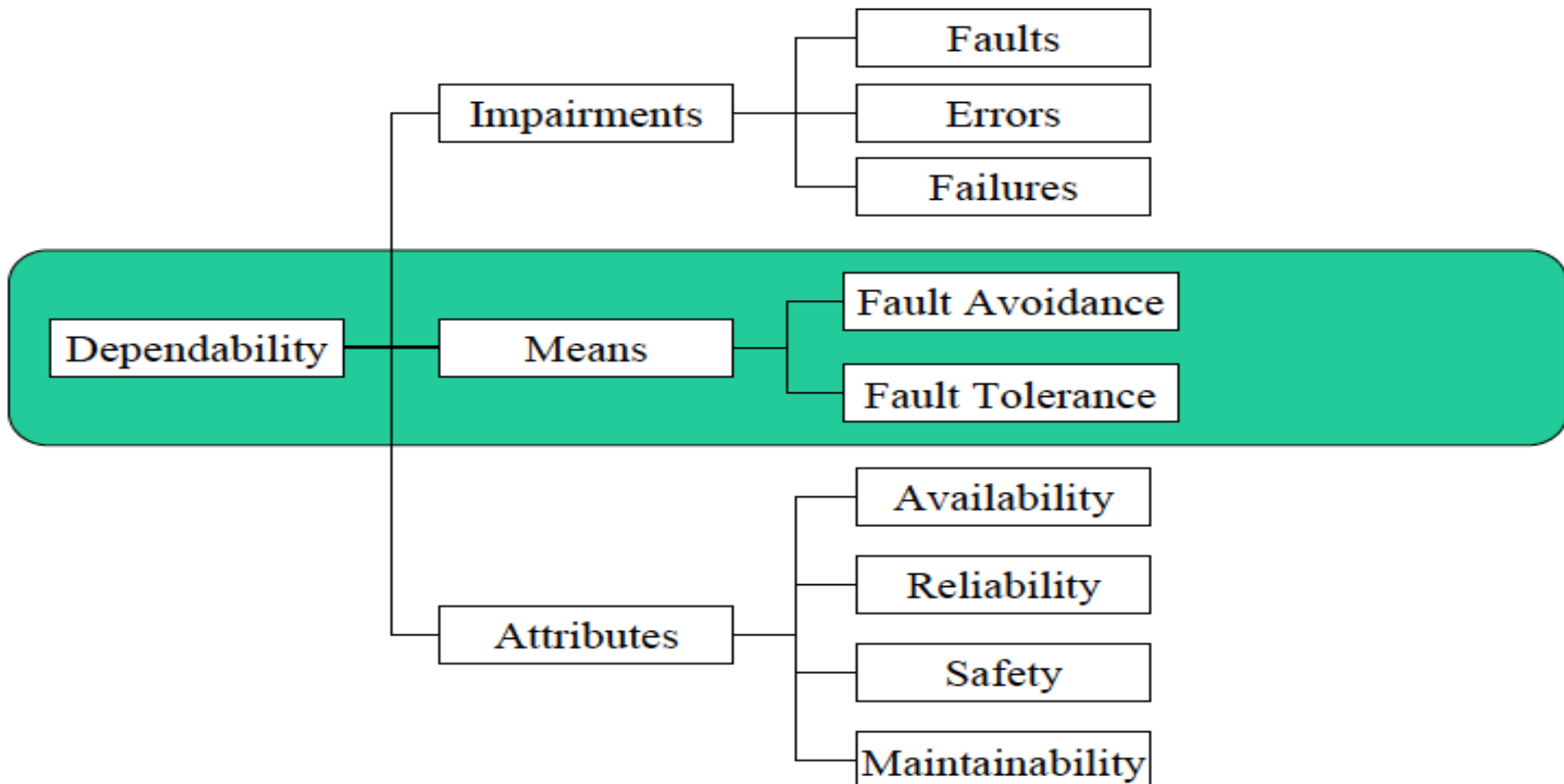
- O que leva a precisarmos de tolerância a falhas (TF)?
 - Onipresença do software (pervasivo) e o seu uso tanto em aplicações críticas e cotidianas
 - e.g. equipamentos domésticos
 - Aumento da complexidade
 - e.g. sistemas embarcados em aviões
 - Presença de software em aplicações críticas (e.g. em reatores nucleares)
 - Precisamos deles em ambientes críticos?

