PADRÕES DE PROTOTIPAÇÃO EM NOVOS PROCESSOS DE RECONHECIMENTO DE PADRÕES E APLICAÇÕES DE SENSORIAMENTO REMOTO: TREINAMENTO ESTRUTURADO SUPERVISIONADO.

Eduardo Lessa Bonelli[1], Luciano Vieira Dutra[2]

Curso de Pós-Graduação em Computação Aplicada (Cap)
Instituto Nacional de Pesquisas Espaciais (INPE) – São José dos Campos – SP – Brazil
dubonelli@dpi.inpe.br, dubonelli@hotmail.com

Abstract. Nowadays almost all the countries need complex systems based on computers. These complexities grow with each demand and the problems related to business rules, productivity, documentation and specification standards let the development task even harder. Thus, researchers have dedicated more efforts on search new prototyping patterns to turn the development process easier and minimize the problems faced in the process during and after development. Several problems in the quality of the processes and softwares are found when the developers have no experience in systems development, pattern recognition, remote sensing and knowledge of Software Engineering. In this context, the goal of this work is to propose a new development methodology based on software engineering paradigms, contributing to increase the application performance through independent structuring of components for Pattern Recognition and Remote Sensing applications based on techniques of structured training.

Resumo. Praticamente todos os países, hoje em dia, dependem de sistemas complexos baseado em computadores. As complexidades crescem a cada demanda e os problemas relacionados às regras de negócios, produtividade, documentações e padrões de especificação tornam a tarefa de desenvolvimento cada vez mais difícil. Assim, pesquisadores têm dedicado esforços em buscar, cada vez mais, novos padrões de prototipação que facilitem o processo de desenvolvimento e minimizem os problemas enfrentados no processo durante e pós-desenvolvimento. Vários problemas na qualidade dos processos e aplicativos são encontrados quando os desenvolvedores não têm experiência na área de desenvolvimento de sistemas, reconhecimento de padrões, sensoriamento remoto e conhecimentos de Engenharia de Software. Neste contexto, o objetivo deste trabalho é propor uma nova metodologia de desenvolvimento baseado nos paradigmas de Engenharia de Software de forma a contribuir ao aumento de desempenho das aplicações via estruturação independente de componentes destinado a processos de Reconhecimento de Padrões e aplicações de Sensoriamento Remoto com base nas técnicas de treinamento estruturado.

Palavras-chave: *Reconhecimento de Padrões, Classificação, Engenharia de Software, Prototipação.*

1 INTRODUÇÃO

1.1 Considerações Iniciais

Nas áreas de Processamento de Imagens e Reconhecimentos de Padrões, por exemplo, o desenvolvimento de sistemas é uma tarefa que consome tempo e exige não apenas conhecimento específico dos algoritmos de Processamento de Imagens, mas também de implementação, de interfaces e principalmente de Engenharia de Software (PAULUS et al.,1999).

Vários problemas são encontrados na questão da qualidade do software quando os desenvolvedores não têm conhecimentos de Engenharia de Software. O processo de projeto, na maioria das disciplinas de engenharia e computação, baseia-se no reuso de sistemas existentes ou componentes. Desenvolvedores de sistemas experientes normalmente não especificam um projeto em que cada componente tenha de ser desenvolvido especialmente. Eles baseiam seus projetos em componentes já experimentados e testados em outros sistemas. Esses componentes não são apenas pequenas funções como cálculo de médias, desvio padrão, mas também incluem subsistemas maiores, como classes responsáveis pelo carregamento de imagens, extração ou seleção de atributos.

A engenharia de software baseada em reuso é uma estratégia de comparação de engenharia de software na qual o processo de desenvolvimento é voltado ao reuso do software existente. Segundo McIIroy (MCIIROY,1968), embora os benefícios do reuso tenham sido reconhecidos por muitos anos, somente nos últimos dez anos houve uma transição gradual do desenvolvimento de software original para o desenvolvimento baseado em reuso. Para Sommerville (SOMMERVILLE, 2007), o movimento para o desenvolvimento baseado em reuso foi uma resposta às demandas por menores custos de produção e manutenção de software, entregas mais rápidas de sistemas e aumento da qualidade do software. Cada vez mais empresas vêem seu software como um ativo valioso e estão promovendo o reuso para aumentar seu retorno sobre investimentos em software.

