TRIBUNAL REGIONAL ELEITORAL DO AMAPÁ
SECRETARIA DE TECNOLOGIA DA INFORMAÇÃO

Processo de Desenvolvimento de Software do TRE-AP (PDS/TRE-AP)

Manual do Processo

Controle de Versões

Versão	***Data***	***Responsável***	***Descrição***
1.0	*20/10/2017*	*Davi Shibayamas (SDS/CSCI)*	*Versão inicial*
1.1	*30/03/2022*	*Elinete Freitas (CSC/STI)*	*Revisão: reestruturação do document e alinhar com a Política de Desenvolvimento de software (Portaria 120/2021).*

Apresentação

O processo de desenvolvimento de software é um conjunto de procedimentos estratégicos e organizados com o propósito de arquitetar, desenvolver e testar o software. Além de prestar manutenção para garantir seu bom funcionamento.

Um software bem desenvolvido oferece muitos benefícios para a organização. É um produto capaz de otimizar várias áreas e serviços de todos os tipos de segmento, além de ajudar na redução de custos. No entanto, para que seja entregue a melhor solução possível, o processo de desenvolvimento de software deve seguir as melhores práticas da área.

Por envolver muito planejamento e estratégia, além de muitas análises, manutenções, implementações e testes, o software possibilita automatizar diversas aplicações. Cada codificação tem suas particularidades justamente para personalizar o produto de acordo com as necessidades do usuário.

Ou seja, a importância do desenvolvimento de sistemas está na eficiência de solucionar problemas de modo automatizado. Isso contribui para a otimização de tempo, serviços e funções dentro da organização. Inclusive, possibilita que haja redução de custos a longo prazo.

A presente versão 1.1, após quase cinco anos da edição inicial, traz um novo visual, fluxo aperfeiçoado e alinhamento com a Política Organizacional de Desenvolvimeno de Software do TRE-AP (Portaria nº 120/2021), que juntos visam proporcionar um trabalho mais eficiente para quem for utilizá-la. Nela estão consolidadas as contribuições das unidades que aplicaram o processo.

Sumário

1 Introdução

1.1 Processo orientado a pessoas

Segundo SOMMERVILLE (2011, p.18), "*processo de software é um conjunto de atividades inter-relacionadas que levam à produção de um produto de software*".

Essa definição, apesar de coerente, coloca de lado uma parte essencial para a criação de qualquer software: as **PESSOAS**! De fato, segundo BROOKS (2009), o desenvolvimento de software é um processo **criativo** e, ainda que uma firme metodologia possa fortalecer e dar liberdade às mentes criativas, ela não pode acender ou inspirar um trabalhador entediado. Nesse mesmo sentido, FOWLER (2005) é mais incisivo ao afirmar que **nenhum processo, por mais bem elaborado e organizado que seja, jamais irá substituir as habilidades de uma equipe de desenvolvimento**. Para ele, o papel de um processo é dar suporte à equipe de desenvolvimento no seu trabalho.

O fator crítico de sucesso para a concepção de um novo produto de software, portanto, não recai tanto sobre a utilização de um processo bem definido e estruturado, com detalhamento minucioso de regras, papéis e atividades, embora isso tenha o seu devido valor, mas sim, primordialmente, **no comprometimento e engajamento de pessoas bem capacitadas, motivadas e que consigam enxergar no seu trabalho uma oportunidade de melhorar o mundo ao seu redor**. A propósito disso, DE MASI (2000) deixa claro que no trabalho intelectual (que é o caso do desenvolvimento de software) **a motivação é tudo**.

É fundamental reconhecer que o processo de software difere diametralmente de outros processos de negócio convencionais das organizações, principalmente daqueles análogos a uma linha de produção em massa de uma fábrica. Nesses casos, o objetivo de estruturar e documentar um processo é tornar possível que qualquer pessoa, independentemente do seu nível de conhecimento e habilidade, possa executá-lo sem maiores dificuldades e, ainda assim, os serviços ou produtos resultantes ficarão dentro do esperado. Com o processo de software, infelizmente, isso não funciona dessa forma (DEMARCO & LISTER, 2013). Isso porque o desenvolvimento de software, como indica LARMAN (2004), é classificado como **desenvolvimento de um novo produto** e, assim o sendo, os princípios e as práticas da indústria de produção em massa não se aplicam a ele. Ao contrário, tentar aplicar as técnicas Tayloristas (TAYLORISMO, 2017) de racionalização do trabalho manual e fabril no campo do desenvolvimento de software, cuja natureza é completamente diferente, já que se trata de um trabalho criativo puramente abstrato e intensamente cognitivo, podem trazer resultados negativos, ao invés de elevar a sua eficiência e eficácia, conforme demonstraram DEMARCO & LISTER (2013), DE MASI (2000) e TELES (2005).

Assim, chega-se à conclusão de que, ao contrário do que pregam as metodologias tradicionais, as pessoas envolvidas no desenvolvimento de software não são partes substituíveis no processo, mas sim são o aspecto mais importante e decisivo para o sucesso, ou não, de qualquer projeto.

1.2 Premissas

Qualquer processo de desenvolvimento de software deve considerar, pelo menos, as seguintes premissas sobre a natureza desse trabalho:

- O desenvolvimento de software é um trabalho de **criação intelectual**, não linear, abstrato e que precisa de alto grau de organização de raciocínio, criatividade e inventividade, em oposição ao trabalho manual, repetitivo e fabril de produção em massa;

- As pessoas envolvidas na criação do software precisam ser capazes de **assumir proativamente a responsabilidade pelo seu trabalho e tomar a iniciativa de resolver problemas** à medida que eles surgem (ELSSAMADISY, 2009);

- As pessoas envolvidas na criação do software **devem ser capazes de identificar a melhor forma de fazer o seu próprio trabalho** (BECK, 2001; FOWLER, 2005);

- As pessoas envolvidas na criação do software **devem ser excelentes em comunicação**, tanto falada quanto escrita, ou seja, devem ser capazes de transmitir ideias e informações com clareza e objetividade, assim como interpretar de forma precisa o que lhes é comunicado;

- O desenvolvimento de software **deve buscar entregar o máximo de valor de negócio ao Solicitante desde o primeiro momento do projeto** e maximizar o retorno sobre o investimento (SCHWABER, 2004);

- O grau de formalidade e o peso (quantidade de regras, cerimônias, papéis, artefatos e etc.) de um processo de software **deve sempre se adequar ao nível de criticidade de cada demanda a ser atendida**. Quanto menor a criticidade do software a ser criado, menor deve ser o peso do seu processo de desenvolvimento (COCKBURN, 2006);

- O desenvolvimento de software exige intenso esforço de **aprendizagem contínua**. Segundo ELSSAMADISY (2009), em qualquer projeto de criação de um novo software, **a aprendizagem é o maior gargalo para sua conclusão bem-sucedida** e, nesse contexto, sem a aprendizagem frequente, a equipe de desenvolvimento nunca será capaz de reconhecer e responder às mudanças que ocorrem. A aprendizagem é necessária não apenas para selecionar e utilizar adequadamente as tecnologias e suas infraestruturas auxiliares, mas, principalmente, é crucial para compreender o domínio do problema e identificar precisamente os fatores críticos de sucesso dos seus processos de negócio;

- A **complexidade** é uma característica inerente ao desenvolvimento de software (BROOKS, 2009). Segundo Brooks, isso é devido ao elevado número de **elementos distintos** que compõem qualquer produto de software e, consequentemente, a quantidade elevada de **estados distintos** desses elementos, o que os torna mais complexos que qualquer outro tipo de construção humana. Essa constatação levou esse autor a afirmar que **não existe bala de prata** (solução única e milagrosa) que modifique essa condição inata da complexidade do software. Acrescentado a isso, SCHWABER (2004) evidencia que o desenvolvimento de software precisa lidar com três tipos de complexidade que, quando são reunidas e interagem entre si, elevam o nível de complexidade ao extremo. São elas: a complexidade dos **requisitos** do domínio do problema, a complexidade das **tecnologias** utilizadas e a complexidade das **pessoas** envolvidas;

- Os dois principais **objetivos do desenvolvimento de software** são:
    - Garantir que o software criado **resolva o problema do Solicitante**, da forma mais simples, eficiente e eficaz possível; e
    - Garantir que o software criado tenha o **menor custo de manutenção possível**.

