Construção de uma metodologia para desenvolvimento de software embarcado

Rogério Dourado,Elthon Allex
{rogerio,elthon}@dsc.ufcg.edu.br

26 de Agosto de 2004

1 Introdução

Uma grande quantidade de sistemas e equipamentos inteligentes tem presença cada vez maior na nossa vida diária: telefones celulares, automóveis, microondas, PDAs, aviões, terminais de atendimento bancário e muitos outros. O fato comum é que todos esses equipamentos possuem sistemas embarcados (*embedded systems*).

Duas características preponderantes são identificadas na definição destes sistemas. A primeira delas é que estes sistemas utilizam tecnologia computacional para atender à aplicações específicas, e portanto são projetados para atender aos requisitos de tais aplicações. Não são, portanto, computadores de uso geral, tais como os PCs que utilizamos em casa ou no escritório.

A segunda característica marcante é que essencialmente sistemas embarcados têm comportamento reativo - definido por sua interação com o ambiente - e restrições de tempo, o que os caracterizam como sistemas de tempo real. É exigido desses sistemas que respondam no tempo certo aos eventos gerados pelo meio ambiente. Contudo, a ordem de ocorrência desses eventos no tempo não é sempre previsível.

O crescimento do uso de sistemas embarcados tem exigido das companhias do setor a redução do tempo necessário para desenvolver tais sistema, ao mesmo tempo que os produtos tenham cada vez mais qualidade. O fator qualidade deve-se principalmente ao fato de que sistemas embarcados de tempo real são usados nos sistemas mais críticos do mundo, como por exemplo nas áreas médicas, de telecomunicações e aviação.

No entanto o projeto deste tipo de sistema computacional é extremamente complexo, por envolver conceitos até então pouco analisados pela computação de proprósitos gerais. Por exemplo, as questões de portabilidade e do limite de consumo de potência sem perda de desempenho, a baixa disponibilidade de memória, a necessidade de segurança e confiabilidade e o curto tempo de projeto tornam o desenvolvimento de sistemas computacionais embarcados uma área de estudo em si.

Por essas razões faz-se necessário o emprego de uma abordagem de desenvolvimento de software que leve em consideração as características e peculiaridades próprias deste contexto.

1.1 Definição de Sistemas Embarcados

Um sistema embarcado é um sistema de computação especializado, que faz parte de uma máquina ou sistema mais amplo. Geralmente, é um sistema microprocessado com os programas armazenados em ROM (*Read Only Memory*).

Alguns sistemas embarcados podem incluir um sistema operacional, mas muitos são tão especializados que tanto as tarefas a serem executadas quanto as funções especificas de um sistema operacional podem estar implementadas em um único programa.

Atualmente está cada vez mais difícil classificar um sistema como sendo ou não um sistema embarcado, em função dos recentes requisitos de funcionalidade que os mesmos vêm apresentando. Por exemplo, alguns autores [CW03] colocam que um sistema embarcado é designado para um propósito especifico, não podendo ser programado pelo usuário da mesma forma que um computador pessoal e que o usuário pode fazer escolhas referentes à funcionalidade, mas não pode mudar esta funcionalidade pela adição ou substituição de software. Esta afirmação é valida quando aplicada a sistemas embarcados tradicionais onde pouca ou nenhuma alteração em termos de funcionalidade é feita no sistema depois de o mesmo ter sido concluído, mas contrasta fortemente com os novos dispositivos embarcados de uso pessoal ou de consumo cuja funcionalidade pode ser modificada dinamicamente. De acordo com a definição acima, um PALM não poderia ser considerado um dispositivo embarcado porque possibilita execução de aplicações diversas da mesma forma que um computador pessoal. De fato, tais dispositivos deveriam se chamados apenas de computadores de bolso.

Além disso, como já foi dito anteriormente sistemas embarcados geralmente devem ser limitados quanto aos recursos computacionais exigidos para sua execução, uma vez que os equipamentos nos quais tais sistemas serão utilizados devem primar pela otimização e simplicidade. Devido a este fato é que pode-se afirmar que todo sistema embarcado é um software que deve ser desenvolvido em uma plataforma diferente da plataforma a qual este deve ser executado.

1.2 Evolução do mercado

O mercado de sistemas embarcados tem crescido numa taxa extremamente alta não só em volume de produção, mas também em diversidade de aplicações. Esta demanda crescente de mercado implica na necessidade de novas ferramentas e metodologias para um suporte efetivo no projeto de tais sistemas. Adicionalmente, os produtos deste mercado possuem um tempo de vida relativamente curto em relação à outras aplicações. Esta peculiaridade exige que o *"time-to-market"* seja o menor possível para que o produto possa ser competitivo no mercado. A redução do *"time-to-market"* é um fator extremamente crítico no projeto de sistemas embarcados.