Software Confiável

- A árvore na próxima ilustração exibe os impedimentos, mecanismos e os atributos da confiabilidade
- Os impedimentos são as falhas, erros e o fracasso
- Os atributos são disponibilidade, confiabilidade, segurança, confiabilidade, integridade e facilidade de manutenção

Software Confiável

Dependability Concept Classification



Mecanismos para ``Dependability''

- São de dois tipos
 - Mecanismos empregados durante a construção do software, i.e., *construction*
 - *Fault avoidance* ou *prevention*: evitar que ocorram erros (e.g. verificar se o usuário digitou os dados corretamente)
 - Mecanismos que contribuem para validar o software após a sua construção, i.e., *validation*
 - *Fault removal* e *forecasting*: eliminar os erros e estimar a presença de erros e as suas consequências

Software Confiável

- Mecanismos na construção
 - *Fault avoidance*: evitar a introdução de erros
 - *Fault tolerance*: verificar o que o sistema pode garantir (dentro do especificado) apesar das falhas
- Mecanismos na validação
 - *Fault removal*: identificar a presença de falhas e eliminá-las
 - *Fault forecasting*: estimar a presença de falhas, assim como a ocorrência e as suas consequências
- A seguir, analisaremos cada mecanismo em detalhe

Fault Avoidance or Prevention

- Estes mecanismos são usados para aumentar a confiabilidade do sistema e reduzir falhas
 - utilizados durante a construção
- Faz o uso dos requisitos e das especificações do sistema
- Utiliza modelos matemáticos para prover uma maior confiabilidade do sistema
- Leva em consideração a reusabilidade de código
 - Aumenta a confiabilidade. Por que?

Fault Avoidance or Prevention

- 1) *System Requirements Specification*
 - Especificações de requisitos são muitas vezes consideradas como imperfeitas
 - Isto ocorre pela falta de comunicação entre os desenvolvedores e os que especificam o sistema
 - Muitas das ferramentas endereçam mais os erros de implementação do que os da engenharia de requisitos
 - Porém, existem ferramentas interativas que ajudam neste processo
 - Estudo de caso na construção de sistemas reais (e.g. ALEPA)

Fault Avoidance or Prevention

- 2) Projeto estruturado e técnicas de programação
 - Estas técnicas tem se mostrado eficiente
 - Reduzem a complexidade (e.g. evitar o *goto*)
 - *Decoupling* e modularização tanto na implementação e no projeto ajudam a reduzir falhas (i.e., menor interdependência entre os 'módulos')
 - Sistemas coesos ajudam neste processo
 - Outra técnica é a componentização: reusabilidade em todos os níveis (multilinguagem)
 - Todas estas técnicas ajudam a reduzir a complexidade e a resolução dos problemas
 - Levam a sistemas com menos falhas

Fault Avoidance or Prevention

- 3) Formal methods
 - Os requisitos são analisados usando-se ferramentas que traduzem-os para modelos matemáticos
 - Mais usado em pequenos projetos. Por que?
 - As especificações matemáticas são praticamente do tamanho do software
 - Porém, partes do software poderão ter os seus riscos avaliados baseados em métodos formais
 - Avaliar apenas as partes críticas

Fault Avoidance or Prevention

- 4) *Software reuse*
 - O reuso de software permite uma maior confiabilidade. Por que?
 - Objetos e componentes testados e aprovados poderão ser reusados
 - Pode trazer outras vantagens além da confiabilidade: economia no processo de desenvolvimento
 - Promove a coesão, o que aumenta a confiabilidade
 - Reuso deve ser medido de caso a caso, pois há diferentes níveis de confiabilidade requisitados em cada sistema individual.