Nos últimos 20 anos, muitas técnicas foram desenvolvidas para apoiar o reuso de software. Elas exploraram o fato de que sistemas em um mesmo domínio de aplicações são similares e têm potencial de reuso, que o reuso é possível em diferentes níveis (a partir de funções simples até aplicações completas) e que padrões para componentes reusáveis facilitam o reuso. Dados esse conjunto de técnicas de reuso, a questão principal é: qual a técnica mais apropriada e quais as vantagens em usá-las? Tais abordagens serão tratadas mais adiante no presente trabalho.

Na área de Processamento de Imagens essa prática é sempre presente, pois, geralmente, estudantes e profissionais com formação e conhecimentos em outras áreas são destinados a desenvolver novos processos de Reconhecimento de Padrões e aplicações de Sensoriamento Remoto. Muitas bibliotecas e pacotes estão disponíveis no mercado, mas ainda assim isso não resolve toda a necessidade exigida, pois possui recursos limitados a área de desenvolvimento de sistemas.

Segundo Paulus (PAULUS, 1999), o maior problema em desenvolver um processo genérico para sistemas de processamento de imagens é que, por uma lado, é

desejável eficiência em tempo de execução e acesso a hardware em baixo nível. Por outro lado, deseja-se a implementação com modularidade, fraco acoplamento, reusabilidade e independência de plataforma.

Além disso, o desenvolvimento de sistemas depende fortemente de especificação de requisitos para produzir uma solução correta, confiável e de fácil manutenção, o que torna a Engenharia de Software imprescindível em todas as fases do desenvolvimento (LOWE e LEANEY, 2008). Sendo assim, para entender esses requisitos é necessário que o desenvolvedor possua uma formação multidisciplinar envolvendo a área específica de processamento de imagens, programação e engenharia de software.

No que se refere a sistemas de Reconhecimento de Padrões, existe também o problema relacionado a parametrização de dados e o padrão de prototipação dos processos existentes nos classificadores. Geralmente, os algoritmos para classificação de imagens na área de Processamento de Imagens e Reconhecimento de Padrões são os mesmos (vide WEBB, 2002; THEODORIDIS e KOUTROUMBAS, 2003), mas esses sistemas não se comunicam devido à incompatibilidade no formato dos dados. Neste contexto, é viável adotar um formato padrão dos dados de forma a possibilitar o intercâmbio de informações entre os métodos dos classificadores e entre os sistemas.

Segundo Sommerville (SOMMERVILLE, 2007), durante o processo de desenvolvimento de sistema, independente de sua natureza, as revisões de software são onerosas, demoradas e, inevitavelmente, atrasam a conclusão de um sistema de software. Idealmente, seria possível acelerar o processo de revisão por meio de ferramentas para processar o projeto ou programa de software e fazer algumas avaliações automatizadas da qualidade de software.

Essas avaliações podem verificar que o software atingiu o limiar de qualidade necessária e, quando isso não ocorrer, aconselha-se destacar essas áreas que a revisão deve enfocar. Essa abordagem será descrita com maiores detalhes no decorrer do trabalho, pois a etapa de avaliação de treinamento dos classificadores é importante em novos processos de Reconhecimento de Padrões e aplicações de Sensoriamento Remoto.

1.2 Objetivos do Trabalho

Frente a essa problemática, pesquisadores têm dedicado seu tempo no sentido de buscar novas abordagens, padrões, metodologias e paradigmas visando principalmente a qualidade, reuso, rapidez e facilidade no desenvolvimento. Neste contexto, o objetivo deste trabalho é propor uma nova metodologia de desenvolvimento baseado nos paradigmas de Engenharia de Software de forma a contribuir ao aumento de desempenho das aplicações via estruturação independente de componentes e amenizar os problemas citados acima.