1.3 Valores e Princípios

A corrente de pensamento que mais se aproxima das premissas elencadas acima é aquela que foi estabelecida pelo "**Manifesto para o desenvolvimento ágil de software**" (BECK, 2001), cujos valores estão transcritos a seguir:

> "***Manifesto para o desenvolvimento ágil de software***
>
> *Estamos descobrindo maneiras melhores de desenvolver software, fazendo-o nós mesmos e ajudando outros a fazerem o mesmo. Através deste trabalho, passamos a valorizar:*
>
> - ***Indivíduos e interações entre eles*** *mais que processos e ferramentas*
> - ***Software em funcionamento*** *mais que documentação abrangente*
> - ***Colaboração com o cliente*** *mais que negociação de contratos*
> - ***Responder às mudanças*** *mais que seguir um plano*
>
> *Ou seja, mesmo havendo valor nos itens à direita, valorizamos mais os itens à esquerda."*

Os doze princípios estabelecidos por esse manifesto, transcritos abaixo, também vão ao encontro das premissas listadas anteriormente:

> "***Princípios por trás do manifesto ágil***
>
> *Nós seguimos os seguintes princípios:*
>
> - *Nossa maior prioridade é satisfazer o cliente, através da* ***entrega antecipada e contínua de software de valor****.*
> - ***Aceitar mudanças de requisitos****, mesmo no fim do desenvolvimento. Processos ágeis se adequam às mudanças, para que o cliente possa tirar vantagens competitivas.*
> - ***Entregar software funcionando com frequência****, na escala de semanas até meses, com preferência aos períodos mais curtos.*
> - *Pessoas relacionadas aos negócios e desenvolvedores* ***devem trabalhar em conjunto e diariamente****, durante todo o curso do projeto.*
> - *Construir projetos ao redor de* ***indivíduos motivados****, dando a eles o ambiente e suporte necessário e* ***confiar que farão seu trabalho****.*
> - *O método mais eficiente e eficaz de transmitir informações para, e por dentro de uma equipe de desenvolvimento, é através de uma* ***conversa frente a frente****.*
> - ***Software funcional*** *é a medida primária de progresso.*
> - *Processos ágeis promovem um* ***ambiente sustentável****. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter, indefinidamente, passos constantes.*
> - *Contínua atenção à* ***excelência técnica e bom design****, aumenta a agilidade.*
> - ***Simplicidade****: a arte de maximizar a quantidade de trabalho que não precisou ser feito.*
> - *As melhores arquiteturas, requisitos e designs emergem de equipes* ***auto organizáveis****.*
> - *Em intervalos regulares,* ***a equipe reflete sobre como ficar mais efetiva****, então se ajustam e otimizam seu comportamento de acordo."*

De forma a complementar os valores e princípios do Manifesto Ágil e tomando por base a proposta de BASTOS (2013), é importante que seja valorizado também:

- *Não apenas software em funcionamento,* ***mas software feito a partir de um CLARO entendimento***
- *Não apenas responder a mudanças,* ***mas SÓ fazer o que agrega valor***
- *Não apenas indivíduos e suas interações,* ***mas sempre procurar entender o sistema sob o ponto de vista de TODOS***
- *Não apenas a colaboração com o cliente,* ***mas é preciso ter o cliente sempre disposto a validar TUDO***

2 Objetivos

O Processo de Desenvolvimento de Software do TRE-AP (PDS/TRE-AP) visa estabelecer uma diretriz de alto nível para o desenvolvimento de software, que seja **orientado a pessoas**, leve em consideração as **premissas**, seja guiado pelos **valores** e coloque em prática os **princípios** expostos anteriormente neste documento.

O PDS/TRE-AP não é uma metodologia prescritiva, com roteiros explicando em detalhes o quê, como e em que ordem as coisas devem ser feitas, na forma de um conjunto fixo e imutável de regras e cerimônias a serem seguidas. Ao contrário, ele deve ser visto como uma estrutura maleável sobre a qual podem ser agregadas outras práticas que contribuam para a criação de software de qualidade e valor para o Solicitante.

A adoção do PDS/TRE-AP deve ser avaliada caso a caso e ele pode ser adaptado ao contexto de cada projeto de software. Elementos podem ser acrescentados, mesclados ou suprimidos do processo, conforme a necessidade, desde que as suas premissas, valores e princípios sejam preservados.

O PDS/TRE-AP poderá ser revisado e modificado sempre que for necessário e a sua versão mais atualizada estará disponível no site de Intranet do TRE-AP.

3 Abrangência

O PDS/TRE-AP aplica-se ao desenvolvimento de software realizado no âmbito do Tribunal Regional Eleitoral do Amapá.

4 Termos e Definições

Ambiente de Homologação: Ambiente computacional utilizado para que o Solicitante possa testar e validar o software que foi criado pela Equipe de Desenvolvimento.

Ambiente de Produção: Ambiente computacional onde o software será utilizado de forma oficial, após os testes e validação do Solicitante.

Arquitetura do sistema ou do software: Forma como o software está organizado internamente, suas estruturas mais relevantes e como se comunicam entre si.

Backlog do Produto: Lista de tudo que a Equipe de Desenvolvimento deve fazer para criar um software, conforme especificado pelo Solicitante. Essa lista é ordenada pelo valor de negócio dos seus itens, sob o ponto de vista do Solicitante.

Casos de Uso: Forma de registro e documentação dos requisitos do software.

Ciclo de desenvolvimento: Subdivisão de um projeto de desenvolvimento de software, onde um conjunto de itens do Backlog do Produto é selecionado para ser implementado.

Documento de Visão: Documento inicial de um projeto de desenvolvimento de software, que deve especificar claramente o problema que deve ser resolvido pelo software a ser criado.

Equipe auto gerenciável e auto organizável: Conjunto de pessoas capacitadas e motivadas, que assumem proativamente a responsabilidade por aquilo que precisa ser feito e se comprometem coletivamente por atingir os objetivos do projeto do qual fazem parte. Devem ter o conhecimento, a experiência e a maturidade suficientes para definir, organizar e gerenciar o seu próprio trabalho, a fim de entregar o resultado esperado ao final do projeto.

FDD: Feature Driven Development. Metodologia ágil para desenvolvimento de software, criada em 1997 por Jeff De Luca e Peter Coad.

Manifesto Ágil: Conjunto de valores e princípios que tem o propósito de instituir uma nova abordagem para o desenvolvimento de software. Foi lançado em 2001 nos Estados Unidos por um grupo de experientes e renomados profissionais da área de software.

PDS/TRE-AP: Processo de Desenvolvimento de Software do Tribunal Regional Eleitoral do Amapá.

Processo de Software ou Processo de Desenvolvimento de Software: Conjunto de atividades relacionadas que levam à produção de um software. É utilizado também com o mesmo sentido de Metodologia.

Protótipos de tela: Desenhos ou esboços das telas que constituirão o futuro sistema, feitos com o objetivo de esclarecer qual deve ser o seu correto funcionamento, tal como idealizado pelo seu Solicitante.

Requisitos do software: Características que um software deve apresentar ou funcionalidades que um software deve executar, conforme definido pelo seu Solicitante.

Solicitante: Unidade do TRE-AP, representada por um servidor ou grupo de servidores, que apresenta à STI/TRE-AP uma demanda de software relacionada a uma necessidade de negócio da sua respectiva área de atuação.

Taylorismo: Princípios criados por Frederick Winslow Taylor no início do século XX, que visava racionalizar e aumentar a produtividade dos trabalhadores da indústria de manufatura. Atualmente, os signatários do Manifesto Ágil consideram que os princípios tayloristas são opostos àqueles que devem ser utilizados por trabalhadores do conhecimento, como os desenvolvedores de software por exemplo.

