

A decorative pattern of hexagons in various colors (yellow, orange, green, blue, purple) arranged in a honeycomb-like structure, primarily on the left side of the slide.

NIDays09

CONFERÊNCIA TECNOLÓGICA SOBRE
PROJETO GRÁFICO DE SISTEMAS



Engenharia de software para desenvolvimento com LabVIEW:

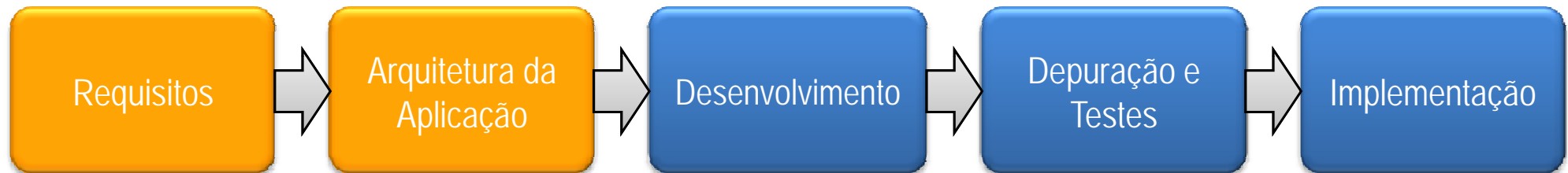
Leandro Fonseca – Gerente Distrital de Vendas
Alexsander Loula – Coordenador da Eng. de Aplicações



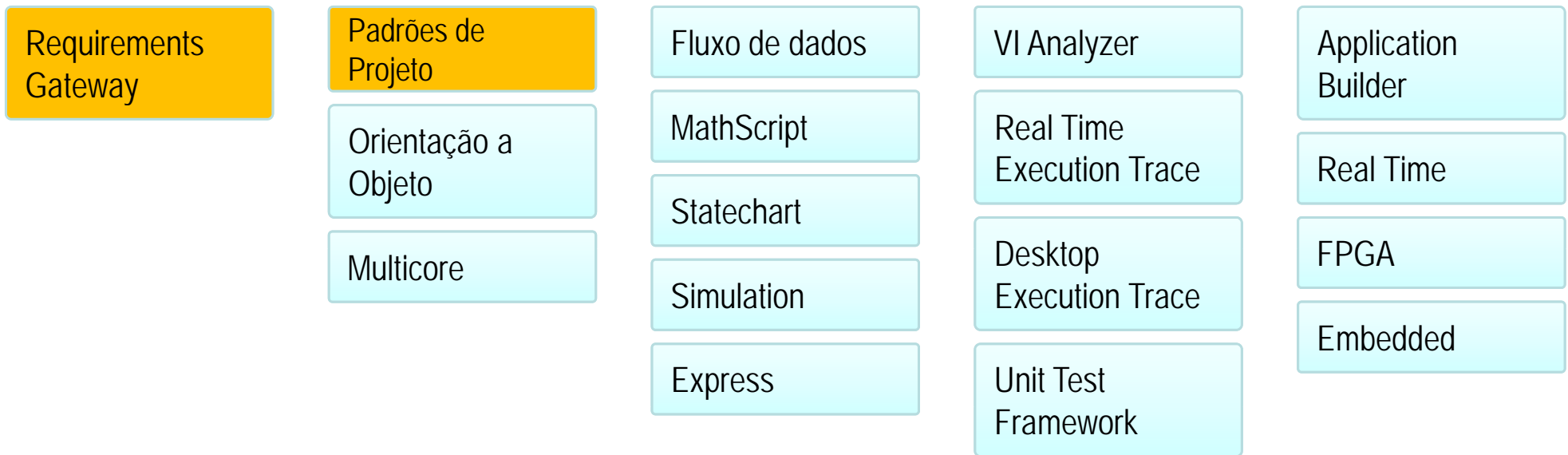
Agenda

- Desafios
- Requisitos
- Gerenciamento de Configuração
- Arquiteturas
- Resumo

Processo de Engenharia de Software



Engenharia de Software e Boas Práticas de Desenvolvimento



Desafios no desenvolvimento de sistemas complexos

- Desenvolver sistemas/software cada vez mais complexos;
- Desenvolvimento em equipes e multi-localidade;
- Desenvolvimento para multiplataformas;
- Teste, Verificação e Certificação de Sistemas (ISO, CMMI, DO, FDA, CTA, JAA, FAA...);

Como começar?

Requisitos

“Uma descrição detalhada do que o cliente deseja, escrita em suas próprias palavras... Não há como enfatizar a importância deste documento...” (Jon Connay & Steve Watts)

“Os requisitos de um sistema são descrições dos serviços fornecidos pelo sistema e as suas restrições operacionais.”
(Sommerville)

