ENGENHARIA DE *SOFTWARE* E SISTEMAS DE MISSÃO CRÍTICA

Aluno: Thiago Pinheiro de Araújo
Orientador: Arndt von Staa

Introdução

A Petrobrás utiliza robôs para inspeção da integridade de dutos e, dentre eles, o robô PIG, que coleta dados a partir de sensores odométricos e ultrassônicos. Para aquisição dos dados coletados é utilizado um software chamado RTScan2. Este software possuía diversos *bugs* em sua ultima versão, que impediam uma inspeção eficiente e confiável dos dutos.

O *software* divide-se em duas partes: API, responsável pela aquisição e armazenamento dos dados, e uma segunda parte que gera a visualização dos dados. Para a criação da API foram utilizadas técnicas de engenharia de *software*, e esta funciona perfeitamente. Porém a segunda parte, de visualização, foi criada sem ter sido feita uma modelagem do projeto e apresentava *bugs* à medida que novas funcionalidades eram criadas.

Objetivos

Realizar uma re-engenharia deste *software* aplicando técnicas de engenharia de *software* e de desenvolvimento de sistemas de missão crítica para tornar o software confiável e tolerante a falhas. Com isso, uma nova versão será criada sem novas funcionalidades, porém com uma arquitetura que minimize as chances de falha ou consiga tratar a mesma. Esta arquitetura deve ser criada de forma que novas funcionalidades sejam implementadas de forma simples e modularizada.

Metodologia

A metodologia utilizada foi quebrar estes dois grandes módulos em pequenos módulos com funções específicas e com um encapsulamento perfeito. Foram utilizados padrões de projeto para criar esta nova arquitetura do *software*. O desenvolvimento modularizado tornou simples a depuração e correção de eventuais *bugs*.

O módulo principal implementa o padrão de projeto *Mediador* e tem como função principal conectar todos os componentes do software com a interface de controle. Estes componentes seguem o padrão de projeto *Observador*, sendo reconstruídos à medida que seu objeto observado, o módulo principal, modifica seu estado.

Cada componente criado é uma forma de visualização distinta dos dados. Como a pintura de cada componente é complexa foi aplicado o padrão de projeto *Decorador*, que consiste em criar diversos módulos chamados de *Decoradores*, delegando a cada um deles gerar a decoração adequada ao gráfico criado pelo seu componente pai. Um dos pontos mais importantes da utilização deste padrão de projeto foi o reuso de código, onde diversos decoradores foram adicionados à componentes distintos, provendo ao seu dono uma funcionalidade que não foi projetada especificamente para ele.

Foi utilizado também um sistema de notificação de eventos, fornecido pela biblioteca QT de C++. Com isto foi possível criar uma interface de controle desligada do módulo principal, onde cada evento pode ser tratado por mais de um componente, sem a necessidade do componente conhecer quem gerou o evento ou de qualquer outro tratador de eventos.

Conclusões

A nova arquitetura mostrou-se eficiente, não apresentando *bugs*. Todos os encontrados durante o processo de desenvolvimento foram rapidamente corrigidos devido a nova arquitetura do software.

O software antigo possuia dois grandes módulos: TRTScan, responsável por construir a interface de controle do usuário e TChapacorteView, responsável pela pintura efetiva dos dados. Eles possuiam 6 mil e 8 mil linhas respectivamente. Na nova arquitetura, estes dois módulos foram substituidos por aproximadamente 30 módulos. Apesar do número de módulos ter aumentado a depuração tornou-se mais fácil e, ao detectar algum *bug*, sua remoção é rápida tendo em vista que o ambiente de busca é em geral pequeno.

Em termos práticos, o produto final é um software fiel e robusto, capaz de fazer inspeções confiáveis. A evolução desta tecnologia de inspeção permite agora um aumento seguro das funcionalidades deste *software*.

Referências

1 – GAMMA, Erich. **Design Patterns: Elements of Reusable Object-Oriented Software.** 1.ed. Addison-Wesley Professional Computing Series, 1995

2 – VLISSIDES, John M. **Pattern Hatching : Design Patterns Applied** 1.ed. Addison-Wesley Professional Computing Series, 1998