2 Requisitos de projeto

Os sistemas embarcados, normalmente, apresentam altas restrições [dS01] em termos de funcionalidade e implementação. Em particular, eles devem reagir a eventos externos em tempo real, adaptar-se a limites de tamanho e peso, gerenciar consumo de potência, satisfazer requisitos de confiabilidade e segurança e adequar-se a restrições de orçamento.

2.1 Resposta em Tempo Real

Um sistema é dito de Tempo Real quando ele responde a entradas e fornece saídas, rápido o suficiente para atender aos requisitos do hardware ou do usuário. Sistemas embarcados são essencialmente sistemas de tempo real.

2.2 Resposta a eventos síncronos/assíncronos

Sistemas embarcados necessitam responder a eventos internos ou externos. Estes eventos podem ser síncronos, caso em que uma política de escalonamento de eventos para garantir performance pode ser aplicável, ou assíncronos, caso em que uma estimativa do comportamento do mesmo deve ser feita para garantir uma performance mínima no pior caso.

2.3 Tamanho e custo reduzidos

Sistemas embarcados geralmente apresentam restrições como tamanho e peso ou custo unitário. Estas restrições são importantes para a definição da arquitetura de um sistema embarcado. O tamanho e/ou peso são restritivos quanto à escolha dos componentes físicos que podem fazer parte do sistema. Da mesma forma, as restrições de custo podem fazer com que sejam escolhidos para compor o sistema componentes de hardware/software que não são os mais modernos no mercado.

2.4 Segurança e confiabilidade

Alguns sistemas embarcados podem trazer risco aos usuários em caso de falha. É o caso, por exemplo, do controle automático de vôo em aeronaves ou sistema automático de freios em carros.

Tradicionalmente, utiliza-se alguma forma de redundância para tornar um sistema tolerante a falhas, seja ela de componentes de software ou hardware, informações ou tempo. E no caso dos sistemas embarcados, dadas as suas restrições não só em termos de custo e desempenho, mas também em relação a volume, peso e consumo de energia, a aplicação de técnicas de tolerância a falhas deve ser criteriosa.

2.5 Ambientes inóspitos

Muitos sistemas embarcados operam em ambientes inóspitos ou mesmo inacessíveis, demandando a necessidade de proteção contra choques, vibrações, flutuações na fonte de energia, calor excessivo, fogo, água, corrosão, etc. Em tais ambientes a possibilidade de falha causada por um sinistro é elevada, o que demanda um bom mecanismo de tolerância a falhas.

3 Situação Atual

Até recentemente, os projetos de sistemas embarcados davam uma ênfase maior ao *design* do *hardware* do que ao *software*. Normalmente a parte da solução em software desses produtos era bem reduzida, e serviam para definir como o *hardware* deveria operar. Nesse período, os sistemas embarcados contavam com processadores de recursos extremamente limitados, nos quais a linguagem *assembly* era geralmente usada para programar um conjunto de funções imutáveis; eram considerados simples programas que forneciam algum tipo de inteligência aos mecanismos mecânicos.

Atualmente, os sistemas embarcados tipicamente contam com processadores bem mais potentes, que são programados principalmente através de linguagens de alto nível como C e C++. Em muitos casos tais sistemas não somente fazem os equipamentos funcionarem mas também interagem com outros sistemas. Isto exige que sistemas embarcados coletem, processem (parcialmente) e transmitam dados. Em média, o *software* para este tipo de equipamento corresponde ente 80% e 90% das funcionalidades como também dos custos de desenvolvimento em sistemas embarcados.

Enquanto o *hardware* dos sistemas embarcados evolui, a demanda por software cresce também. Isto torna a escolha das técnicas de engenharia, da linguagem de programação e da plataforma de desenvolvimento para sistemas embarcados muito significante. Muitos sistemas embarcados possuem um sistema operacional que formam o núcleo (e algumas vezes o todo) de seu *software*. Diferentemente da computação de proprósito geral, o mercado de sistemas embarcados suportam muitos diferentes sistemas operacionais.

Um esforço considerável tem sido feito para melhorar tanto o processo quanto a gerência de projetos para sistemas embarcados nos últimos 10 anos. No entanto o desenvolvimento destes sistemas têm evoluído de uma maneira isolada dos avanços obtidos no desenvolvimento de aplicações tradicionais. Enquanto o desenvolvimento informal e *ad-hoc* podem ter conseguido produzir soluções razoáveis há duas décadas atrás, quando os sistemas não excediam mil linhas de código, os sistemas embarcados atuais freqüentemente excedem um milhão de linhas de código, e essa abordagem não é mais eficiente e nem confiável.

Para se ter uma idéia, o mercado norte-americano de sistemas embarcados saltou de um volume de $32 bilhões de dólares negociados em 1998 para uma previsão de $66 bilhões em 2004. Durante esse período os sistemas embarcados deixaram de ser empregados quase que exclusivamente na automação industrial, para se fazerem necessários em uma grande variedade de mercados. Essa expansão é um dos principais fatores da necessidade por sistemas mais flexíveis e funcionais. Reduzir os custos e o tempo de desenvolvimento, nunca foram tão cruciais, e nesse cenário a reusabilidade tem um papel fundamental.