User Stories ou Histórias de Usuário: Uma das formas de registro dos itens do Backlog do Produto. Não é considerado como forma de documentação de requisitos.

5 Papéis

5.1 Solicitante

O Solicitante do software a ser desenvolvido tem as seguintes responsabilidades:

- **Preencher formulário** de Solicitação de solução Informatizada, disponível no SEI e encaminhar para a Secretaria de Tecnologia da Informação;
- **Fornecer informações** para o desenvolvimento do software, como por exemplo:
  - Detalhes sobre o domínio do problema a ser tratado pelo software;
  - Detalhes sobre os processos de negócio, rotinas de trabalho, planilhas e documentos utilizados para o controle das atividades relacionadas ao domínio do problema;
  - Informar todos os potenciais usuários do sistema e áreas impactadas;
  - Possíveis dependências ou necessidade de comunicação com outros sistemas; e
  - Demais informações relevantes para o desenvolvimento do sistema.
- **Delimitar o escopo do software** a ser desenvolvido, informando à Equipe de Desenvolvimento quais funcionalidades devem fazer parte do sistema e quais não devem ser implementadas, sempre visando obter um software com o maior valor de negócio possível;
- **Decidir, em conjunto com a Equipe de Desenvolvimento, a ordem de prioridade de implementação das funcionalidades do software**, sempre buscando colocar no topo as funcionalidades de maior valor para o negócio;
- **Validar minuciosamente todo e qualquer produto de trabalho entregue pela Equipe de Desenvolvimento**, quer sejam requisitos, protótipos, software funcionando, manuais operacionais ou outros tipos de artefatos, inclusive devendo solicitar a correção ou adequação daquilo que não estiver de acordo com o esperado;
- **Participar ativamente durante todo o desenvolvimento do software solicitado**, estando sempre disponível para tirar dúvidas, explicar processos e procedimentos de trabalho, fornecer dados e informações relevantes, testar e validar todas as funcionalidades do software, entre outras atividades. **O Solicitante deve estar ciente de que a sua presença atuante e constante junto à Equipe de Desenvolvimento é fundamental para o sucesso do projeto**;

5.2 Coordenador(a) da Coordenadoria de Soluções Corporativas (CSC)

O(A) Coordenador(a) da CSC tem as seguintes responsabilidades:

- Acompanhar, orientar e supervisionar o trabalho de desenvolvimento do software solicitado;
- Dirimir dúvidas do Solicitante quanto aos procedimentos e atividades afetos à área de Tecnologia da Informação;
- Auxiliar o Solicitante, em conjunto com a Equipe de Desenvolvimento, na tomada de decisão em questões relacionadas ao desenvolvimento do software solicitado;

- Atuar e deliberar, dentro do limite da sua competência, em questões que tenham impacto sobre o andamento dos trabalhos da Equipe de Desenvolvimento, inclusive junto a outras áreas do Tribunal e/ou órgãos externos;
- Definir, juntamente com as Seções subordinadas à CSC, a Equipe de Desenvolvimento do software solicitado;

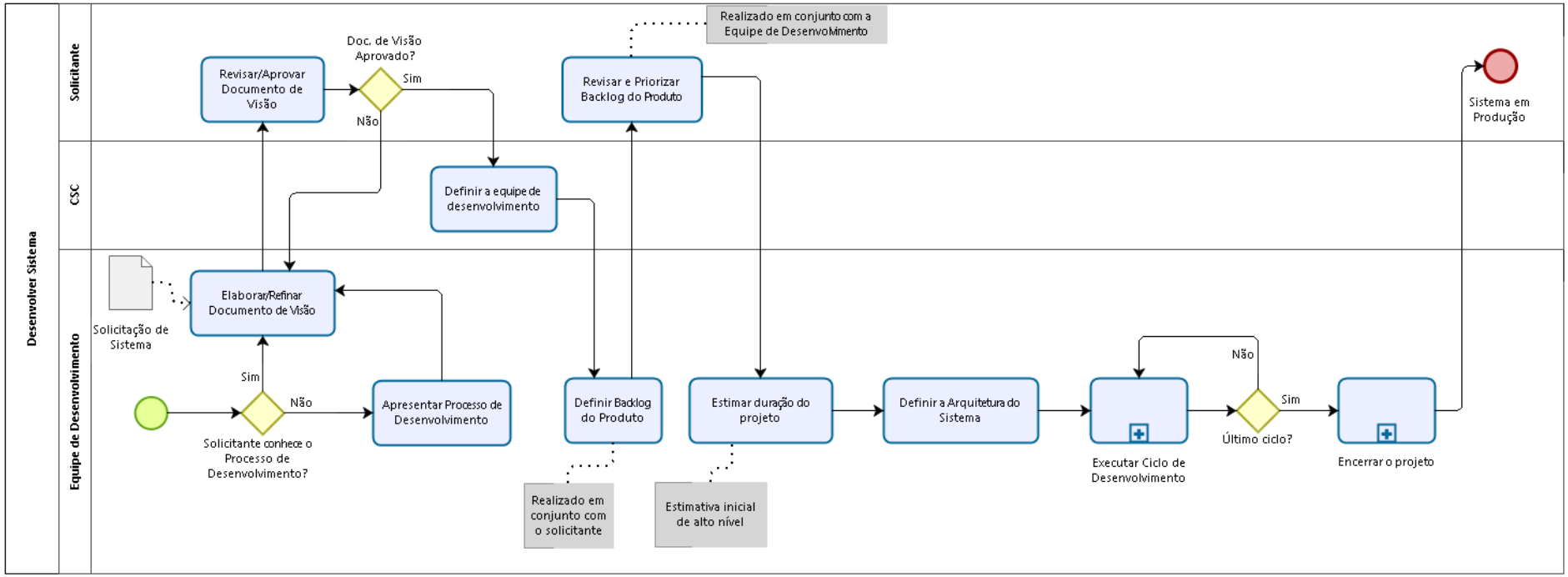
5.3 Equipe de Desenvolvimento

A Equipe de Desenvolvimento é um conjunto de pessoas com habilidades multidisciplinares, capazes de se auto-gerenciar e auto-organizar para criar um software com o máximo valor de negócio para o Solicitante.

Os membros da Equipe de Desenvolvimento são responsáveis coletivamente pelo sucesso do projeto como um todo e também por:

- Fazer o levantamento, análise e registro dos requisitos do software;
- Criar protótipos do sistema para validação dos requisitos junto ao Solicitante;
- Definir ou adaptar a arquitetura de software;
- Executar as atividades do desenvolvimento do software, de acordo com padrões e convenções existentes;
- Tomar decisões técnicas que orientam todo o design e a implementação do projeto;
- Prezar pela qualidade do código-fonte;
- Realizar testes de unidade, de integração, de sistema e de aceitação operacional;
- Analisar, modelar e criar a estrutura do banco de dados, de acordo com padrões e convenções existentes;
- Criar as interfaces solicitadas pelo usuário;
- Zelar pela integridade e desempenho do banco de dados;

6 Visão Geral do Processo



6.1 Elementos do Processo

6.1.1 ☐Apresentar Processo de Desenvolvimento

Descrição

Caso o Solicitante não conheça o PDS/TRE-AP, deve-se fazer uma breve explicação do seu fluxo de trabalho e apresentar as responsabilidades de cada parte envolvida nesse processo.

Executantes

Coordenador(a) da CSC, Equipe de Desenvolvimento

6.1.2 ☐Elaborar/Refinar Documento de Visão

Descrição

Realizar reuniões com o Solicitante a fim de elaborar o Documento de Visão.

O objetivo do Documento de Visão é deixar claro o motivo pelo qual o software será criado e estabelecer expectativas comuns entre todas as partes interessadas acerca das suas principais características e funcionalidades.