Benefícios da Documentação de Requisitos

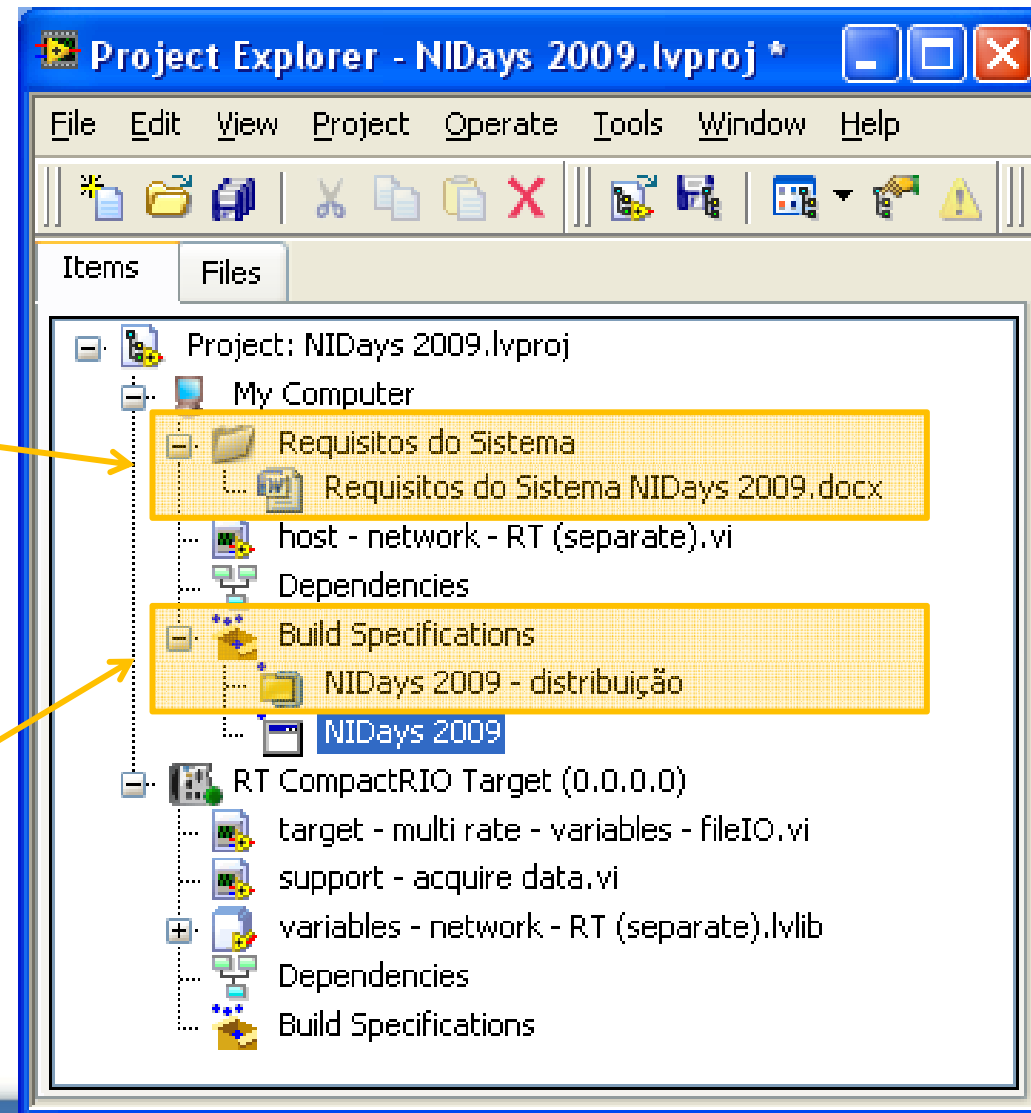
- Facilitar o entendimento entre os clientes e os fornecedores/desenvolvedores (internos e externos);
- Auxilia no gerenciamento do andamento do projeto;
- Lidar facilmente com o aumento da complexidade;
- Atender padrões da empresa, governo e indústria;
- Posteriormente utilizada criação do Plano de Testes;
- Documentação total do sistema, desde a fase de concepção.