2 PARADIGMAS DE ENGENHARIA DE SOFTWARE

2.1 Considerações Iniciais

Segundo Wada (WADA, 2006), para um melhor entendimento sobre técnicas de gestão de projetos, é necessário ter o conceito da palavra projeto muito bem definido. Assim, projeto pode ser interpretado como algo planejado, que surgiu da idéia ou necessidade para atender algum fim específico e que possui uma série de fatores influentes que podem impactar no sucesso ou insucesso deste, dentro de um período estipulado.

Desta maneira, projeto é um conjunto de atividades a serem executadas através de ações, a fim de se atingir o objetivo final dentro de um prazo limitado de tempo, seguindo estas atividades uma sequência lógica definida. Muitas vezes, um objetivo é constituído por vários objetivos secundários. Utilizando a definição de projetos pelo PMBOK – 3° Edição, temos:

"Um projeto é um esforço temporário empreendido para criar um produto, serviço ou resultado exclusivo"

Conforme descrito anteriormente, um projeto possui algumas características essenciais para a sua sobrevivência. Projetos incluem um escopo definido, um orçamento, um resultado específico e recursos designados. Outra característica de um projeto é que o trabalho é único. Mesmo se um projeto for semelhante a outro, não será exatamente o mesmo porque as circunstâncias mudam conforme os ambientes culturais de cada empresa.

Muitas pessoas associam o termo software aos programas de computador. Na verdade, essa é uma visão muito restritiva. Software não é apenas o programa, mas também todos os dados de documentação e configuração associados, necessários para que o programa opere corretamente. Segundo Sommerville (SOMMERVILLE, 2007), um sistema de software consiste, geralmente, de um conjunto de programas separados; arquivos de configuração, que são utilizados para configurar esses programas; documentação do sistema, que descreve a estrutura do sistema; a documentação do usuário, que explica como usar o sistema.

Um processo de software é um conjunto de atividades e resultados associados que produz um produto de software. Assim, existem quatro atividades fundamentais para o processo de desenvolvimento de um software. São: especificação do software, projeção e desenvolvimento do software, validação de software e evolução de software que corresponde às possíveis modificações visando adaptar às mudanças dos requisitos por parte do cliente e do mercado.

Um modelo de processo de software pode ser entendido como uma descrição simplificada desse processo de software que apresenta uma visão dele. Segundo Lowe e Leaney (LOWE e LEANEY, 2008), os modelos de software incluem as atividades, que fazem parte do processo de software. Alguns exemplos dos principais tipos de modelo de processo de software que podem ser produzidos e que serão utilizados no processo de criação dos novos processos de prototipação apresentados neste trabalho são:

1. *Um modelo de workflow:* mostra a sequência de atividades ao longo do processo, com suas entradas, saídas e dependências entre elas. As atividades neste modelo podem representar ações humanas.
2. *Um modelo de fluxo de dados ou modelo de atividade:* representa o processo como um conjunto de atividades, no qual cada atividade realiza alguma transformação de dados, ou seja, mostra como deve ser realizada a entrada de dados do processo, no formato de especificação, por exemplo, gerando o formato de saída deste processo.

2.2 Modelo de Processos de Software

Diante da diversidade natural dos softwares, a maioria dos modelos de processos de software utilizados é baseada em um dos três modelos gerais ou paradigmas de desenvolvimento de software, Sommerville (SOMMERVILLE, 2007):

1. *Modelo em cascata:* considera as atividades apresentadas anteriormente e as representa como fase separadas de processo, como especificação de requisitos, projeto de software, implementação, teste e assim por diante. Depois que cada estágio é concluído, ele é aprovado e o desenvolvimento prossegue para o estágio seguinte.
2. *Desenvolvimento iterativo:* esta abordagem intercala as atividades de especificação, desenvolvimento e validação. Um sistema inicial é desenvolvido rapidamente com base em especificações abstratas. È então refinado com as informações do cliente, para produzir um sistema que atenda as necessidades deste. O sistema pode, então, ser entregue. Como alternativa, ele pode ser reimplementado, utilizando uma abordagem mais estruturada, para produzir um sistema robusto e mais fácil de ser mantido.
3. *Engenharia de software baseada em componentes (CBSE – Component Based Software Engineering)*: esta técnica supõe que partes do sistema já existam. O processo de desenvolvimento concentra-se mais na integração dessas partes do que no seu desenvolvimento a partir do início.