Em última análise, o Documento de Visão deve ser capaz de responder às seguintes perguntas:

- Quais problemas esse software pretende resolver?
- Quem será impactado, dentro e fora da Instituição, após a criação e implantação desse software?
- Como o trabalho do Solicitante será diferente após a criação desse software?
- Quais são as funcionalidades essenciais desse software, sob o ponto de vista do Solicitante?
- Quais são as funcionalidades desejáveis, mas não essenciais desse software, sob o ponto de vista do Solicitante?
- Como esse software contribui para os objetivos da Instituição?
- Qual o benefício maior para a Instituição e para a sociedade com a utilização desse software?
- Qual a motivação por trás do desenvolvimento desse software?

O Documento de Visão é o documento de planejamento mais importante do software a ser construído, pois ele sintetiza o objetivo de alto nível do produto e as razões pelas quais ele deve ser criado, servindo como guia para o rumo que o projeto deve tomar ao longo de todo o seu curso até a entrega do produto final, uma vez que todo o trabalho de desenvolvimento de software deriva dele.

A elaboração do Documento de Visão deve observar as seguintes diretrizes:

- Deve ser um esforço conjunto entre o Solicitante e a Equipe de Desenvolvimento;
- O Documento de Visão deve ser escrito na linguagem de negócio do Solicitante e não deve conter termos técnicos da área de TI;
- O Documento de Visão, sempre que for possível, deve conter uma frase que expresse, de forma simples e clara, o objetivo de alto nível do software a ser criado. Essa frase geralmente é conhecida como a sua "Declaração de Visão";

- O tamanho e formalismo do Documento de Visão dependerá muito do tamanho e criticidade do software a ser construído. Projetos simples e de baixa criticidade, por exemplo, podem ter esse documento resumidos a uma "Declaração de Visão", publicada em um site de Wiki interno;

O detalhamento das funcionalidades do sistema, restrições e regras de negócio constarão nos itens do Backlog do Produto (user stories, casos de uso, itens de trabalho, features e etc.).

Modelos e exemplos de Documento de Visão podem ser encontrados na seguinte página: http://stiwiki.tre-ap.gov.br:8080/stiwiki/gestao/processo-de-desenvolvimento-de-software/documento-de-visao

Executantes

Equipe de Desenvolvimento, Solicitante

6.1.3 ☐Revisar/Aprovar Documento de Visão

Descrição

Revisar o Documento de Visão e garantir que ele atenda aos requisitos descritos na atividade "Elaborar/Refinar Documento de Visão".

O Solicitante poderá aprová-lo ou enviá-lo de volta para que a Equipe de Desenvolvimento faça as correções/alterações necessárias.

Executantes

Solicitante

6.1.4 ☐Definir a equipe de desenvolvimento

Descrição

O(A) Coordenador(a) da CSC, em conjunto com as suas Seções subordinadas, irá definir quem irá compor a Equipe de Desenvolvimento, levando em conta questões como criticidade e natureza do projeto, disponibilidade da equipe, projetos em andamento, tecnologias utilizadas, entre outros fatores.

Executantes

Coordenador(a) da CSC, Equipe de Desenvolvimento

6.1.5 ☐Definir Backlog do Produto

Descrição

O Backlog do Produto é a lista de tudo que a Equipe de Desenvolvimento deve fazer para atingir os objetivos estabelecidos no Documento de Visão. Dessa forma, pode-se afirmar que os itens que compõem o Backlog do Produto devem ser derivados do Documento de Visão.

Os itens do Backlog do Produto são geralmente requisitos funcionais do sistema e podem estar descritos em qualquer formato que a Equipe de Desenvolvimento se sinta confortável para trabalhar e que possa ser compreendida pelo Solicitante. Esses itens do Backlog podem ser também bugs a serem corrigidos ou documentação para usuários finais, por exemplo. **O importante é que o Solicitante possa enxergar valor de negócio nos itens do Backlog do Produto**.

Exemplos de tipos de itens que podem compor o Backlog do Produto: User Stories, Features no formato A.R.O. (Ação, Resultado, Objeto) do FDD, Casos de Uso, bugs a serem corrigidos, roteiros e manuais para usuários finais e etc.

O Backlog do Produto é uma lista dinâmica que é ordenada pelo seu valor para o Solicitante, onde os itens mais críticos para o sucesso do negócio deverão estar no topo e devem ser atacados primeiro.

Em um dado momento do projeto, os itens mais prioritários do Backlog do Produto deverão estar em um nível maior de detalhes, de tal forma que possibilite a sua implementação, ao passo que os itens mais abaixo poderão estar descritos de forma mais vaga e com menos detalhes. A ideia é que os itens somente sejam detalhados no Backlog do Produto quando estiverem prestes a serem desenvolvidos.

O Backlog do Produto inicial pode ser longo, contendo desde itens pequenos e bem detalhados até itens grandes e vagos. Mas ele também pode ser curto e conter apenas uma quantidade de itens necessária para se iniciar o desenvolvimento.

O Backlog do Produto evoluirá ao longo de todo o projeto e será frequentemente modificado com a adição, subtração, detalhamento, divisão, reordenamento e modificação de seus itens.

Os itens do Backlog do Produto que forem mais críticos para o sucesso do projeto ou que forem mais complexos poderão ser complementados com os **critérios de aceitação**, que são condições de verificação pré-definidas que visam garantir a sua correta implementação. Os critérios de aceitação servem também como guia para que a Equipe de Desenvolvimento possa saber quando a implementação do item foi concluída.

Para mais detalhes sobre critérios de aceitação, favor consultar a seguinte página: http://stiwiki.tre-ap.gov.br:8080/stiwiki/gestao/processo-de-desenvolvimento-de-software/criterios-de-aceitacao

Recomenda-se a utilização de ferramenta eletrônica para o registro e acompanhamento do Backlog do Produto, que esteja disponível via web, que facilite o trabalho colaborativo e que seja de uso simples e prático. Um exemplo de ferramenta com essas características é o **Trello** (http://trello.com).

Para maiores informações sobre a composição do Backlog do Produto, favor consultar a seguinte página: http://stiwiki.tre-ap.gov.br:8080/stiwiki/gestao/processo-de-desenvolvimento-de-software/backlog-do-produto

Executantes

Equipe de Desenvolvimento, Solicitante

6.1.6 ☐Revisar e Priorizar Backlog do Produto

Descrição

O Solicitante e a Equipe de Desenvolvimento deverão fazer o seguinte com relação ao Backlog do Produto:

- Revisar cada item do Backlog do Produto para garantir que estejam de acordo com os objetivos do projeto, estabelecidos no Documento de Visão. Além disso, os itens devem estar em linguagem clara, objetiva e sem ambiguidades;
- Ordenar os itens do Backlog do Produto de tal forma que os itens de maior valor para o negócio e mais críticos para o sucesso do projeto estejam sempre no topo da lista;
- Verificar se os itens do Backlog do Produto que são mais críticos ou mais complexos tiveram os seus critérios de aceitação corretamente definidos. Pode ser verificado também se algum item precisa ter os seus critérios de aceitação definidos.

Executantes

Equipe de Desenvolvimento, Solicitante

6.1.7 Estimar duração do projeto

Descrição

A Equipe de Desenvolvimento, com base nos itens do Backlog do Produto, fornece uma estimativa inicial de alto nível para a duração do projeto. Essa estimativa deverá ser revista e ajustada no decorrer do projeto, na medida em que o desenvolvimento do software avançar.

Entretanto, o Solicitante e os níveis hierárquicos superiores devem ter a plena ciência de que, **em estágios iniciais de um projeto de desenvolvimento de software, é impossível estimar a sua duração com o mínimo de precisão**.