3.1 Desafios

Estudos da empresa VDC (Venture Development Corporation) com mais de 450 desenvolvedores de sistemas embarcados, revelaram alguns dos fatores que estão contribuindo para o aumento do número de projetos atrasados ou cancelados nessa área. O resultado são os altos custos de desenvolvimento e suporte e muitas oportunidades de mercado desperdiçadas.

Uma comparação entre esses projetos mostraram também que as indústrias que desenvolvem aplicações maiores e mais complexas têm uma porcentagem maior de projetos que finalizaram dentro do cronograma. Os fabricantes estão procurando incluir mais funcionalidades do que nunca em seus produtos em intervalos de tempo cada vez menores de desenvolvimento.

Ordem	Fatores
1	Mudanças na especificação
2	Especificação incompleta
3	Complexidade da aplicação
4	Quantidade insuficiente de desenvolvedores

Tabela 1: Fatores que mais contribuem para o fracasso de projetos

Em adição à crescente complexidade, os projetos de sistemas embarcados continuam sendo frustrados por especificações inadequadas bem como por mudanças dessas especificações durante o desenvolvimento do produto. O uso de ferramentas podem ajudar a automatizar o processo de *design* e especificação ao mesmo tempo em que fornecem algum tipo de alívio para os desenvolvedores, no entanto, é esperado que os fabricantes precisarão olhar além da disponibilidade de ferramentas para começarem a examinar a gerência de desenvolvimento dos seus projetos.

4 Metodologias Existentes

4.1 Extreme Embedded

A metodologia *Extreme Programming* [Xis], talvez seja uma das metodologias ágeis mais divulgadas e efetivamente usadas na indústria de software atualmente. Esse relativo sucesso, levou a comunidade de desenvolvedores de sistemas embarcados a se perguntarem se o XP seria aplicável também em seus projetos, levando-se em consideração todos os requisitos inerentes ao desenvolvimento de tais sistemas. Essa é uma questão ainda muito polêmica, a qual discutiremos mais profundamente a seguir. Porém antes disso, abaixo são listados algumas das principais práticas do XP, para que a partir delas possamos discutir o assunto.

- O software é entregue em **pequenas versões** para que o cliente possa obter o seu ganho o mais cedo possível e para minimizar riscos;
- **Primeiro são escritos os testes**, depois é feita a implementação e por último trabalha-se o design;
- **Integração contínua**, no qual os diversos módulos do software são integrados diversas vezes por dia e todos os testes unitários são executados;
- **Design simples**, para que o código esteja, a qualquer momento, na forma mais simples que passe em todos os testes;
- Sempre **refatorar** o código, para que a cada nova funcionalidade adicionada, seu design permaneça na sua forma mais simples;
- Todo código que vai para produção deve ser feito por **pares de programadores**.

Como em qualquer outra metodologia ágil o XP fomenta o foco nos indivíduos e suas interações do que nos processos e ferramentas. Defende também que software em funcionamento "vale"mais do que uma documentação abrangente e principalmente que responder a mudanças é mais importante do que seguir planos. Apesar de em um primeiro momento acharmos que esses princípios e práticas contrastam com os requisitos que levantamos até aqui para sistemas embarcados, existem alguns relatos e experiências de sucesso na adoção do XP em tais projetos, o que alguns vêm chamando de *Extreme Embedded* [Gre02], que nada mais é do que a metodologia XP aplicada nesse contexto.

Segundo alguns trabalhos nessa área [MS02] [MB02], a primeira prática, a programação em par tem oferecido resultados excelentes no desenvolvimento de sistemas embarcados. Isso se deve principalmente ao fato que para programas complexos os benefícios desta prática se tornam ainda mais visíveis. Estudos mostram a diminuição de quase metade dos erros de codificação, e a melhoria da produtividade que se torna melhor do que o de dois programadores solo somados. A idéia embutida também nesta prática é que a melhor forma de você aprender o que um determinado código faz, é ensinando a alguém (nesse caso, o parceiro).

A segunda prática chave de XP, é primeiro codificar os testes, para depois implementar o código propriamente dito. A cada mudança na implementação os testes devem ser executados novamente para aumentar a confiança que nenhum erro foi inserido na modificação. Essa facilidade em executar os testes no momento em que convir ao programador algumas vezes não é possível em um ambiente de desenvolvimento para sistemas embarcados, isto vai depender muito das características do projeto em si. Mas sem dúvida, se possível, esta prática tende a melhorar e muito a qualidade do software como um todo.

Existe sempre uma tendência para se englobar mais funcionalidades, ou até mesmo para implementar certas funcionalidades de uma maneira mais complexa baseado em expectativas futuras. Nesse sentido XP sempre defende que o código deve ser mantido o mais simples possível para que passe nos testes. Essa regra, dentro do contexto de sistemas embarcados, tem ainda mais relevância pelo fato de existir normalmente uma complexidade considerável que é inerente aos requisitos essenciais que devem ser satisfeitos. Indiscutivelmente, aumentar ainda mais essa complexidade contribui e muito para o fracasso do projeto.