Desta forma, devido a complexidade dos processos de Reconhecimento de Padrões e aplicativos de Sensoriamento Remoto em especial os classificadores baseado em treinamento estruturado supervisionado, o último modelo apresentado terá mais ênfase neste trabalho sendo melhor detalhado mais adiante.

2.3 Engenharia de Software Baseada em Componentes (CBSE)

Como descrito anteriormente, a engenharia de software baseada em reuso está, cada vez mais, se tornando a principal abordagem de desenvolvimento para sistemas corporativos e comerciais. As entidades reusadas variam de pequenas funções a completos sistemas de aplicação. Entretanto, a reutilização de procedimentos individuais, com maior número de funcionalidade, era difícil.

Assim, com a contribuição dos fornecedores de software referente aos avanços na padronização possibilitando a interoperação de componentes dentro de um framework, criou a oportunidade de reuso sistemático por meio da engenharia de software baseada em componentes.

A engenharia de software baseada em componentes (CBSE – Component-Based Software Engineering) surgiu no final da década de 1990 como uma abordagem baseada em reuso para desenvolvimento de sistemas de software. Sua criação, segundo Sommerville (SOMMERVILLE, 2007), foi motivada pela frustração dos projetistas quanto ao desenvolvimento orientado a objetos não serem conduzidos ao reuso amplo, como originalmente sugerido. Classes de objetos individuais eram muito detalhadas e específicas e, muitas vezes, tinham de ser ligadas a uma aplicação no momento da compilação. Era necessário conhecer detalhadamente as classes e usá-las, o que significava geralmente que era necessário possuir o código-fonte do componente. Tornado assim, difícil a idéia de que os objetos eram componentes reusáveis.

Há na comunidade científica um consenso de que um componente é uma unidade de software independente que pode ser composta por outros componentes para criar um sistema de software. Segundo Councill e Heineman (COUNCILL e HEINEMAN, 2001), um elemento de software que esta em conformidade com um modelo de componente e pode ser independentemente implantado e composto sem modificação de acordo com um padrão de composição.

Essa definição é baseada em padrões, segundo Sommerville (SOMMERVILLE, 2007), uma unidade de software que esteja em conformidade com esses padrões é um componente. Já Szyperski (SZYPERSKI, 2002), define um componente de software como uma unidade de composição com interfaces contratualmente especificadas e dependências de contexto explícitas somente. Um componente de software pode ser implantado independentemente e está sujeito à composição de terceiros.

Para Peter e Pedrycz (PETER e PEDRYCZ, 2001), um componente de software é uma unidade que pode ser adaptada, portada para outras plataformas, reutilizável e que pode ser testada separadamente para alguma finalidade específica.

Essas definições formais de componente não fornecem realmente um quadro claro do que um componente faz. Uma das maneiras mais úteis de se considerar um componente é como um provedor de serviços independente. Quando um sistema necessita de algum serviço, chama um componente para fornecer este serviço sem a preocupação de como e onde este serviço esta sendo executado ou ainda a linguagem de programação usada para desenvolver o componente.

Por exemplo, um componente que faz a conversão de um formato gráfico em outro (por exemplo, TIFF em JPEG) fornece um serviço de conversão de dados, ou ainda, um componente que realiza o processo de arquivamento em disco rígido de uma determinada saída de método.

Um modelo de componente é uma definição de padrões para implementação, documentação e implantação de componentes. Esses padrões são para que os desenvolvedores, experientes ou com pouco conhecimento na área, se assegurem de que os componentes podem operar entre si. Diante dessas informações e necessidades em

novos processos de Reconhecimento de Padrões e aplicativos de Sensoriamento Remoto a solução será conduzida.

2.4 Processos de Evolução

Os processos de evolução de software variam consideravelmente, dependendo do software a ser mantido, dos processos de desenvolvimento usados e das possíveis necessidades de mercado ou de operação. Em algumas organizações, a evolução pode ser um processo informal no qual as solicitações de mudanças, na maioria das vezes, provêm de conversas entre os usuários e desenvolvedores. Em outras empresas, é um processo formalizado, com documentação estruturada e produzida em cada estágio do processo.