Tipos de Requisitos

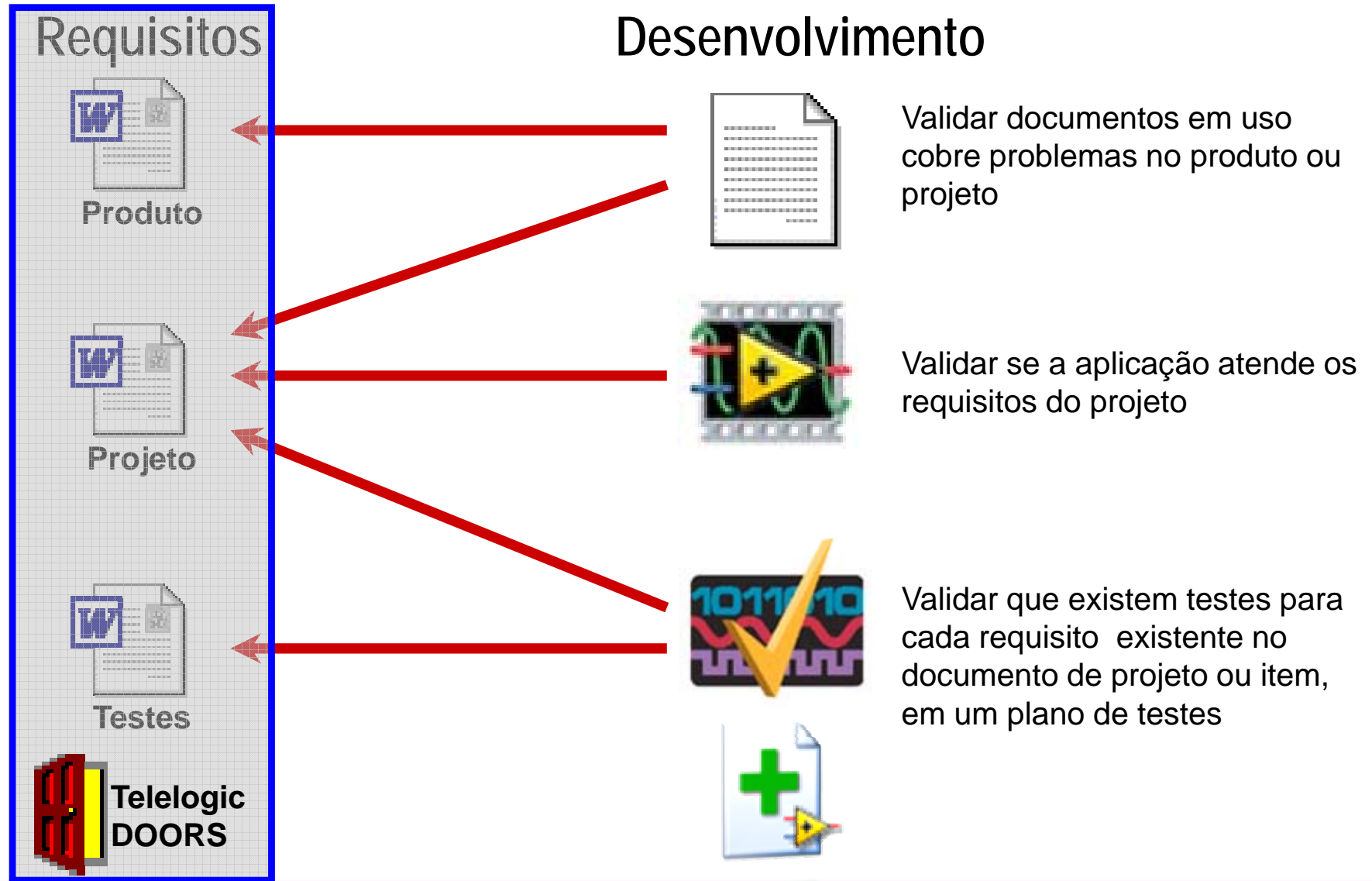
- **Funcionais:** são os serviços que o sistema deve fornecer;
 - Exemplo: Geração de relatório de teste;
- **Não Funcionais:** restrições sobre os serviços/funções do sistema;
 - Exemplo: interoperabilidade com sistemas legados;
- **De Domínio:** provenientes especificamente do domínio da aplicação do sistema.
 - Exemplo.: Sincronismo de placas de aquisição.

Integrando Requisitos no LabVIEW

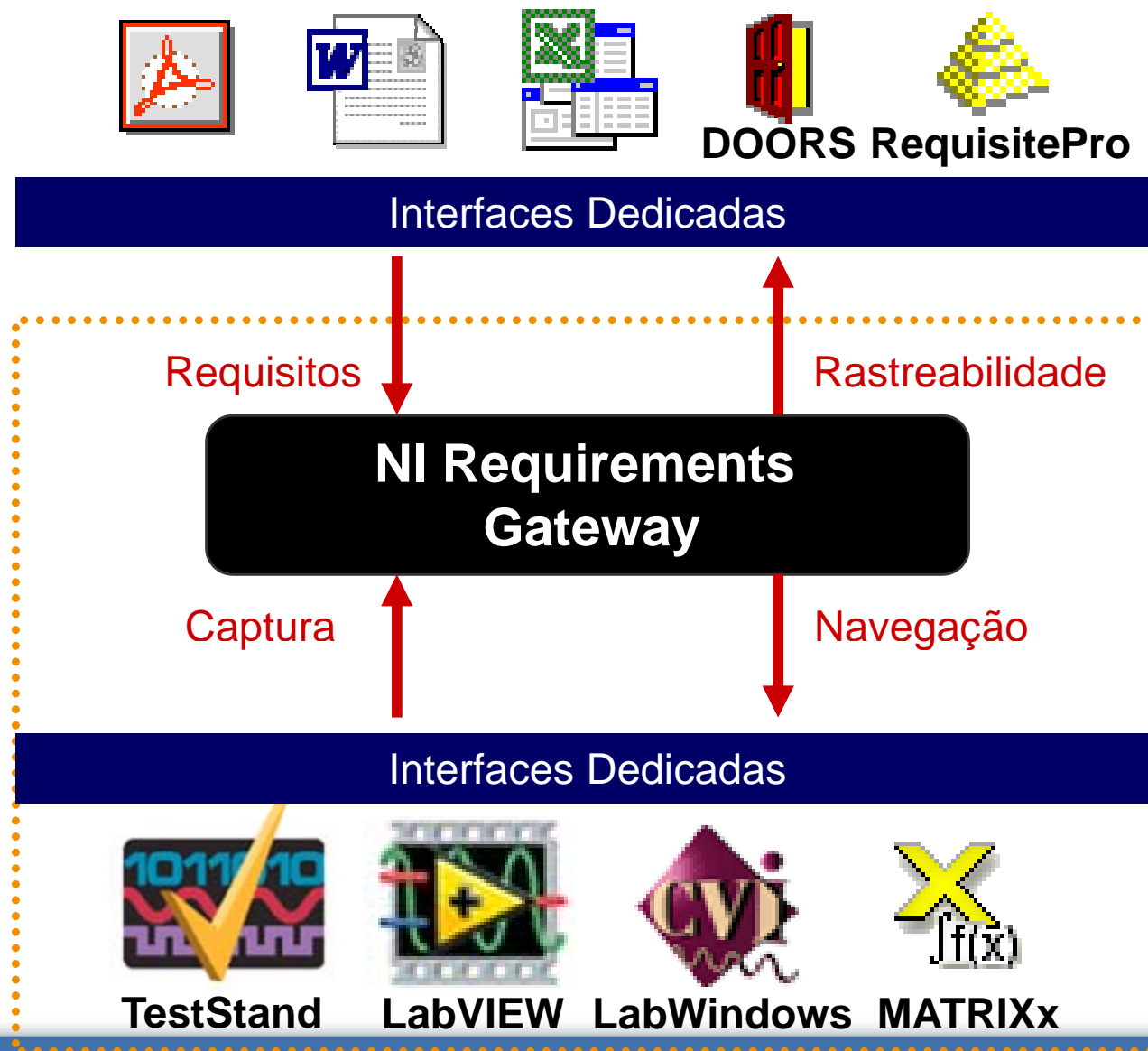
- Com o LabVIEW Project é possível visualizar numa mesma hierarquia o código-fonte e também os requisitos do sistema;
- Utilizando a ferramenta de Build do LabVIEW é possível criar versões para distribuição do projeto que incluem toda a documentação.