Isso decore de vários fatores, entre eles:

- Durante o desenvolvimento de qualquer software é normal que novas ideias possam emergir como consequência do contato do Solicitante e da própria Equipe de Desenvolvimento com protótipos ou versões preliminares do sistema;
- O Solicitante ou a Equipe de Desenvolvimento podem se deparar com situações não previstas no início do projeto, relacionadas à tecnologia, ao funcionamento do próprio sistema, a mudanças de circunstâncias externas ao projeto ou à descoberta de fatores que não eram do conhecimento de nenhuma das partes interessadas em estágios anteriores do projeto;
- A capacidade de comunicação entre o Solicitante e a Equipe de Desenvolvimento tem grande impacto no andamento do projeto. O Solicitante deve conseguir transmitir à Equipe de Desenvolvimento, de forma clara e objetiva, os detalhes que realmente serão decisivos para o sucesso do projeto. A Equipe de Desenvolvimento, por sua vez, deve ser capaz de interpretar de forma precisa o que lhes é transmitido, fazendo as perguntas certas e validando junto ao Solicitante toda e qualquer hipótese levantada;

Mais importante do que saber se um projeto de software está aquém ou além do seu cronograma, é saber se ele está entregando o máximo de valor de negócio para o Solicitante e para a Instituição como um todo, com o menor custo de manutenção possível.

Executantes

Equipe de Desenvolvimento

6.1.8 Definir a Arquitetura do Sistema

Descrição

A Equipe de Desenvolvimento, com base no Documento de Visão, na natureza do projeto e em informações prestadas pelo Solicitante, elege a arquitetura de software (Anexo 1) que deve servir de base para o desenvolvimento da solução proposta, levando em consideração, se for o caso, os padrões de implementação pré-definidos da Instituição e seguir critérios de Segurança estabelecidos no normativo próprio, com foco na Segurança da Infraestrutura e dos Dados Pessoais e nas demais normas de Segurança da Informação do TRE-AP.

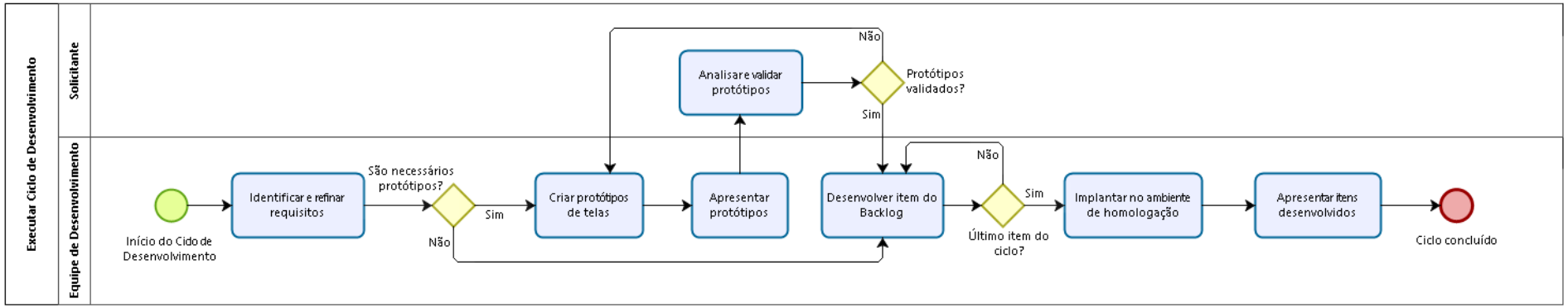
Executantes

Equipe de Desenvolvimento

6.1.9 Executar Ciclo de Desenvolvimento (sub-processo)

6.1.10 Encerrar o projeto (sub-processo)

7 Executar Ciclo de Desenvolvimento (sub-processo)



7.1 Elementos do Processo

7.1.1 ❑Identificar e refinar requisitos

Descrição

A Equipe de Desenvolvimento, com o auxílio do Solicitante, seleciona os itens do Backlog do Produto mais prioritários que serão implementados no atual ciclo de desenvolvimento e, a partir deles, quaisquer das seguintes tarefas poderão ser realizadas, conforme a necessidade:

• Verificar necessidade de maior esclarecimento e/ou detalhamento acerca dos itens do Backlog;

• Verificar necessidade da definição de critérios de aceitação para os itens do Backlog;

• Verificar necessidade de divisão ou união de itens do Backlog para facilitar sua implementação; e

• Verificar quais itens do Backlog necessitarão de posterior criação de protótipos de telas.

Executantes

Equipe de Desenvolvimento, Solicitante

7.1.2 ❑Criar protótipos de telas

Descrição

Criar protótipos de telas para os itens do Backlog mais críticos, com o propósito de validar junto ao Solicitante as hipóteses levantadas pela Equipe de Desenvolvimento durante a análise dos requisitos.

Para a criação desses protótipos, podem ser utilizadas as ferramentas de desenho que a Equipe de Desenvolvimento se sentir mais confortável. Um exemplo de ferramenta desse tipo é o Pencil (https://pencil.evolus.vn).

Executantes

Equipe de Desenvolvimento

7.1.3 ❑Apresentar protótipos

Descrição

A Equipe de Desenvolvimento apresenta os protótipos de telas ao Solicitante e descreve os seus elementos constituintes, forma de uso, fluxo de trabalho, entradas e saídas esperadas, validações e outros detalhes pertinentes.

Executantes

Equipe de Desenvolvimento, Solicitante

7.1.4 ❑Analisar e validar protótipos

Descrição

O Solicitante deve analisar e validar cada um dos protótipos de tela criados, informando à Equipe de Desenvolvimento se há necessidade de correções e ajustes.

Executantes

Solicitante

7.1.5 ☐Desenvolver item do Backlog

Descrição

A Equipe de Desenvolvimento, com base nas informações prestadas pelo Solicitante, nos critérios de aceitação e nos protótipos criados, procede com o design, implementação e testes do item do Backlog, em conformidade com os padrões de arquitetura, de implementação, de modelagem de objetos e dados pré-estabelecidos, caso existam.

É importante ressaltar que a descrição registrada no item de Backlog não deve ser vista como fonte de informação ou de documentação para subsidiar a sua implementação. O item do Backlog deve ser visto como um lembrete para uma conversa que deverá haver entre a Equipe de Desenvolvimento e o Solicitante antes da sua implementação, onde serão discutidos e esclarecidos seus detalhes e, se for o caso, os seus critérios de aceitação deverão ser definidos.

Executantes

Equipe de Desenvolvimento

7.1.6 ☐Implantar no ambiente de homologação

Descrição

Realizar a implantação do software no ambiente de homologação, com os itens do Backlog que foram implementados durante o ciclo de desenvolvimento.

Executantes

Equipe de Desenvolvimento

7.1.7 ☐Apresentar itens desenvolvidos

Descrição

A Equipe de Desenvolvimento apresenta ao Solicitante o software com as funcionalidades que foram implementadas durante o ciclo de desenvolvimento. Os itens que ficaram pendentes ou que não foram implementados também devem ser informados ao Solicitante.