Diante de um contexto em que softwares são desenvolvidos por equipes, conforme descrito anteriormente. A necessidade de uma documentação é primordial para a agilidade, compreensão das nomenclaturas utilizadas (variáveis) nos códigos-fonte, transição dos métodos, segurança de informação e padronização no intercambio de dados entre as camadas.

As propostas de mudanças podem envolver requisitos ainda não implementados no sistema liberado, solicitações de novos requisitos ou reparo de defeitos detectados pelos *stakeholders* (financiadores) do sistema, além de novas idéias e propostas de aprimoramento de software solicitadas pela própria equipe de desenvolvimento.

Conforme ilustrado na Figura 1, os processos de identificação de mudança e evolução são cíclicos e continuam ao longo do ciclo de vida de um sistema. Desta forma, a existência de uma documentação é importante em todos os processos, não apenas na fase de desenvolvimento inicial.

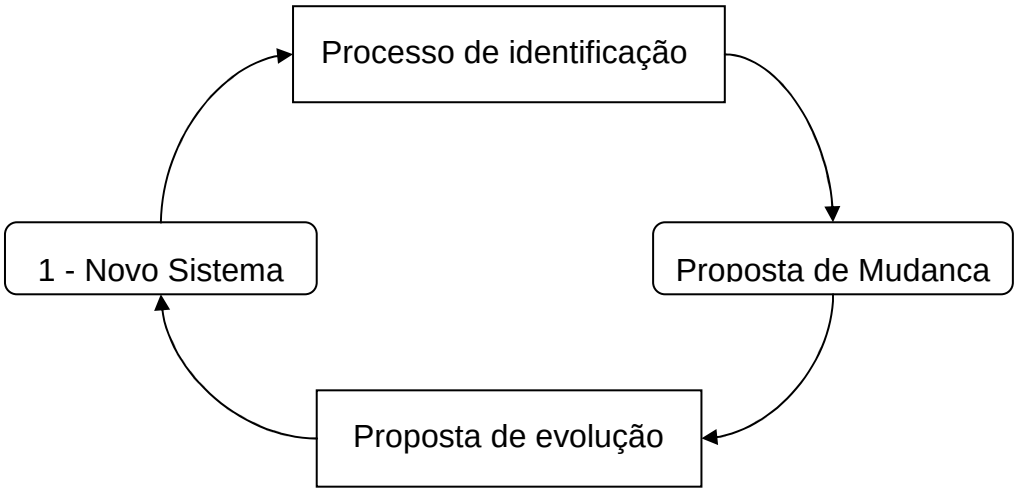

Figura 1 Processos de identificação de mudança e evolução (Sommerville, 2007)

Segundo Lowe e Leaney (LOWE e LEANEY, 2008), o processo de evolução inclui as atividades fundamentais de análise de mudanças, planejamento de releases, implementação de mudanças e liberação de um sistema para os usuários. O custo e o

impacto dessas mudanças são avaliados para descobrir quanto do sistema é afetado pelas mudanças e quanto pode custar para implementar a mudança.

Durante o planejamento de releases, todas as mudanças propostas (reparo de defeitos, adaptações e novas funcionalidades) são consideradas. Uma decisão deve habilitar quais mudanças devem ser implementadas na próxima versão do sistema. As mudanças são implementadas e validadas, e um novo release do sistema é liberado. O processo se repete com um novo conjunto de mudanças propostas para o próximo release. A figura 2, adaptada de Sommerville (SOMMERVILLE, 2007), mostra uma visão geral desse processo.

O processo de implementação de mudança é, essencialmente, uma iteração do processo de desenvolvimento na qual as revisões do sistema são projetadas, implementadas e testadas. O presente trabalho, conforme descrito anteriormente visa propor um modelo padrão de prototipação no processo de desenvolvimento de novos processos de Reconhecimento de Padrões e aplicativos de Sensoriamento Remoto demonstrando detalhadamente toda a questão formal de construção e documentação dos componentes necessários para um classificador genérico e elaboração de novas releases. Será utilizado o classificador estruturado supervisionado Maxver, com o intuito de demonstrar tal uniformização.