O refatoramento do código e a noção de que desenvolvedores cansados são contraproducentes pode, em um primeiro momento, serem vistos como práticas que desperdiçam recursos e diminuem a produção. No entanto as suas aplicações têm mostrado que a produtividade aumenta a partir do momento em que trabalhadores dispostos e atentos acrescentam novas funcionalidades a um código limpo, o que aumenta as chances de manter este código assim.

Vimos que algumas das práticas do XP, trazem para o desenvolvimento de sistemas embarcados praticamente as mesmas vantagens que quando aplicadas no contexto dos sistemas tradicionais. No entanto alguns alegam que XP como um todo é de certa forma incompatível com o desenvolvimento de sistemas embarcados, porque é necessário fazer mais design nas primeiras fases do processo do que XP pode tolerar. Isto se deve principalmente ao *hardware*, que deve ter ao máximo seus requisitos previstos e planejados o quanto antes, pois muito mais que no *software*, mudanças no projeto geram altíssimos custos. Por isso nesse ponto, os relatos sobre a aplicação de XP nesses ambientes, diferenciam um pouco a abordagem para o *hardware* da porção da solução em *software*.

4.2 XP em sistemas críticos

Desenvolver sistemas críticos, onde qualquer falha pode resultar em perdas de vidas humanas exige uma metodologia densa e rigorosa. Não é bem isso que defende Ron Morsicato [MP02], um desenvolvedor veterano que há décadas desenvolve software para controle de dispositivos. Recentemente, Ron começou a usar XP no desenvolvimento de um instrumento farmacêutico, e acabou concluindo que XP é compatível para um ambiente altamente regulado e crítico. Um auditor externo concluiu que a equipe havia implementado práticas suficientemente boas para as características exigidas pelo projeto, no entanto o processo não conseguiu passar na auditoria pelo simples fato de ter se dado a liberdade a equipe de implementar o XP unilateralmente.

Ron considera que existem dois pontos chaves relacionados aos sistemas críticos. Primeiramente, é necessário conhecer todas as possíveis situações nas quais uma condição de risco pode ocorrer. E a maneira de descobri-las é através de reuniões com pessoas que conhecem o domínio da aplicação, para um exercício de imaginação de cenários que poderiam acarretar em falhas de segurança. Uma vez que esses cenários são identificados, é relativamente fácil construir controles no sistema que devem evitá-los. O difícil, segundo ele é pensar sobre tudo que poderia dar errado em um primeiro momento. Nesse tipo de software dificilmente problemas que foram antecipados ocorrem durante sua operação. Contudo o aspecto mais importante, é

ter certeza que todas as possibilidades operacionais foram previamente consideradas.

O segundo ponto chave é que uma vez que cenários de perigos foram identificados e controles foram projetados para impedir suas ocorrências, é preciso se certificar que que modificações futuras do código mantenham esses controles intactos. Há um conflito inerente a esses dois objetivos, e a melhor maneira de identificar todos os possíveis cenários de perigo é refatorar continuamente o design de forma a sempre estar reavaliando as questões de segurança.

Ron afirma que até recentemente haviam duas abordagens para evitar o esquecimento de cenários de falha. A primeira é a abordagem *ad hoc*, que tende a identificar mais fatores de risco, mas não garante que eles serão levados em conta durante todo o ciclo de vida do produto. A outra abordagem é "paralisar" o design e rastrear todo o código de volta aos requisitos originais. Essa última garante que fatores de risco terão sempre seus devidos controles implementados, mas não é boa para encontrar esses fatores. Teoricamente, uma boa metodologia deveria englobar ambas característica. Nesse contexto, XP seria a terceira opção, que segundo o autor é muito melhor em encontrar os fatores de risco citados acima, como também em proteger controles existentes.

A rastreabilidade dos requisitos talvez fizesse com que o processo obtivesse êxito na auditoria. Mas o fato é que em XP toda implementação de funcionalidades deve ser guiada por *user stories*, e esta nova implementação deve estar sempre sendo integrada e testada juntamente com o resto da aplicação, para assegurar que erros não sejam inseridos ao longo do desenvolvimento. Com isso o que Ron defende é que a forma de se garantir a cobertura dos requisitos por exemplo, bem como a qualidade dos sistemas desenvolvidos sob o XP, não é necessariamente através da rastreabilidade e sim através de suas próprias práticas.

Mesmo para sistemas embarcados os requisitos sempre mudam ao longo do desenvolvimento. E além disso é perigoso imaginar que todas as questões de segurança serão levantadas durante o design inicial da solução. Por mais criteriosa que seja essa etapa ainda é improvável que isso ocorra. São por esses dois fatores que Ron defende que a prática do refatoramento melhora a qualidade dos controles de segurança porque as oportunidades de simplificar seu design contribuem para a descoberta de novos cenários de falha. Tenta-se alguma coisa, verifica como essa coisa funciona e a melhora.