Análise da Cobertura de Requisitos



Solução de Rastreabilidade de Requisitos



NI Requirements Gateway

- Conectar os requisitos às aplicações de teste ou controle;
- Especificar relação de cobertura entre documentos;
- Capturar requisitos e informações de rastreabilidade;
- Utilizar métodos gráficos para visualizar análise de cobertura, de impacto e seus inter-relacionamentos;
- Detectar e destacar alterações em requisitos e cobertura;
- Gerar relatórios de rastreabilidade e análise de impactos.

Integração entre o LabVIEW e Documentos de Requisitos utilizando Requirements Gateway

DEMO

Gerenciamento de Configuração

"...é o desenvolvimento e uso de padrões e procedimentos para gerenciamento de sistemas e software em desenvolvimento...

As diferentes versões incorporam propostas, correções de defeitos e adaptações para diferentes HW e Sistemas Operacionais... " (Sommerville)

"A definição e uso de padrões de gerenciamento de configuração são essenciais para a certificação da qualidade, tanto para o padrão ISO 9000, quanto para os padrões CMM e CMMI." (Paulk et al., 1995; Ahern et al., 2002; Peach 1996)

Benefícios de Gerenciar Configurações

- Facilidade para desenvolver sistemas multiplataformas;
- Pode ser utilizado com parte do sistema de gerenciamento de qualidade do software;
- Utiliza padrões definidos IEEE 828;
(IEEE Standard for Software Configuration Management Plans)
- Rastrear diferenças entre versões para que as novas versões sejam derivadas de maneira controlada (controle de versões).

LabVIEW

Programação Gráfica

Linux



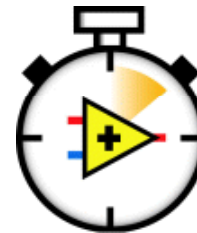
Macintosh



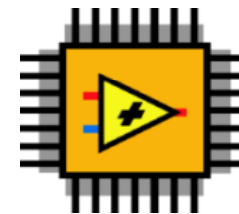
Mobile
XP Embedded
Windows CE

Plataformas Desktop

Real-Time



FPGA



MPU



Plataformas Embarcadas

Uma linguagem que permita ao programador, desenvolver projetos em múltiplas plataformas, sem ter que estudar uma linguagem nova a cada uma delas, pode ser a diferença entre atender ou não os prazos e custos do projeto.

Uso de diretivas de compilação

DEMO

Arquiteturas

“Sistemas grandes são sempre decompostos em sub-sistemas que fornecem algum conjunto de serviços relacionados. O processo inicial de projeto, que consiste em identificar esses sub-sistemas e estabelecer um *framework* para o controle e a comunicação de sub-sistemas, é denominado projeto de arquiteturas...” (Sommerville)

“Quantas vezes você teve um *déjà-vu* de arquitetura – aquele sentimento de que você já havia solucionado aquele problema anteriormente sem saber onde nem como?” (Gamma et. al)

Benefícios da utilização de arquiteturas

- É uma apresentação em alto-nível do sistema proposto;
- Torna a arquitetura do sistema explícita em um estágio inicial do desenvolvimento;
 - Desempenho, confiabilidade e facilidade de manutenção
- Reuso em larga escala, tanto da arquitetura, quanto dos componentes.