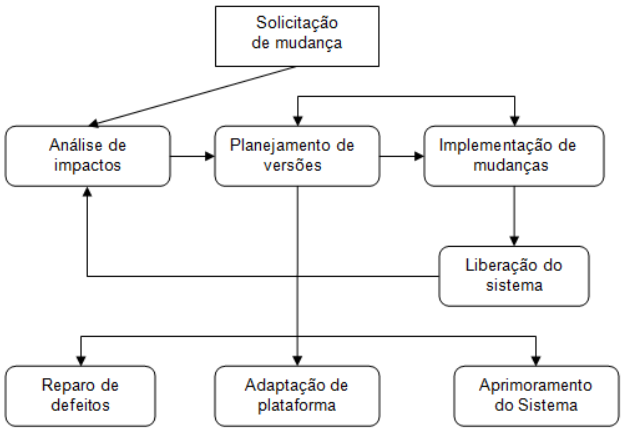

Figura 2 Processos evolução de sistema (Sommerville, 2007)

2.4 Reengenharia de sistemas

O processo de evolução, segundo Sommerville (SOMMERVILLE, 2007), envolve a compreensão do programa a ser mudado e, depois, a implementação das mudanças. Contudo, muitos sistemas, especialmente os sistemas legados mais antigos e sistemas desenvolvidos por equipes distintas, são difíceis de compreender e de mudar.

Para simplificar os problemas de mudança de sistemas, a reengenharia de software pode ser uma alternativa quando as questões citadas acima são relevantes. Este processo esta relacionado à reimplementação de sistemas para torná-los mais fáceis de manter. A reengenharia pode envolver uma nova documentação, organização e reestruturação do sistema, conversão do sistema em uma linguagem mais moderna,

modificação e atualização da estrutura e dos valores dos dados de sistema e de novos padrões de prototipação para o desenvolvimento.

Diante dessas necessidades, há quatro estratégias de abordagens quanto a evolução dos sistemas: descartar o sistema completamente e iniciar um novo projeto sob os conceito de planejamento e gerência, deixar o sistema sem alterações e continuar com a manutenção regular, utilizar reengenharia de software para aprimorar sua facilidade de manutenção e evolução, conforme descrito anteriormente ou substituir todo ou parte do sistema por um outro sistema.

3 ENGENHARIA DE SOFTWARE - PROCESSOS DE RECONHECIMENTO DE PADRÕES E APLICAÇÕES DE SENSORIAMENTO REMOTO

O processo de desenvolvimento de sistemas, conforme tratado anteriormente, não esta relacionado exclusivamente a produção de códigos a fim de solucionar uma necessidade. Um aplicativo de alta qualidade está diretamente relacionado a altos níveis de modelagem e estratégias de desenvolvimento.

Desta forma, diversos paradigmas de Engenharia de software podem ser utilizados a fim de contribuir com a diminuição de problemas encontrados no processo de desenvolvimento de aplicações em geral. Contudo, diante da complexidade das operações é pertinente identificar os processos comuns que se encontram distribuídas por todo o código visando a sua prototipação e padronizações das operações básicas a fim de facilitar a documentação, reuso, manutenção, repetição de métodos e com isso ganhar agilidade nos processos críticos.

Assim, este trabalho visa padronizar as diversas operações dos aplicativos de sensoriamento remoto e processos de reconhecimento de padrões com base nos paradigmas de Engenharia de Software. Esta padronização abordará documentação, métodos, requisitos e meios de utilização das funcionalidades de forma intuitiva, contribuindo para a elaboração de uma nova metodologia de desenvolvimento. Esta prototipação usará como abordagem a exemplificação do processo de classificação supervisionado, Maxver.

Para o melhor entendimento dos fluxos existentes nos processos de reconhecimento de padrões e aplicações de sensoriamento remoto é necessário conhecer algumas definições realizada por pesquisadores consagrados sobre tais processos. Para Therrien (THERRIEN, 1989), a meta de reconhecimento de padrões é a classificação de objetos de interesse em um número de categorias ou classes. Segundo Theodoridis (THEODORIDIS, 2006), reconhecimento de padrões é uma disciplina científica cuja meta é a classificação de objetos em categorias ou classes.