Desenvolvedores de sistemas embarcados irão produzir códigos de alta qualidade se els puderem entender claramente o que significa qualidades para seus usuários, se eles puderem constantemente testar o código face a esse entendimento, e se eles puderem regularmente refatora-lo. Com uma boa gerência, eles se sentirão como membros de um time de segurança, imersos em uma cultura de segurança.

4.3 Metodologia DESS

Define uma metodologia de desenvolvimento de software orientado a objeto baseado em componentes para sistemas embutidos de tempo real, cria ferramentas de suporte e prova a decência da metodologia através de vários casos de teste de validação.

Em geral, a meta do *DESS* [Int] é prover um instrumento que traga os benefícios das estratégias modernas de OO e de desenvolvimento baseado em componentes (DBC) [Bro00] para o mundo embutido, onde há restrições de tempo e memória.

A metodologia *DESS* tem sido desenvolvida levando em consideração a existência de muitas linguagens de modelagem e metodologias (UML, UP, V-model), herdando suas semânticas. Ela usa um conceito mais prático de *workflow* do que as demais. *Workflow* é considerado um conjunto de atividades relacionadas que são agrupadas por natureza e produzem uma coleção de artefatos relacionados.

Nesta metodologia, os *workflows* são agrupados em três *Workflow V's*. Um *Workflow V* é um conjunto de *workflows* que podem estar logicamente conecta-

Figura 1: Representação gráfica do DESS

dos por fluxos de artefatos e podem ser graficamente representados em forma de 'V', como pode ser visto na Figura 1. O modelo de desenvolvimento consiste de três desenvolvimentos V's que ocorrem em paralelo: de realização, de validação e verificação, e de gerenciamento de requisitos. Cada um destes modelos representa o fluxo de informação que ocorre entre os artefatos de cada um. Cada *workflow* produz uma saída que serve como dado de entrada para produção de um artefato seguinte dentro do mesmo *Workflow V*.

4.3.1 Etapas do *Workflow V* de realização

Nesta Seção são apresentados os workflows, ou etapas, relacionados diretamente à realização do sistema. É a parte principal do desenvolvimento do sistema, exceto pela ausência da validação e verificação.

1. **Engenharia de requisitos do usuário** captura todas as funcionalidades do sistema esperadas pelo usuário. Requisitos de *stakeholders*, contexto do trabalho, conhecimento sobre o domínio e modelo do domínio são artefatos de entrada para este *workflow*. A partir deles, são construídos documentos contendo os requisitos formais dos usuários e/ou modelos informais de eventos baseados em UML.

2. **Engenharia de requisitos do sistema** captura toda a funcionalidade do sistema. Os *requisitos formais dos usuários* juntamente com alguns padrões e diretrizes são processados e são produzidos como saída os requisitos do sistema.

3. ***Design* do sistema** define uma separação entre as funcionalidades de hardware e de software do sistema e como eles interagem. Nesta etapa, os *requisitos do sistema* e os *modelos informais de eventos baseados em UML* são processados juntamente com, os condutores de projeto e restrições, padrões e diretrizes produzindo-se a arquitetura do sistema e alocação de hardware/software.

4. **Engenharia de requisitos do software** provê um modelo da funcionalidade, bem estruturado, para avaliação do usuário. Os artefatos produzidos pela etapa anterior se juntam aos condutores de projeto e restrições internas e externas, padrões e diretrizes. A partir deles são geradas as especificações de software.

5. A etapa de **análise** provê especificações que permitem desenvolvedores construirem uma boa arquitetura de *design*, definir uma primeira análise de trabalho, alocação de tarefa e planejamento. O processamento, nesta etapa, é feito nas *especificações de software* e nos padrões e diretrizes de companhia, produzindo uma especificação da análise.

6. **Especificação e *design*** provêem a especificação e o *design* da arquitetura do componente. Nesta etapa do *workflow*, as *especificações da análise* e as bibliotecas de esquema dos componentes, padrões e diretrizes de companhia, geram as especificações e o *design* de esquema dos componentes.

7. A **implementação** dos componentes e dos *frameworks* dos componentes que foram identificados ocorre nesta etapa. Os artefatos de entrada são bibliotecas de esquema de componente, *designs* de conectores e componentes,padrões e diretrizes de companhia. São produzidos como artefatos implementações dos componentes e conectores.

8. A **instanciação** de um arcabouço e dos componentes ocorre nesta etapa. Possui como artefatos de entrada: a especificação e o *design* do esquema dos componentes, e bibliotecas de componentes, diretrizes de companhia e padrões.

9. A **integração de software** produz um software consistente a partir dos artefatos de entrada: todas as especificações e *designs* produzidos nas etapas anteriores, bibliotecas de esquema de componente, padrões e diretrizes de companhia.