Benefícios da utilização de arquiteturas

Simplifica o processo de desenvolvimento

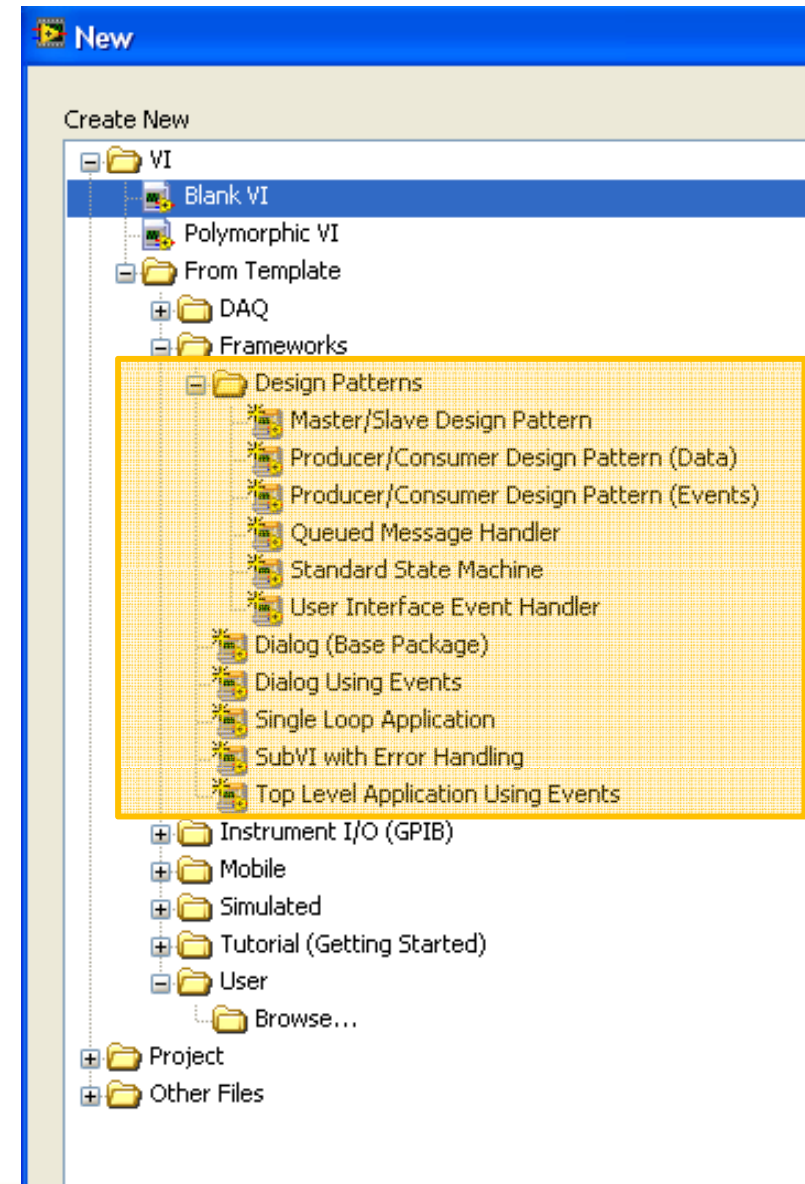
- Facilita a interpretação do código;
- Não há necessidade de “reinventar a roda”;
- Utilizar soluções pré-existentes para solucionar problemas comuns.

Confiabilidade

- A maioria já é utilizada há anos – elas são “testadas e aprovadas”;
- Consultar uma enorme comunidade de desenvolvedores e recursos ilimitados *online*.

Alguns tipos de arquiteturas comuns em LabVIEW

- Fluxo de Dados (inerente ao LabVIEW);
- Máquina de Estados;
- Interface de Usuário Baseada em Eventos;
- Mensagens enfileiradas (*script's*);
- *Pipeline*;
- Cliente e Servidor;
- Mestre e Escravo (síncrono);
- Produtor Consumidor de Dados e de Eventos (assíncrono);
- Orientação à Objetos.



Como escolher a arquitetura ideal?

- Identifique os aspectos mais importantes da sua aplicação:
 - Processos que precisam rodar em paralelo (desacoplados);
 - Desempenho;
 - Componentes de missão-crítica (tempo-real);
 - Segurança e Disponibilidade.
- Selecione uma arquitetura que permita o crescimento da sua aplicação.

Cuidado!

Você pode complicar tremendamente a sua vida caso escolha uma arquitetura desnecessariamente complexa demais.