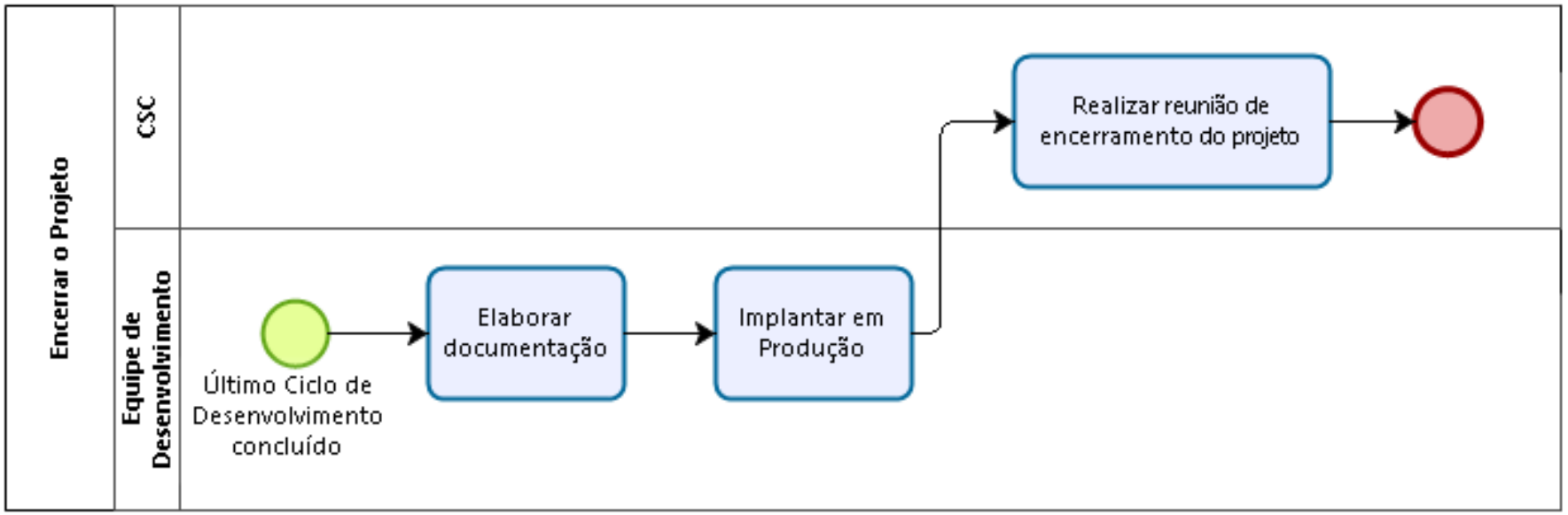
O Solicitante deve fazer uma análise minuciosa de tudo que foi desenvolvido, inclusive fazendo a validação das funcionalidades com os critérios de aceitação previamente definidos, quando for o caso. O Solicitante pode solicitar um tempo adicional para realizar a validação do que foi implementado, conforme a necessidade.

Caso sejam encontradas inconformidades ou erros, esses devem ser registrados e adicionados ao Backlog do Produto para posterior priorização, de acordo com a sua importância e criticidade.

Executantes

Equipe de Desenvolvimento, Solicitante

8 Encerrar o projeto (sub-processo)



8.1 Elementos do Processo

8.1.1 ☐Elaborar documentação

Descrição

A Equipe de Desenvolvimento deverá elaborar as seguintes documentações:

- Página no Wiki com informações abrangentes sobre o sistema, visando subsidiar as áreas técnicas nas atividades de suporte aos usuários finais e na garantia da disponibilidade do sistema; e
- Manual de Operação do sistema para os usuários finais, que deverá ser validado pelo Solicitante.

Executantes

Equipe de Desenvolvimento

8.1.2 ☐Implantar em Produção

Descrição

A Equipe de Desenvolvimento procederá com todas as tarefas relacionadas à implantação do software em ambiente de produção, entre as quais, mas não somente:

- Criação da base de dados do sistema no ambiente de produção;
- Implantação do sistema no servidor de aplicação de produção; e
- Atualização e/ou criação de links e/ou páginas na Intranet e/ou Internet para acesso ao sistema, conforme o caso.

Executantes

Equipe de Desenvolvimento

8.1.3 ☐Realizar reunião de encerramento do projeto

Descrição

Fazer avaliação geral do projeto, registrando as lições aprendidas, como por exemplo:

- Experiências bem-sucedidas durante o projeto;
- Principais problemas encontrados durante o projeto e o que foi feito para resolvê-los;
- Ocorrências que estavam fora do controle das partes envolvidas, mas que tiveram impacto no projeto e a probabilidade de reincidência em projetos futuros;

Preencher o Termo de Encerramento do Projeto;

Fazer documento de aceitação do software, a ser assinado pelo gestor Negocial do Sistema.

Executantes

Coordenador(a) da CSC, Equipe de Desenvolvimento

9 Artefatos

Ordem	***Artefato***	***Modelo***
1	Documento de Visão	Anexo 2
2	Lista de Tarefas	Feito na ferramenta Trello
3	Termo de Encerramento do Projeto	Disponível no SEI
4	Termo de Aceite de Software	Anexo 3

10 Referências

BASTOS, L. ***Da descoberta do Ágil ao Manifesto Luca Bastos***. AGILE Brazil 2013. Disponível em: <https://vimeo.com/82462105>. Acesso em: 27/11/2017.

BECK, K.; BEEDLE, M.; BENNEKUM, A.; COCKBURN, A.; et al. ***Manifesto for Agile Software Development***. Disponível em <http://agilemanifesto.org>, 2001. Acesso em: 27/11/2017.

BROOKS, Frederick P. ***O Mítico Homem-Mês*** - *Ensaios sobre Engenharia de Software*. 1ª ed. São Paulo: Elsevier, 2009.

COCKBURN, Alistair. In: ***Characterizing people as non-linear, first-order components in software development***. Disponível em: <http://alistair.cockburn.us/Characterizing+people+as+non-linear%2c+first-order+components+in+software+development>, 1999. Acesso em: 27/11/2017.

COCKBURN, Alistair. ***Agile software development***: *The Cooperative Game*. 2ª ed. Boston: Pearson Education, 2006.

DEMARCO, Tom; LISTER, Timothy R. ***Peopleware***: *productive projects and teams*. 3ª ed. New York: Dorset House, 2013.

DE MASI, Domenico. ***O Ócio Criativo***. 10ª ed. Rio de Janeiro: Sextante, 2000.

ELSSAMADISY, Amr. ***Agile Adoption Patterns*** - *A Roadmap to Organizational Success*. 1ª ed. Boston: Pearson Education, 2009.

FOWLER, Martin. In: ***The New Methodology***. Disponível em: <https://www.martinfowler.com/articles/newMethodology.html>, 2005. Acesso em: 27/11/2017.

LARMAN, Craig. ***Agile & Iterative Development*** - *A Manager's Guide*. 1ª ed. Boston: Pearson Education, 2004.

MARTIN, R.; MCBREEN, P. In: ***Manifesto for Software Craftsmanship***. Disponível em: <http://manifesto.softwarecraftsmanship.org/#/pt-br>, 2009. Acesso em 27/11/2017.

SCHWABER, Ken. ***Agile Project Management with Scrum***. 1ª ed. Redmond: Microsoft Press, 2004.

SOMMERVILLE, Ian. ***Engenharia de Software***. 9ª ed. São Paulo: Pearson Education do Brasil, 2011.

TAYLORISMO. In: ***WIKIPÉDIA***, a enciclopédia livre. Disponível em: <https://pt.wikipedia.org/w/index.php?title=Taylorismo&oldid=50153132#Organização_racional_do_trabalho>. Acesso em: 27/11/2017.

TELES, Vinícius M. "***Um estudo de caso da adoção das práticas e valores do Extreme Programming***". Dissertação de Mestrado, Núcleo de Computação Eletrônica, UFRJ, 2005.

ANEXO I

Arquitetura de Software de Referência para Sistemas Corporativos do TRE-AP

Controle de Versões

Versão	***Data***	***Responsável***	***Descrição***
1.0	*15/12/2015*	*Davi Shibayamas (SDS/CSCI)*	*Versão inicial*
1.1	*30/03/2022*	*Elinete Freitas (CSC/STI)*	*Revisão: reestruturação do documento e adequação ao Processo de Desenvolvimento de Software do TRE-AP.*

1. Introdução

Arquitetura de Software é o conjunto de decisões significativas acerca da organização de um sistema de software, da seleção dos seus elementos estruturais, suas interfaces e comportamento (BOOCH, RUMBAUGH e JACOBSON, 2006, p. 34).

Este documento visa definir e documentar uma arquitetura de software de referência a ser utilizada em projetos de desenvolvimento de sistemas informatizados no âmbito do Tribunal Regional Eleitoral do Amapá.

2. Objetivos

A arquitetura de software de referência tem o objetivo de fornecer um ponto de partida padronizado para a construção de sistemas corporativos, de tal maneira que, ao se iniciar o desenvolvimento de um novo software, já se tenha uma arquitetura de sistema pré-definida e uma infraestrutura básica que disponibilize as implementações dos requisitos não-funcionais mais comuns de uma aplicação, tais como armazenamento e persistência de dados, segurança, controle transparente de transações e exceções, auditoria, configuração, geração de relatórios e etc.