Estatisticamente reconhecimento de padrões, segundo Webb (WEBB, 2002), é um termo usado para a convergência de todos os estágios de investigação na formulação de problemas e coleção de dados através da discriminação e classificação, avaliação de resultados e interpretação.

Um padrão, segundo Watanabe (WATANABE, 1985) é a oposição do caos; é uma entidade, vagamente definida, que pode receber um nome. Alguns exemplos de

padrões podem ser encontrados em: Sinal da voz, assinaturas, códigos de barras, impressão digital, uso e cobertura do solo, dentro outros.

O elemento atributo na área de reconhecimento de padrões pode ser considerado uma quantia mensurável obtida de um objeto ou padrão. Alguns exemplos podem ser dados para o melhor entendimento, são: cor, marca, modelo de um determinado carro; altura, peso, nome de uma determinada pessoa, as possíveis bandas de uma imagem de satélite, dentre outras.

Uma classificação pode ser entendida como o particionamento do espaço de atributos em regiões de decisão, sendo estas denominadas pertencentes a uma classe. A fronteira entre as regiões são conhecidas como superfície de decisão.

A idéia principal dos algoritmos de reconhecimento de padrões e aplicativos de sensoriamento remoto é encontrar a fronteira de decisão entre as classes e rotulá-las de acordo com um modelo pré-definido. Assim, um aplicativo ou algoritmo de classificação, busca particionar o espaço de atributos em regiões de interesse e a partir desses dados avaliar em qual classe a região de interesse encontrada pertencerá.

Para o desenvolvimento de um processo ou aplicativo cuja finalidade é a classificação de regiões de interesse, algumas etapas são estritamente necessária para uma classificação de alto nível de avaliação e qualidade. A figura 13 ilustra o diagrama de fluxos de um classificador genérico, segundo Theodoridis (THEODORIDIS, 2006).

A partir do fluxo existente, é notável a presença de níveis de operações (etapas) necessárias para um projeto genérico de classificação. Desta forma, através da metodologia desenvolvida serão geradas e aplicadas as especificações a todos os níveis de operações presente no fluxo de desenvolvimento, contribuindo com a redução de dimensionalidade, redução na perda de informações durante a transição das operações, aumento da qualidade e principalmente aumento da facilidade de compreensão do processo como um todo por parte dos desenvolvedores inexperientes.

A contribuição científica presente neste trabalho destina-se além da facilidade e compreensão dos métodos, flexibilidade na manutenção dos componentes existentes e criação de novos componentes em qualquer etapa do processo utilizando a nova metodologia proposta, de forma a diminuir a perda de desempenho.

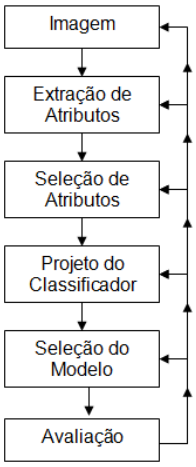

Figura 3 Sistema de Classificação (Theodoridis,2006)

4 CONCLUSÃO

Na atualidade a grande maioria dos problemas encontrados nos processos de desenvolvimento de software ocorre devido a não utilização da Engenharia de Software ou a má execução de seus conceitos. Diante da grande oferta de ferramentas, técnicas e conceitos existentes no mercado provenientes do avanço da tecnologia, deve-se ressaltar que nenhum paradigma constitui uma solução trivial.

Os problemas encontrados na área de tecnologia da informação com foco em desenvolvimento de sistemas abrangem questões de produtividade, portabilidade, qualidade, levantamento de requisitos e documentação. Contudo, antes de adotar uma determinada abordagem de desenvolvimento, é necessário entender a natureza do problema e os conceitos inseridos no contexto.

Para Stairs e Reynolds (STAIRS E REYNOLDS, 1998), “um sistema de informação é um conjunto de componentes inter-relacionados que coletam, manipulam e disseminam dados e informações para proporcionar um mecanismo de realimentação para atingir um objetivo”.