Nunca da mais comum das arquiteturas – **Fluxo de Dados!**



National Instruments
Customer Education

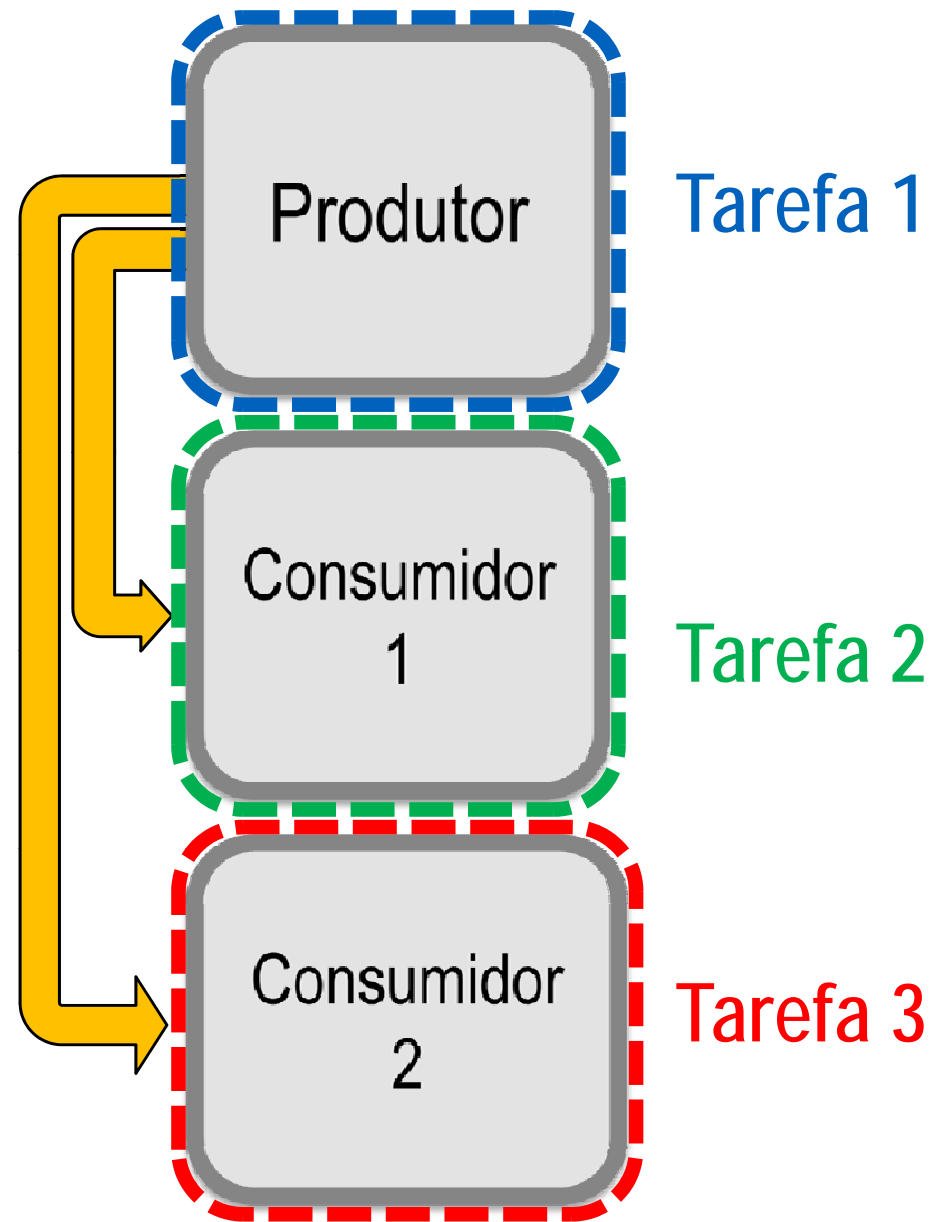
LabVIEW
Intermediate I

Produtor/Consumidor

Eu tenho dois processos que precisam ser executados simultaneamente, porém um não pode afetar o desempenho do outro.

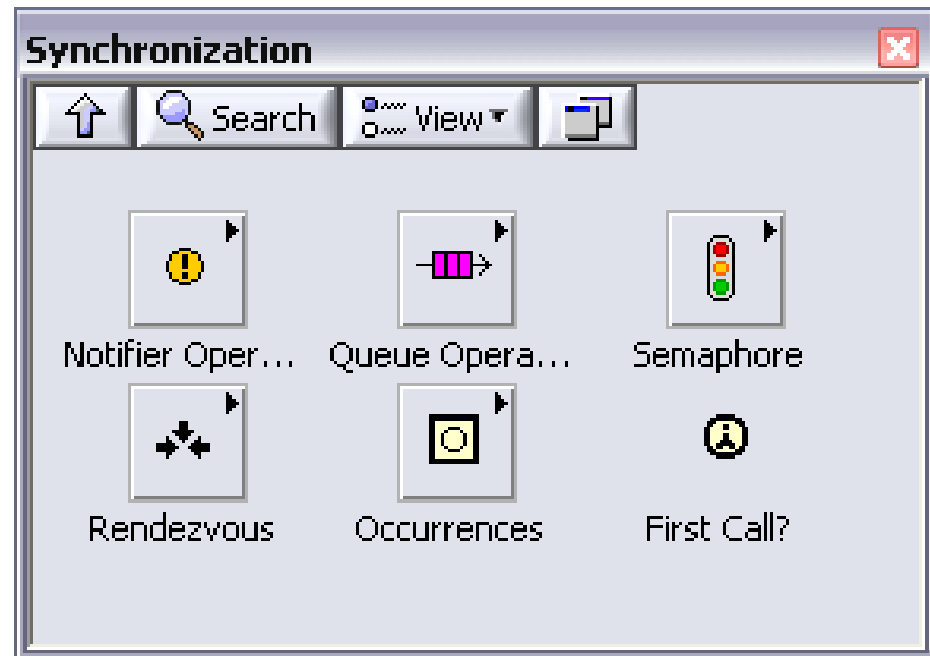
Como funciona

- *Loop* Produtor avisa um ou mais Consumidores quando ele deve rodar;
- Permite execução assíncrona dos loops;
- Independência de dados quebra a execução por fluxo de dados e permite execução multitarefa;
- Desacopla os processos.