10. A **integração de sistema** integra o desenvolvimento de hardware e software para um sistema, usando o software embutido no hardware concreto. A partir do código integrado, resultados do desenvolvimento do hardware, requisitos e arquitetura de sistema, alocação de hardware/software, padrões e diretrizes de companhia, é produzido um sistema pronto para ser executado.

11. E por fim, na **organização**, há a transformação do sistema todo num produto final pronto para ser comercializado. Os artefatos de entrada são o sistema executável, padrões e diretrizes de companhia.

4.3.2 Etapas do *Workflow V* de validação e verificação

Nesta Seção são apresentados os *workflows* rederentes a validação e verificação do sistema. Como dito anteriormente, este *Workflow V* é executado em paralelo aos outros dois (de realização e de gerenciamento de requisitos).

1. A **revisão** verifica se o objeto a ser analisado possui ou não todos os seus requisitos. Esta etapa possui como entrada um objeto a ser analisado e seus requisitos. Sua saída é composta dos resultados da revisão.

2. A **verificação de modelos** verifica se o modelo referente ao sistema atende aos seus requisitos ou não. Recebe como artefatos de entrada os requisitos do modelo levantados previamente e o modelo a ser verificado. Produz como artefato de saída os resultados da verificação.

3. Em **teste de componente**, cada componente é testado para verificar se atende ou não aos seus requisitos. A partir dos componentes a serem testados e seus requisitos, são produzidos os resultados dos testes.

4. Em **teste de integração**, são testadas as interfaces entre os componentes, e entre os componentes e o hardware alvo ao seu redor. A partir dos requisitos da interface, de um sistema contendo os componentes a serem testados e do hardware onde serão executados os componentes, são gerados os resultados obtidos a partir dos testes de integração.

5. Na etapa **teste de sistema**, um sistema completo é posto em teste para verificar se o mesmo atende aos seus requisitos. Os artefatos de entrada são o sistema completo e seus requisitos. Daí é que serão obtidos os resultados dos testes.

6. No **teste de aceitação**, o sistema completo é testado para averiguar se o mesmo atende aos seus requisitos. Este tipo de teste recebe como artefatos de entrada os requisitos do usuário e do cliente, e o sistema completo. O artefato de saída é o conjunto de resultados obtidos a partir dos testes realizados.

7. No **teste de regressão**, um sistema previamente testado é submetido a um novo teste para verificar se atende aos seus requisitos. Recebe como artefatos de entrada o sistema e seus requisitos. O artefato de saída é o conjunto de resultados obtidos a partir dos testes de regressão.

4.3.3 Etapas do *Workflow V* de gerenciamento de requisitos

Esta Seção trata dos *workflows* referentes ao gerenciamento de requisitos. Deve haver uma harmonia entre o cliente e a equipe de desenvolvimento, ou seja, ambos devem estar em acordo com metas estabelecidas, *deadlines*, etc.

1. Na **escrita do plano de gerenciamento de requisitos** são definidas as atividades relacionadas ao gerenciamento de requisitos. Os artefatos de entrada para esta etapa são os condutores de projeto e restrições, e repositório de informação. O artefato de saída é o plano de gerenciamento de requisitos.

2. A etapa de ***traceability* de gerência** tem como propósito garantir a implementação apropriada e planejado. O artefato de entrada é o *plano de gerenciamento de requisitos* e o de saída é um relatório.

3. Os **atributos de gerência** tem como propósito ter os atributos definidos para os seus respectivos ítens atualizados. O artefato de entrada é o *plano de gerenciamento de requisitos* e o artefato de saída é um conjunto de matrizes de atributo.

4. Em **mudanças de gerência**, mudanças são realizadas aos requisitos de acordo com o plano de gerenciamento de requisitos. Os artefatos externos de entrada são a requisição de mudança e o relatório de problemas. Os artefatos internos de entrada são o plano de gerenciamento de requisitos e o plano de gerenciamento da configuração. Os artefatos de saída são os estados das requisições de mudança e os relatórios da análise do impacto.

5. E por último, em **relatório de gerência**, é informado à gerência sobre o processo de gerenciamento dos requisitos. Os artefatos de entrada e saída são o plano de gerenciamento de requisitos e os relatórios, e métricas, respectivamente.

5 Experiências no Desenvolvimento de SW Embarcado

Como grupo de desenvolvimento de sistemas embarcados que fazem uso de uma metodologia no processo de desenvolvimento, podemos citar a equipe de desenvolvimento liderada pelo Prof. Dr. Elmar U. K. Melcher, docente da Universidade Federal de Campina Grande, na Paraíba. Na verdade, esta equipe vem trabalhando, juntamente com equipes de outras oito universidades, na criação de uma moderna e bem definida metodologia para o design de IP, usando ferramentas profissionais EDA (*Electronic Design Automation*) e técnicas modernas de especificação, verificação funcional e prototipagem rápida.