A arquitetura proposta também deve ser flexível o suficiente para atender as necessidades específicas de cada projeto, sendo capaz de suportar, sem maiores dificuldades, tanto a incorporação de novos elementos como também a eliminação ou adaptação de elementos já existentes.

A definição da arquitetura de referência descrita neste documento abrange os seguintes aspectos:

- Definição de uma arquitetura em camadas
- Diretrizes gerais para a composição e comunicação entre as camadas que compõem a arquitetura
- Tecnologias e frameworks a serem utilizados em cada camada da arquitetura
- Linguagens de programação, ferramentas e softwares de apoio a serem utilizados
- Definição de padrões de implementação para os requisitos não funcionais mais comuns em sistemascorporativos

3. Arquitetura em camadas

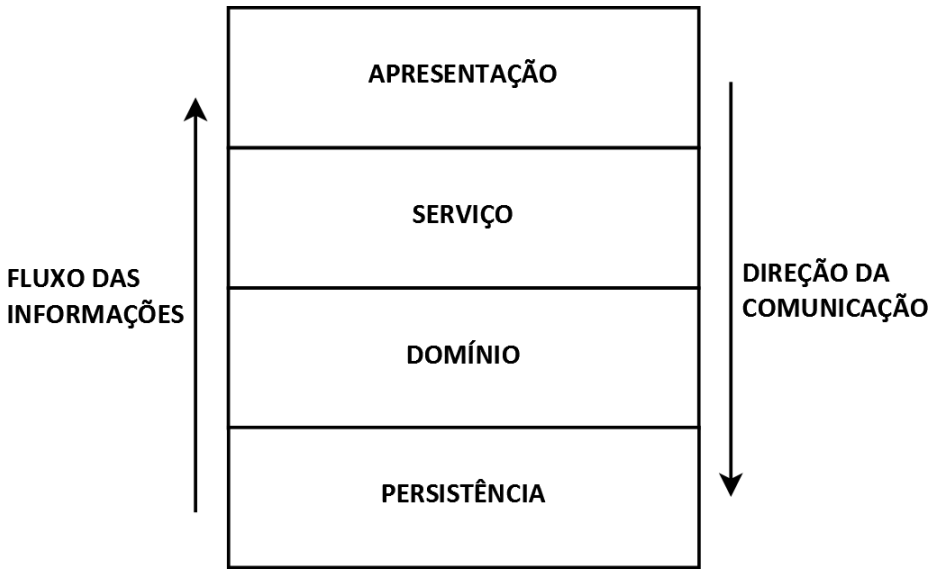
O padrão arquitetural mais amplamente aceito para organizar sistemas de software é a **arquitetura em camadas**. Nessa abordagem, cada camada é responsável por tratar um aspecto específico do sistema (armazenamento e recuperação de dados, apresentação, regras de negócio, etc) e cada camada, idealmente, só deve se comunicar com a camada imediatamente abaixo dela.

Segundo Buschmann (1996, p. 31), "o padrão arquitetural em camadas ajuda a estruturar aplicações que podem ser decompostas em grupos de subtarefas nas quais cada um desses grupos está em um nível particular de abstração".

Os principais benefícios da utilização de uma arquitetura em camadas são os seguintes:

- Facilita a compreensão dos elementos que compõem o software, através da **separação de responsabilidades** e,consequentemente, facilita a manutenção e evolução da base de código.
- Aumenta a **coesão** do software, uma vez que cada camada trata apenas de um aspecto específico do sistema, podendo ser mantida e evoluída isoladamente com facilidade.
- Diminui o **acoplamento** entre os elementos do software, já que uma camada não conhece os detalhes internosde outras camadas, mas apenas solicita os serviços necessários através de mensagens bem definidas.
- Isola as partes do código que não tem relação entre si, diminuindo ou eliminando eventuais efeitos colateraisdecorrentes de alterações evolutivas ou corretivas no software.

A arquitetura de referência está organizada em camadas, a saber, **Camada de Persistência**, **Camada deDomínio, Camada de Serviço** e **Camada de Apresentação**, conforme ilustrado no diagrama abaixo.



3.1. Camada de Persistência

3.1.1. Responsabilidades e diretrizes de implementação

A camada de persistência é responsável por intermediar as chamadas para o serviço de armazenamento e recuperação de dados que, geralmente, é executado por um Sistema de Gerenciamento de Banco de Dados Relacional (SGBDR).

Essa camada deve concentrar todo o código específico de manipulação de dados. Para os casos em que o sistema se comunica diretamente com o SGBDR, esses códigos serão chamadas na linguagem estruturada de consultarelacional (SQL) ou chamadas para procedimentos armazenados. Para os casos em que é utilizado um framework de mapeamento objeto-relacional, esses códigos serão constituídos basicamente por chamadas para a API do framework em questão e/ou para a sua linguagem de consulta específica, caso disponível.

3.1.2. Tecnologias utilizadas

Na arquitetura de referência, quem faz o papel da camada de persistência, na prática, é o framework de mapeamento objeto-relacional do Java, o **JPA** (Java Persistence API).

Embora o JPA não faça parte diretamente da arquitetura, as classes de serviço da camada de aplicação, principalmente a classe **`CrudService`**, referenciam as duas principais interfaces do JPA: **`EntityManager`** e **`Query`**.

3.2. Camada de Domínio

3.2.1. Responsabilidades e diretrizes de implementação

A camada de domínio representa os conceitos, regras e lógicas do negócio e é constituída pelas classes do domínio que representam esses elementos.

Por conta da utilização do JPA, as classes da camada de domínio devem conter metadados com informações de mapeamento objeto-relacional, na forma de *annotations* do Java específicos desse framework de persistência.

Essa camada é considerada o coração de qualquer sistema de software, pois é nela que são mantidos os conceitos centrais do domínio do problema. Deve ser dispensada, portanto, especial atenção na concepção, análise e modelagem das classes que formam essa camada, sempre visando alcançar o maior nível possível de coesão e, ao mesmo tempo, evitando ao máximo o acoplamento entre seus elementos constituintes.

Sempre que possível, também, deve-se projetar as classes da camada de domínio de modo a evitar o anti- padrão que Martin Fowler (FOWLER, 2003) convencionou chamar de **Modelo de Domínio Anêmico**, onde as classes de domínio não implementam efetivamente as regras de negócio, pois não possuem comportamento nem responsabilidades, mas apenas são repositórios de atributos. Nesse cenário indesejável, outras classes, não pertencentes ao domínio, manipulam os atributos das classes de domínio para executar ações e implementar as regras do domínio do sistema.

3.2.2. Tecnologias utilizadas

As classes da camada de domínio devem ser objetos Java simples, também conhecidos como POJOs (FOWLER, 2001), sem dependências de tecnologias externas. A única exceção são as anotações do JPA responsáveis pelo mapeamento objeto-relacional que, nesse caso, são mais convenientes do que o seu correspondente em arquivo XML.

3.3. Camada de Serviço

3.3.1. Responsabilidades e diretrizes de implementação

Idealmente, a camada de serviço não deve possuir nenhuma lógica de regra de negócio, mas deve ter o papel de coordenar o trabalho que é solicitado a ela pela camada de apresentação, delegando o processamento das regras de negócio à camada de domínio do sistema.

3.3.2. Tecnologias utilizadas

A camada de serviço faz uso do Spring Framework, que é um framework de inversão de controle através da injeção de dependência, mas também tem um vasto conjunto de funcionalidades como controle transparente de transações, concorrência, manipulação de exceções, programação orientada a aspectos, entre muitas outras características.

3.4. Camada de Apresentação

3.4.1. Responsabilidades e diretrizes de implementação

A camada de apresentação é responsável por tratar os eventos de entrada do usuário, delegar essas requisições para a camada de serviço e também exibir apropriadamente as informações recebidas da camada de serviço para os usuários.