Os sistemas de informação estão mudando constantemente a forma como os representantes das organizações conduzem seus negócios. As informações têm valor e o comércio freqüentemente trata o intercâmbio de informações em contrapartida aos bens tangíveis. Os sistemas de informação são usados para criar, armazenar e transferir informações. (PINHEIRO, 2004)

Alinhar os objetivos organizacionais e os objetivos do Sistema de Informação são essenciais para qualquer iniciativa de desenvolvimento de sistemas, já que os Sistemas de Informação são ferramentas de apoio a outras atividades. Assim, tanto a equipe de Sistema de Informação quanto áreas envolvidas precisam alinhar os objetivos e entender as responsabilidades e tarefas mutuamente.

A Tecnologia da Informação está cada vez mais presente no dia-a-dia das pessoas e das empresas, e é a área com maior crescimento em tecnologia e investimentos. Devido à grande relevância da informação no cenário globalizado, é imprescindível o reconhecimento da importância da área, o que certamente modificará o cenário competitivo ao seu redor, gerando vantagens competitivas em relação aos concorrentes.

Contudo, os problemas enfrentados por desenvolvedores de aplicativos de Sensoriamento Remoto e processos de Reconhecimento de Padrão não serão solucionados de forma rápida, pois a simples adoção de ferramentas, conceitos e técnicas, reusabilidade de componentes e estruturas, além da linguagem de programação utilizada não necessariamente leva a produção de software com qualidade. É preciso propor soluções partindo do âmbito da necessidade para o ambiente de implementação. Nessa abordagem, a adoção de padrões de modelagem, desenvolvimento e documentação são essenciais para o sucesso do projeto.

Com base na abordagem realizada, o presente trabalho visa propor uma nova metodologia concretizada em padrões de prototipação com foco no desenvolvimento de aplicações de sensoriamento remoto e processos de reconhecimento de padrões de forma a aumentar a compreensão de cada etapa dos processos de reconhecimento de padrões e

contribuir para o aumento da qualidade, facilidade no desenvolvimento e sustentabilidade do projeto ao longo do tempo.

REFERÊNCIAS BIBLIOGRÁFICAS

COUNCILL, W. T.; HEINEMAN, G. T. (2001) Definition of a software component and its elements. In Component-based software engineering. Boston: Addison-Wesley, 5-20. Chapter 19.

LOWE, D. B.; LEANEY, J. R. (2010) The impact of Software Engineering on Image Processing. Disponível em: http://citeseer.ist.psu.edu/53986.html.

MCIIROY, M. D. (1968) Proc. Nato conf. on software engineering, Garmisch, Germany: **Mass-produced software components** – chapter 18 (Springer-Verlag).

PAULUS, D.; HORNEGGER, J.; NIEMANN, H. (1999). Software Engineering for Image Processing and Analysis. Handbook of Computer Vision and Applications – chapter 3 (Systems and Applications). Academic Press.

PETER, J. F.; PEDRYCY, W. (2001) Engenharia de Software: teoria e prática. Campus.

PINHEIRO. J. M. S. (2004) Guia completo desenvolvimento de aplicações. Capítulo 2, página 23-42.

SOMMERVILLE, I. (2007) Engenharia de Software. 8° edição. Pearson Education.

SZYPERSKI, C. (2002) Component Software: beyond object-oriented programming, 2nd edn. Harlow: Addison-Wesley. Chapter 12 and 19.

STAIR, R. M.; REYNOLDS, G. W. (1998) Princípios de Sistemas de Informação: Uma abordagem gerencial**.** 2. ed. Rio de Janeiro: LTC.

THEODORIDIS, S., KOUTROUMBAS, K. (2003) Pattern Recognition. Second edition. Academic Press.

THEODORIDIS, S., KOUTROUMBAS, K. (2006) Pattern Recognition, 3nd ed., Academic Press, feb. (4th 2008).

THERRIEN, C.W. (1989) Decision, Estimation and Classification: An Introduction to Pattern Recognition and Related Topics. John Wiley & Sons Inc, Feb.

WADA, R. (2006) Planejamento e Gerência de Projetos com ênfase no PMI. 1° Edição. Impacta Tecnologia.

WEBB, A.R. 2002 Statistical Pattern Recognition, John Wiley & Sons, 2nd ed., oct.

WATANABE, S. (1985) Pattern Recognition: Human and Mechanical .Wiley, New York.