Como estudo de caso, o projeto tem focado o *design* de uma plataforma *wireless*. Este grupo é responsável pelo projeto de prototipação de um dos módulos que farão parte do projeto maior. Para gerenciar o andamento deste trabalho, a equipe do professor Elmar vem fazendo uso da metodologia em desenvolvimento.

Em síntese, a metodologia consiste das seguintes etapas[1]:

1. Levantamento de requisitos
   A lista dos requisitos é uma definição de alto nível das características e funcionalidades. Ela deve conter a definição das possíveis funções e a definição dos limites de cada requisito.

2. Especificação
   A especificação em alto nível dos requisitos levantados.

3. Especificação detalhada
   Especificação bem detalhada do que acha que se pode implementar em um mês.

4. Implementação
   A metodologia usa um padrão para nomes de variáveis, funções, arquivos, etc. Facilitando a manutenção do código por programadores que não o tenham implementado. Parta cada arquivo criado, é obrigatória a inserção de um cabeçalho contendo algumas informações como data da versão, autor, descrição da funcionalidade do arquivo, etc. Deste modo, é mais fácil apontar qual programador implementou qual funcionalidade ou mudança no arquivo. Além disso, é usado um sistema de controle de versões que mantém as modificações feitas pelos participantes da equipe do projeto atualizadas e evita inconsistências (SVN - melhores chances de atravessar firewalls, mais flexível e poderoso do que CVS). Esta etapa é constituída das seguintes partes:

   (a) Criação/obtenção de modelo de referência / modelo comportamental
   O modelo de referência é uma descrição formal da especificação

   (b) Simulação do modelo de referência

   (c) Criação de uma descrição
   A descrição é o resultado de um refinamento manual do modelo comportamental.

   (d) Síntese e verificação do protótipo

5. Verificação
   Nesta etapa são feitos testes e simulações a partir da especificação.

[1] Alguns passos foram omitidos por serem bastante específicos. Foram extraídos apenas os passos considerados relevantes a este trabalho. Para maiores detalhes, consultar [BRA03].

6. Teste real
   Após alguns a fase de verificação, o software é testado em seu hardware alvo.

7. Integração
   Após realizado o teste no hardware, e os módulos estiverem de acordo com a *OCP-IP* [OI], os módulos são integrados.

8. Testabilidade
   E por fim, são usadas ferramentas específicas para o teste final.

Esta metodologia apresentada, ainda não esta totalmente completa, faltando ainda uma base formal em grande parte da mesma. Entretanto, em [dSMA04] é apresentada uma metodologia para testes que é usada na etapa de testes da metodologia em desenvolvimento.

6 Proposta de uma metodologia para desenvolvimento de sistemas embarcados

As consequências de uma falha em um sistema, podem variar de um simples desconforto do usuário, perda de dinheiro momentânea ou definitiva, ou até mesmo a perda de vidas humanas. No caso específico de sistemas embarcados, independente do quão crítico ele seja, uma falha em tempo de produção acarreta em altíssimos custos, pois diferentemente do software muitas vezes sua correção não pode ser feita pelo próprio usuário. Daí, conclui-se que uma metodologia para sistemas embarcados, deve ser muita mais criteriosa no quesito qualidade, na medida em que exige soluções certas desenvolvidas da maneira correta.

Por outro lado, as metodologias ágeis vêm ganhando cada vez mais espaço na indústria de software, principalmente pelo fato de se adequarem melhor a um ambiente incerto, no qual mudanças sempre ocorrem. No entanto, apesar de que temos visto na prática que o uso de metodologias ágeis contribuem para a melhoria da qualidade do software como um todo, no ambiente de sistemas embarcados, principalmente para aqueles que são críticos é necessário que essa qualidade seja comprovada e auditada.

Com isso, propomos aqui uma metodologia híbrida, que seja baseada nas práticas defendidas pelas metodologias ágeis, porém que seja também passível de um controle externo um pouco mais rígido. Abaixo segue um esquema dessa metodologia proposta, que é baseada no desenvolvimento de componentes dentro de uma perspectiva iterativo-incremental:

Levantamento Inicial de Requisitos - Essa é a primeira etapa do processo, na qual o cliente irá definir os requisitos do sistemas, através de *user stories*. Além disso é necessário também o entendimento do domínio do problema onde o sistema será implantado. Nessa fase os requisitos são detalhados apenas o suficiente para se ter uma noção geral do que o sistema deve realizar e o prazo necessário para implementá-lo. Vale ressaltar que devem ser levantados também os requisitos técnicos, que não necessariamente são provenientes do cliente. Após esse levantamento, os requisito devem ser priorizados de acordo com as necessidades do cliente, porém deve ser levando em consideração as possíveis restrições técnicas.

Decomposição em Componentes - Os sistemas definidos através da composição de componentes permitem que sejam adicionadas, removidas e substituídas partes do sistema sem a necessidade de sua completa substituição. Partindo dessa afirmação, é feito uma análise dos requisitos com o intuito de decompor

cada um deles em um ou mais componentes de software. Obviamente que para cada requisito podem existir um ou mais componentes que colaborem entre si para atender a este requisito.