Fault Avoidance or Prevention

- Outros mecanismos são também perfeitamente aceitos
 - Refinamento de requisitos de forma interativa com o usuário final, por exemplo
 - Utilização de bons métodos de ES (e.g. XP)
 - Implementação de códigos legíveis (e.g. documentação de código)
- A técnica de métodos formais é baseada em mecanismos muito complexos
- O reuso de software deve ser avaliado em cada caso, pois os requisitos alteram de sistema para sistema

Fault Removal

- Apesar dos esforços para evitar falhas, elas vão existir
 - Por isso, a remoção delas é necessária
- É uma técnica utilizada durante a verificação e validação
- É baseado em três técnicas
 - *Software testing*
 - *Formal inspection*
 - *Formal design proofs*

Fault Removal

- 1) Software testing
 - A técnica mais comum para remover erros
 - Dificuldade chave: custo para realizar testes exaustivos
 - Testar o software sob todas as circunstâncias e com todas as possíveis entradas
 - A chave é ter uma cobertura adequada de testes e derivar uma boa qualidade nos testes (para cada sistema)
 - Testes multiníveis: componentes mais críticos devem receber uma maior gama de testes

Fault Removal

- 2) Formal Inspection
 - É uma técnica prática bastante utilizado pela comunidade da indústria
 - Possui um rigoroso processo acompanhado de uma documentação que foca em
 - Examinar o código fonte
 - Encontrar falhas
 - Corrigi-las
 - Certificar que eles foram retiradas
 - Esta técnica é realizada por um grupo de pessoas antes da fase de testes

Fault Removal

- 3) *Formal Design Proofs*
 - Esta técnica está relacionada com a dos métodos formais (*fault avoidance and prevention*)
 - Usa modelos matemáticos para assegurar a corretude do programa
 - e.g. fornecendo uma bateria de dados para testes
 - A ferramenta gera *test cases* automaticamente
 - Não encontra-se consolidado ainda
 - Alto custo para sistemas robustos
 - Adequado para partes do software tidos como críticas, durante a avaliação de riscos

Fault Removal

- As técnicas de *test* e *formal inspection* são largamente utilizadas
 - Podem ser mais enfatizadas em partes consideradas como críticas em termos de confiabilidade
 - Não é possível testar de forma exaustiva
 - Os testes podem acusar a presença de falhas, mas não a sua ausência
 - As falhas só serão identificados quando o sistema estiver em operação
- A técnica de *Formal Design Proof* é custosa e complexa

Fault Forecasting

- Remoção de falhas é imperfeita e por isso a previsão de falhas é necessária
- A previsão de falhas é mais uma técnica para aumentar a confiabilidade do sistema
- Ela estima a presença, a ocorrência e as consequências das falhas
- Estuda o ambiente operacional do sistema para a previsão
- Utiliza também um conjunto de dados para gerar falhas de forma 'proposita'

Fault Forecasting

- 1) Estimativa da Confiabilidade
 - Utiliza técnicas estatísticas para inferir o grau de confiabilidade do sistema
 - Em cada estágio temporal aplica-se tal técnica estatística
 - Verificado/estimado durante os testes e durante a operação do sistema
 - Ele provê uma visão do grau de confiabilidade que um sistema tem em um determinado estágio

Fault Forecasting

- 2) Previsão de confiabilidade
 - Esta técnica é aplicada em vários estágios de desenvolvimento de software
 - Prevê a confiabilidade futura do software usando métricas e medidas pertencentes a cada domínio
 - A medida que as falhas ocorrem, modelos de confiabilidade podem ser utilizados para analisá-las
 - Mais utilizado após o início dos primeiros testes realizados no sistema

Fault Forecasting

- Estas técnicas verificam através de previsões futuras se eles possuem o comportamento desejado
 - Comparando-se com os requisitos definidos previamente
- A previsão revela se mais testes ou medidas/técnicas de aumento de confiabilidade são necessárias
- A previsão não é uma análise de requisitos e não aponta os requisitos que foram deixados de lado

Fault Tolerance (TF)

- TF é uma forma de aumentar a confiabilidade do software
- Técnicas que permitem que os softwares falhem após o seu desenvolvimento
- Quando tais falhas ocorrem, tais técnicas evitam que o 'fracasso' ocorra na execução
- Técnicas clássicas de TF já existem há mais de 20 anos
- Várias destas técnicas serão vistas a seguir

Fault Tolerance (TF)