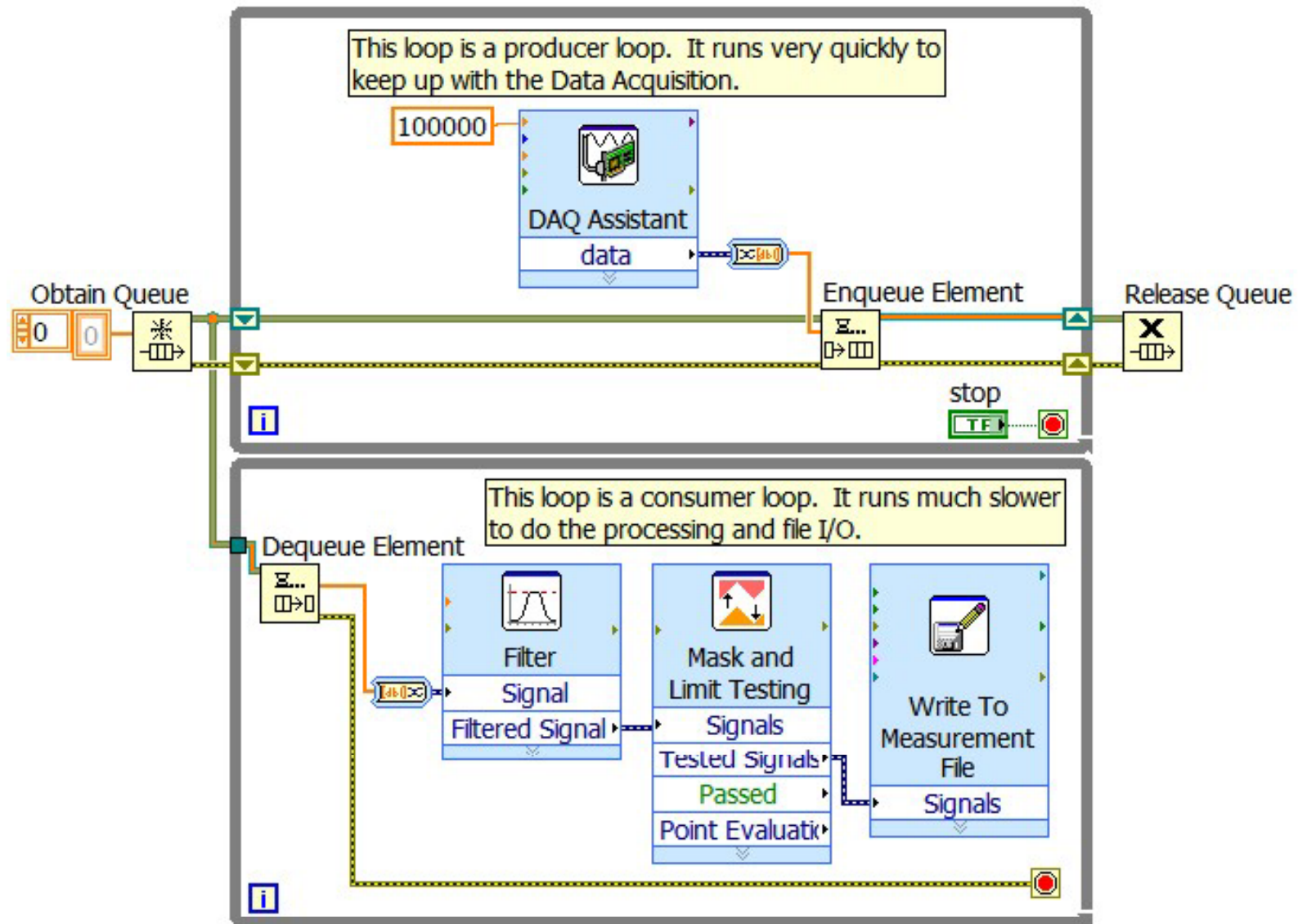


Comunicação entre *Loop* 's

- Variáveis
- Ocorrência
- Notificador
- Fila
- Semáforo
- Encontro



Produtor/Consumidor



Produtor/Consumidor

DEMO

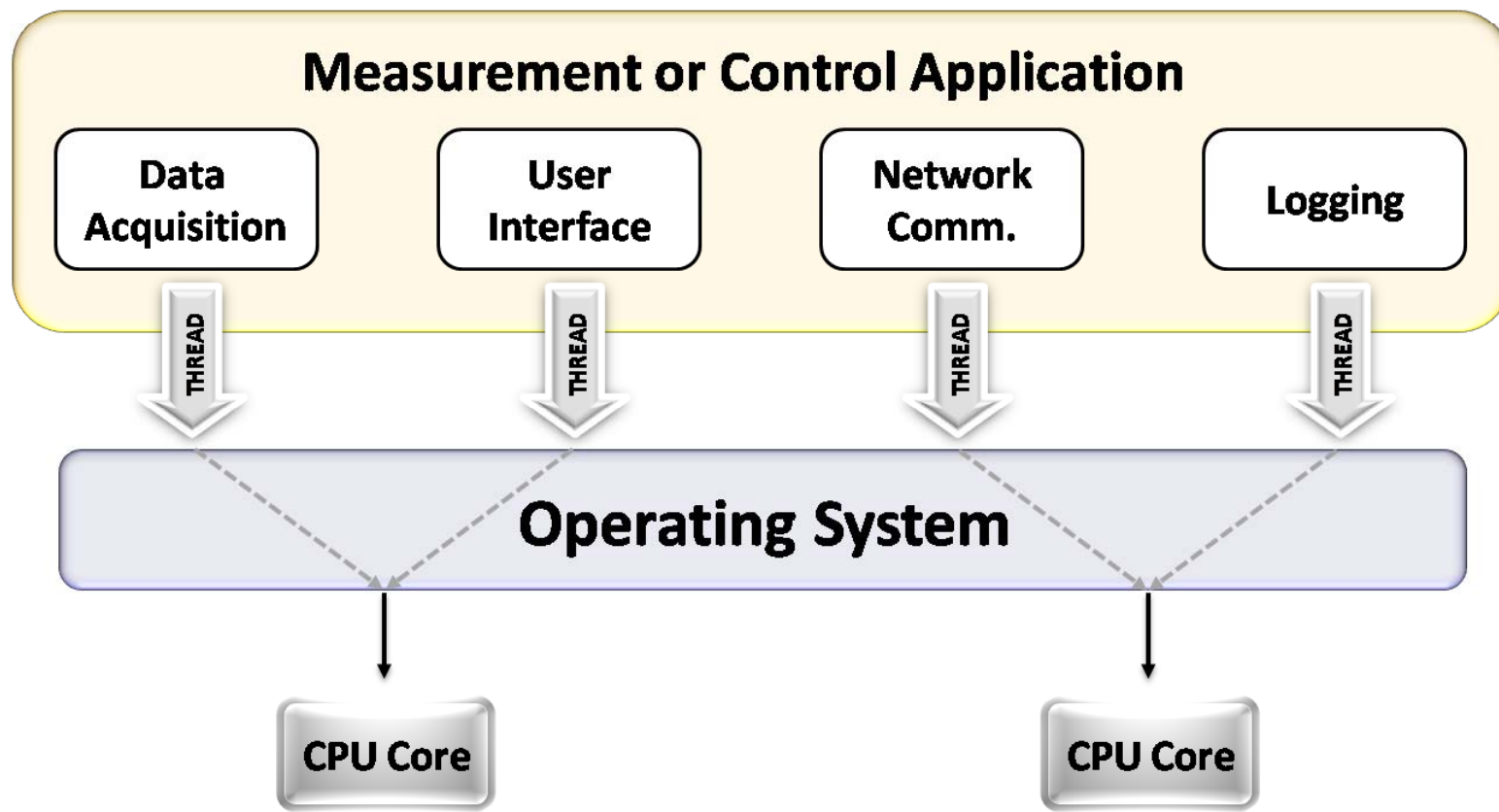
Multitarefa

Eu tenho uma aplicação que mesmo sendo executada no melhor computador ainda é lenta.

(The free lunch is over -

Como Funciona

Loop's que não tem dependência de dados são automaticamente distribuídos em múltiplas tarefas no LabVIEW (desde 1998)



Mutitarefa

DEMO



National Instruments
Customer Education

LabVIEW OOP
System Design

Programação Orientada à Objetos

Assista a apresentação:

Engenharia de software para desenvolvimento com
LabVIEW: Orientação a Objetos, Statechart e Validação

Auditório Vermelho - 15:20hs

Utilizando Arquiteturas

Problema: Criar um interface de usuário interativa

Precisamos criar uma aplicação na qual a interface de usuário detecta as entradas (eventos) e responde de acordo com cada entrada recebida. Esta interface não deve utilizar muitos recursos da CPU. As ações que serão desempenhadas são independentes uma das outras.

Solução: Interface de Usuário Baseada em Eventos

Devemos usar a arquitetura Interface de Usuário Baseada em Eventos (event-based design pattern) porque desejamos limitar a utilização da CPU enquanto aguardamos por um evento. Não devemos encontrar condições concorrentes porque as ações são independentes umas das outras.

Utilizando Arquiteturas

Problema: Sistema de Teste e calibração

Precisamos testar diferentes dispositivos em uma linha de produção. Baseado nos resultados de teste, podemos calibrar o sistema utilizando uma ou duas rotinas e então repetir o teste.

Solução: Máquina de Estados

Porque não sabemos qual rotina de calibração precisamos usar, utilizamos a máquina de estados para selecionar dinamicamente para qual estado a máquina deve ir.

Nota: Não devemos a arquitetura de Programação Orientada a Objetos (object-oriented programming factory design pattern) uma vez que só temos duas rotinas de calibração. Programação Orientada a Objetos seria desnecessariamente complexa.