As classes da camada de apresentação **NÃO DEVEM CONTER LÓGICA DO DOMÍNIO DO SISTEMA**, mas sim delegar o que precisa ser feito para a camada de serviço, de forma a preservar a

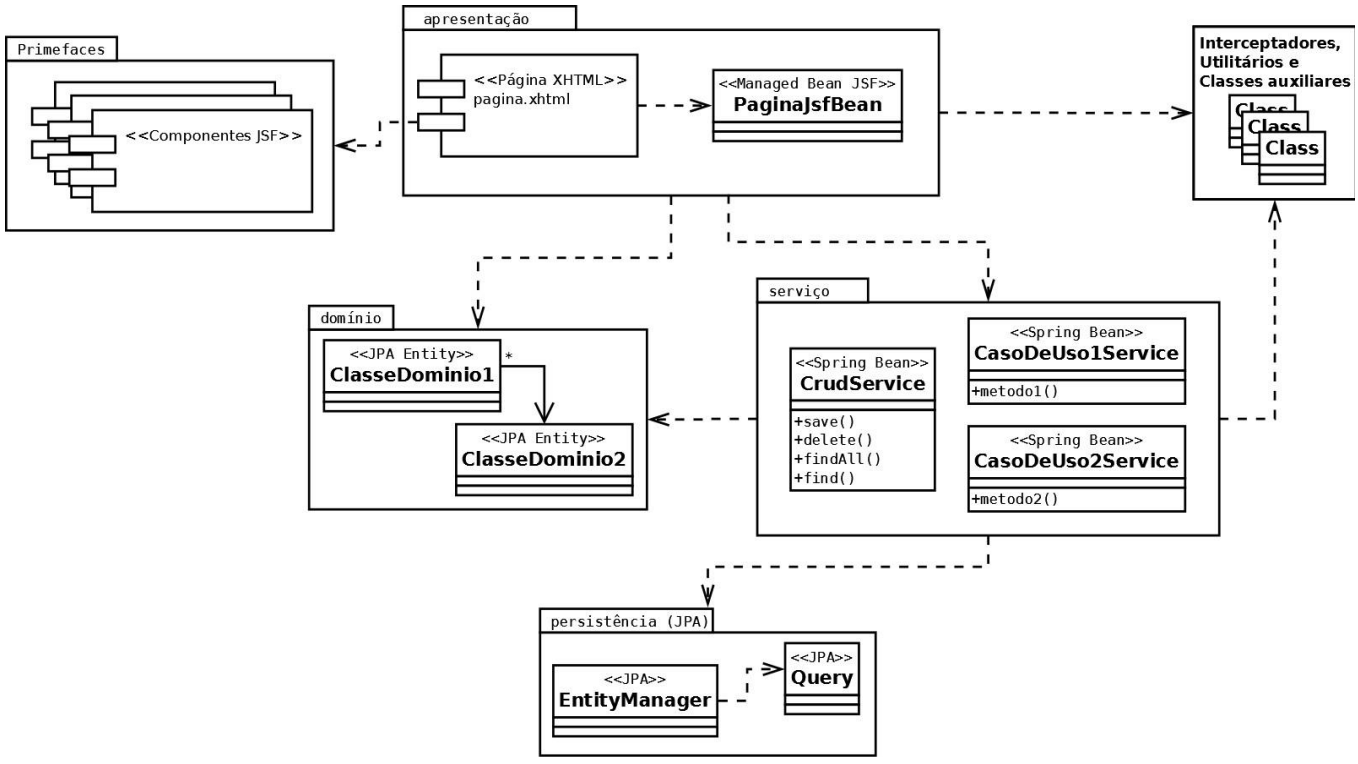
separação de responsabilidades (DIJKSTRA, 1982), princípio elementar da engenharia de software que garante a clareza e a facilidade de manutenção do sistema.

3.4.2. Tecnologias utilizadas

Na camada de apresentação é utilizado o framework baseado em componentes Java ServerFaces (JSF) 2.xcomo tecnologia de apresentação, juntamente com a biblioteca de componentes Primefaces.

4. Visão geral da arquitetura

No diagrama de pacotes da UML abaixo está representada uma visão geral, de alto nível, da arquitetura dereferência.



5. Sistemas desenvolvidos

Os sistemas em funcionamento no TRE-AP e que foram desenvolvidos pela Coordenadoria de Soluções Corporativas (CSC/STI) utilizando a arquitetura de referência proposta neste documento estão

disponíveis na intranet no link: http://supermariobros.tre-ap.jus.br:10100/apex/sistemastreap/r/114/login_desktop.

6. Referências

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. UML Guia do Usuário. 2ª.ed. Rio de Janeiro: Elsevier, 2006.

BUSCHMANN, F. et al. Pattern-Oriented Software Architecture - A System of Patterns. London: Wiley, 1996.

FOWLER, Martin. Anemic Domain Model, 2003. Disponível em: <http://martinfowler.com/bliki/AnemicDomainModel.html>. Acesso em: 15 de dez. 2015.

FOWLER, Martin. POJO, 2001. Disponível em: <http://www.martinfowler.com/bliki/POJO.html>. Acesso em: 15 dedez. 2015.

DIJKSTRA, Edsger. On the role of scientific thought, 1982. Disponível em: <http://www.cs.utexas.edu/users/EWD/transcriptions/EWD04xx/EWD447.html>. Acesso em: 18 de dez. 2015.

ANEXO II

Documento de Visão

Tribunal Regional Eleitoral do Amapá
Secretaria de Tecnologia da Informação

DOCUMENTO DE VISÃO

1. IDENTIFICAÇÃO DO SISTEMA

1.1 Nome do Sistema:

1.2 Data do Documento:

2. OBJETIVOS DO DOCUMENTO

O objetivo deste documento é deixar claro o motivo pelo qual o sistema informatizado em questão será desenvolvido. Ele também busca estabelecer expectativas comuns entre as partes interessadas e a equipe técnica de desenvolvimento no que diz respeito às características e funcionalidades que devem ser implementadas no sistema.

O detalhamento dessas funcionalidades, assim como: restrições e regras de negócio, constarão, se necessário, em documentos suplementares.

3. DEFINICÃO DO PROBLEMA

Problema	
Quem é afetado	
Impacto	
Uma solução ideal seria	

4. PARTES INTERESSADAS

Nome	Descrição	Resumo das Responsabilidades

5. VISÃO GERAL DO SISTEMA

5.1 Necessidades e Funcionalidades

Necessidade do Solicitante	Prioridade	Funcionalidade do Sistema	Usuário no Sistema

7. APROVACÃO

Estou ciente e de acordo com as informações passadas por mim para composição deste documento e com o conteúdo deste.

Data da Aprovação:

Papel	Nome	Assinatura
Solicitante		
Gestor negocial		
Coordenador CSC		
Desenvolvedor		

ANEXO III

Termo de Aceite de Sistema

Tribunal Regional Eleitoral do Amapá
Secretaria de Tecnologia da Informação

TERMO DE ACEITE DE SISTEMA

8. IDENTIFICACÃO DO SISTEMA

1.1 Nome do Sistema:

1.2 Data da entrega:

9. OBJETIVOS DO DOCUMENTO

Este documento detalha e formaliza a tomada de visão e o aceite da entrega dos requisitos propostos no sistema, segundo conformidade com os requisitos e os critérios de aceitação definidos.

2. DETALHES DA ENTREGA

Documente neste item os itens que estão sendo contemplados pelo projeto, caso haja requisitos ou solicitações pendentes descreva-os de forma criteriosa e detalhada, se possível liste-os na tabela abaixo.

PENDÊNCIA	RESPONSÁVEL	ENTREGA

3. ACEITE FINAL DA ENTREGA DO SISTEMA

Todos os envolvidos diretamente ou de forma secundária, que encontram-se listados abaixo, ao assinarem este documento declaram o entendimento de todos os itens entregues e sua conformidade com os critérios de aceitação previstos no documento de visão do sistema.

PAPEL	NOME	ASSINATURA
SOLICITANTE		
GESTOR NEGOCIAL		
COORDENADOR		
DESENVOLVEDOR		