- 1) *Single version software environment*
 - Nesta técnica, a TF é limitada; ela monitora as falhas seguindo os seguintes mecanismos:
 - Atomicidade das ações – verifica se alguns procedimentos críticos implementam tais ações
 - Verificação das decisões – certifica de que as decisões estão dentro das normas de segurança delineadas previamente
 - Tratamento das exceções – analisa se algumas operações possuem tratamento de exceções, obrigando em alguns casos (e.g. já implementado em algumas linguagens com o IDE)

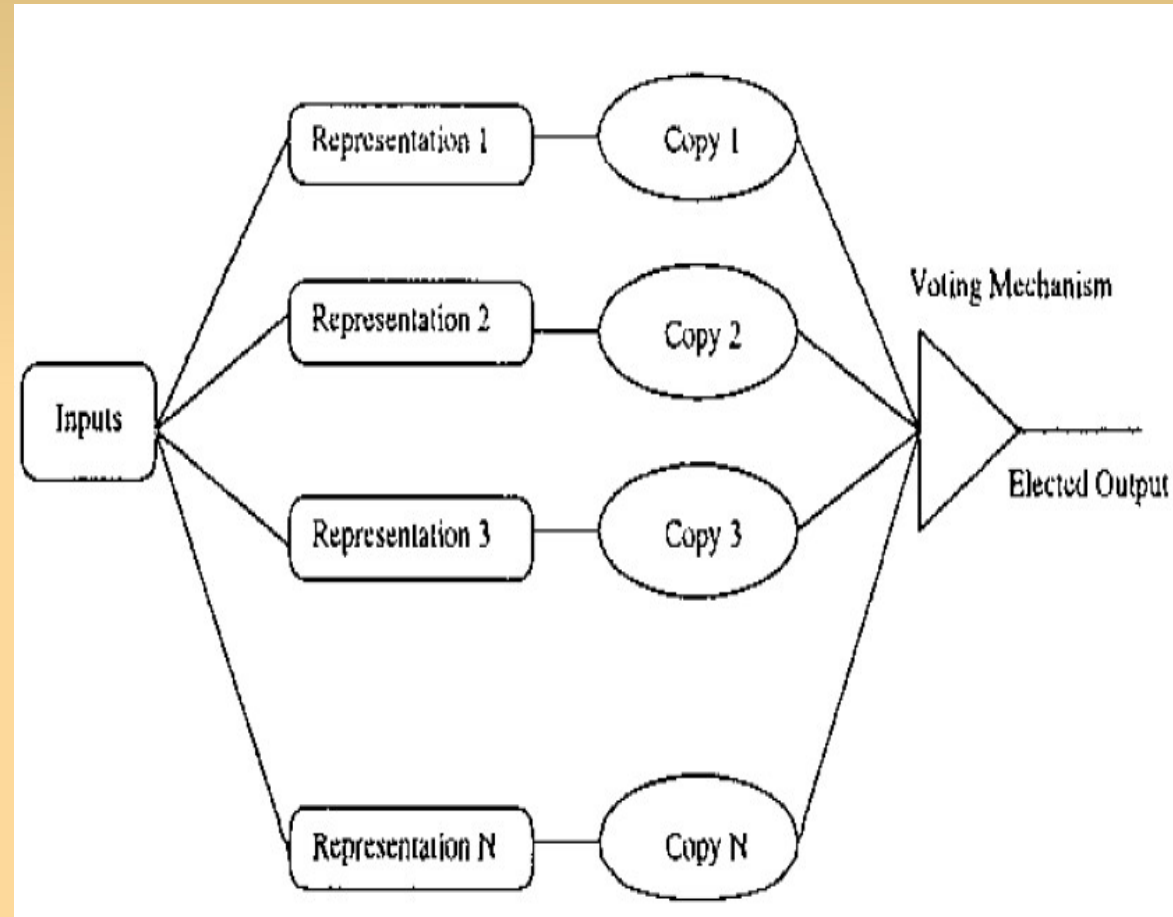
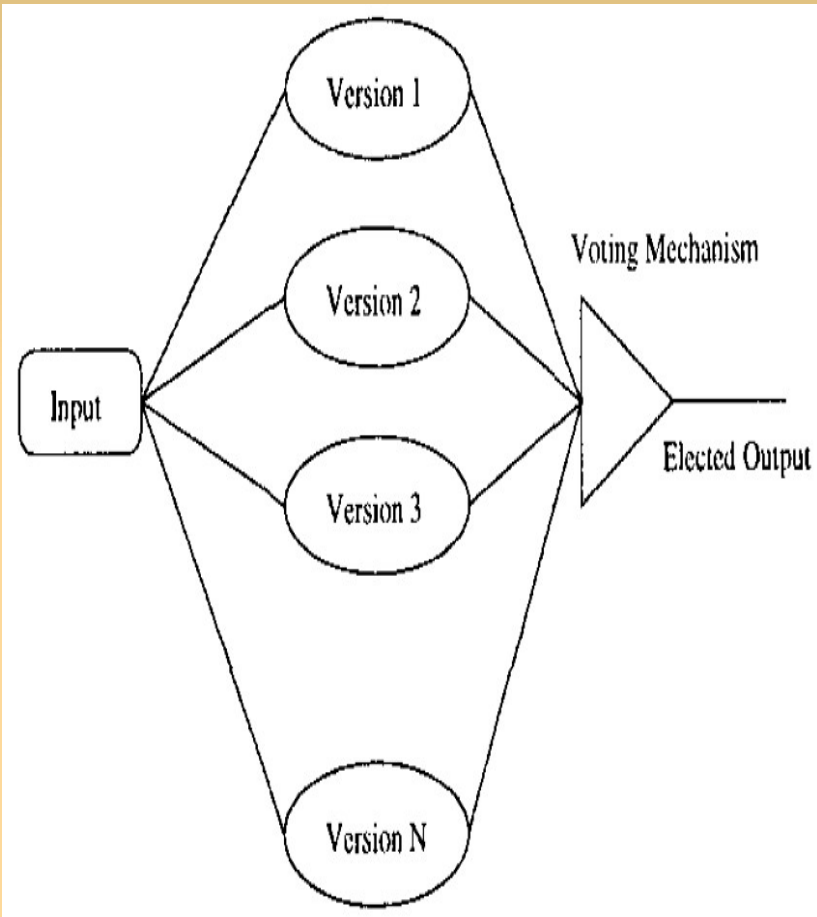
Fault Tolerance (TF)