Utilizando Arquiteturas

Problema: Aquisição de dados e gravação

Precisamos adquirir dados de dois instrumentos externos utilizando a mesma taxa de aquisição, filtrar os dados, adicionar data/hora do teste e o nome do operador que realizou o teste e então gravar os dados em um arquivo.

Solução: Produtor/Consumidor

Devemos utilizar a arquitetura produtor/consumidor porque temos múltiplas tarefas que executam em diferentes velocidades e não podem ser afetadas pelas tarefas mais lentas. Cada Instrumento externo será um loop produtor as tarefas de processamento e gravação serão loops consumidores.

Utilizando Arquiteturas

Problema: Renderização dinâmica de um grupo de objetos 3D

Precisamos criar uma série de objetos 3D e apresentá-los. Esses objetos são diferentes uns dos outros, porém compartilharão algumas propriedades. O número de objetos que precisa ser criado de cada tipo só será conhecido quando o programa for executado.

Solução: Programação Orientada a Objectos

Devemos usar a POO como uma fábrica que produz um determinado número de cada objeto 3D. Porque não sabemos quantos objetos serão produzidos de antemão e terão propriedades similares, criar esses objetos dinamicamente de uma fábrica POO (object-oriented programming factory design pattern) será a solução mais eficiente.

Resumo

- Métodos de desenvolvimento de software são essenciais para o desenvolvimento de aplicações complexas;
- O estudo e a utilização destes métodos pode definir o seu fracasso ou sucesso, decida-se!

Referências

- Engenharia de Software – Sommerville
- A Software Engineering Approach to LabVIEW – Jon Conway e Steve Watts
- Treinamento LabVIEW Intermediário – National Instruments
- The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software - By Herb Sutter

Obrigado!

Não esqueça de preencher a avaliação.

Para mais informações acesse ni.com ou
ligue para (11) 3149-3149