Definição da Arquitetura - Neste momento a equipe deve avaliar as condições técnicas para a implementação dos componentes, e então definir a forma como os mesmos irão interagir. Faz-se necessário também a elaboração do projeto arquitetural de alto nível.

Os passos definidos acima, dizem respeito à fase anterior ao início da implementação propriamente dita. As etapas abaixo fazem parte do ciclo iterativo de desenvolvimento, que será finalizado após todas as funcionalidades (componentes) tenham sido concretizadas e devidamente testadas.

Especificação do(s) Componente(s) - Nesta fase é feita a especificação detalhada dos componentes que serão implementados na próxima *release* (cada release é composta por iterações, assim como no XP, por exemplo). Essa especificação pode demandar um refinamento dos requisitos junto ao cliente. Além disso, modelos e artefatos são usados para documentar essa especificação, pode ser necessário também uso de uma linguagem formal para verificação do modelo. Esse mesmo modelo pode ser usado também para a geração automática de casos de teste, a serem usadas para verificar a conformidade do componente com o modelo.

Planejamento da Iteração - Etapa na qual as tarefas necessárias para desenvolver o(s) componente(s) escolhidos para a próxima *release* são definidas e alocadas entre os pares de programadores.

Realização - Implementação das tarefas alocadas anteriormente, e que deve seguir as seguintes práticas:

- Escrever os testes de unidade antes de codificar;
- Equipes tem liberdade para **sugerir** modificações no design do componente, e isto deve ser feito nas reuniões periódicas de acompanhamento do projeto;
- Uso de métricas para avaliação do processo;
- Quando o fim de uma iteração coincidir com o término de uma *release*, uma outra equipe (preferencialmente formada por revisores experientes e especialistas em testes) deverá ser incumbida de gerar um documento com o parecer sobre a conformidade do(s) componente(s) em relação à sua especificação;

Testes de Integração - Estes testes devem ser realizados a cada novo componente finalizado, também gerando um relatório com os resultados obtidos.

Testes de Aceitação - Eles serão levantados principalmente na fase inicial do projeto, e devem ser realizados para cada requisito, assim que o mesmo possua todos os componentes, necessários à sua realização, implementados.

7 Conclusão

As experiências na indústria de software, com a aplicação de metodologias ágeis em seus processos de desenvolvimento têm mostrado bons resultados, tanto na qualidade do código produzido quanto em relação ao tempo gasto no processo como

um todo. Porém para o contexto de sistemas embarcados a agilidade oferecida por essas metodologias não é a única característica desejável. Faz-se necessário também prover meios de assegurar um nível de confiança aceitável para tais sistemas. Devido a isso, propomos neste trabalho uma abordagem híbrida, que visa suprir a lacuna existente nas metodologias ágeis com mecanismos de controle e gerência mais explícitos.

Referências

[BRA03] BRAZIL IP /CT-INFO. *PROJETO FÊNIX*, 2003. https://lad.dsc.ufcg.edu.br/svn/fenix/metodologia/tronco/metodologia.doc - Visitado em 25/08/2004.

[Bro00] Alan W. Brown. *Large-Scale, Component-Based Development*. Object and Component Technology Series. Prentice Hall PTR, London, Sydney, 2000. ISBN 0-13-088720-X.

[CW03] Luigi Carro and Flávio Rech Wagner. Sistemas embacados. *Simpósio Brasileiro de Computação*, 2003.

[dS01] Wellington João da Silva. Tecnologia java para sistemas embarcados. Technical report, Centro de Informática - UFPE, 2001.

[dSMA04] Karina R. G. da Silva, Elmar U. K. Melcher, and Guido Araujo. An automatic testbench generation tool for a systemc functional verification methodology. *SBCCI 2004 - 17th Symposium on Integrated Circuits and Systems Design*, 2004.

[Gre02] James W. Grenning. Extreme programming and embedded software development. *Embedded Systems Conference*, 2002.

[Int] International Test and Evaluation Association. The DESS ITEA Project Website. http://www.dess-itea.org - Visitado em 25/08/2004.

[MB02] Gary Mueller and Janet Borzuchowski. Extreme embedded a report from the front line. In *OOPSLA 2002 Practitioners Reports*, pages 1–ff. ACM Press, 2002.

[MP02] Ron Morsicato and Mary Poppendieck. Xp in a safety-critical enviroment. *Cutter IT Journal*, september 2002.

[MS02] Ron Morsicato and Nancy Van Schooenderwoerf. Freeing the slave with two masters. an embedded programming team's transition to xp. *Cutter IT*, 15(9):34–41, september 2002.

[OI] OCP-IP. Open Core Protocol International Partnership. http://www.ocpip.org/home - Visitado em 25/08/2004.

[Xis] XisPê. Xispê, supere o medo. http://www.xispe.com.br - Visitado em 25/08/2004.