- *2) Multiple version software environment (MVSE)*
 - Esta técnica utiliza múltiplas versões de um mesmo software desenvolvidos de forma independente
 - Existem várias técnicas que implementam o MVSE
 - Recovery blocks
 - N-version programming
 - N self-checking programming

Fault Tolerance (TF)

- 3) *Multiple data representation environment*
 - Esta técnica utiliza diferentes representações de dados de entrada para prover TF
 - Exemplos destas técnicas incluem
 - *Retry blocks* – utiliza um *acceptance test* para prosseguir com a execução dos *primary block* que processam dados com diferentes representações
 - *N-copy programming* – implementação *n-version* do data diversity
 - Estas técnicas serão revisitadas

Fault Tolerance (TF)



- Exemplos de *N-version e N-copy programmings*

Fault Tolerance (TF)

- As técnicas de TF 'toleram' falhas em sistemas após o seu desenvolvimento através de duas formas
 - *Single version*
 - *Multiple version*
 - *Design diversity*
 - *Data diversity*
- As técnicas de TF protegem a tradução de requisitos para implementação
- Mas, não 'protegem' a especificação de requisitos

Fault Tolerance (TF)

- As técnicas de TF tem sido utilizadas em diversas áreas que inclui
 - Aviação
 - Reatores nucleares
 - Saúde
 - Telecomunicação
 - Redes de sensores sem fio
 - Transporte



Exercício Prático

- Professor
 - Roteador inteligente tolerante a falhas
 - Usa o algoritmo de *bully* para eleger o líder de um cluster de sensores Sun SPOT
 - O líder é o roteador de todo o cluster de sensores
 - Desenhar o diagrama
- Cada equipe de alunos
 - Derivar uma software básico que explora a tolerância a falhas de forma inteligente
 - Deve ser adaptativo para ressaltar a flexibilidade
 - Ambiente de computação ubíqua