**FACULDADE IMP**

**Curso de Pós-Graduação Lato Sensu em Gestão da Tecnologia da Informação na Administração Pública**

# UMA VISÃO CRÍTICA SOBRE O PROCESSO DE DESENVOLVIMENTO DE SOFTWARE DO MPDFT

**MARCELO HENRIQUE DE OLIVEIRA LIMA**

**BRASÍLIA**
**2014**

**FACULDADE IMP**

**Curso de Pós-Graduação Lato Sensu em Gestão da Tecnologia da Informação na Administração Pública**

# UMA VISÃO CRÍTICA SOBRE O PROCESSO DE DESENVOLVIMENTO DE SOFTWARE DO MPDFT

Trabalho de conclusão de Curso de Especialização apresentado como requisito parcial para a obtenção de titulo de Especialista em Gestão da Tecnologia da Informação na Administração Pública pelo Curso de Especialização Lato Sensu da Faculdade IMP.

**Orientador: Prof. Dr. Renato Jorge Brown Ribeiro**

**Brasília**
**2014**

**MARCELO HENRIQUE DE OLIVEIRA LIMA**

# UMA VISÃO CRÍTICA SOBRE O PROCESSO DE DESENVOLVIMENTO DE SOFTWARE DO MPDFT

Artigo apresentado como requisito parcial para obtenção de título de Especialista, pelo Programa de Pós-Graduação em Gestão da Tecnologia da Informação na Administração Pública da Faculdade IMP.

**Aprovado em:** _____/_____/_____

**Banca Examinadora:**

---

**Prof. Dr. Renato Jorge Brown Ribeiro**

# Uma Visão Crítica sobre o Processo de Desenvolvimento de Software do MPDFT

Marcelo Henrique de Oliveira Lima

2014

## Resumo

Os processos de desenvolvimento de software estão em constante evolução e, geralmente, trazem melhorias para o dia a dia dos envolvidos com essa atividade. As melhorias entram nos processos de software das organizações através de avaliações periódicas dos seus modelos. Com o mundo ágil ganhando cada vez mais espaço, modelos considerados pesados precisam se adaptar às novas abordagens para continuarem competitivos. O processo de desenvolvimento do Ministério Público do Distrito Federal e Territórios não é diferente e este artigo faz justamente isso, uma avaliação do processo atual. Problemas são detectados e sugestões são feitas para eliminar ou minimizar esses problemas.

**Palavras-chaves**: Processo Unificado. MPDFT. Métodos Ágeis.

## Introdução

Desde que Gordon Earle Moore, co-fundador e presidente da Intel Corporation, profetizou que o número de transistores dos chips dobraria pelo mesmo custo a cada dezoito meses vemos uma revolução tecnológica. Atualmente, quase tudo está ligado à tecnologia, que faz o mundo se mover cada vez mais rápido. Isso não muda para o ambiente de desenvolvimento de software. Essa área vem se desenvolvendo e se estruturando desde meados dos anos setenta quando a programação estruturada dava forma ao desenvolvimento de software. Com o advento da programação orientada a objetos tudo mudou mais uma vez e as técnicas tiveram que evoluir e se adaptar ao novo paradigma.

Com o mundo orientado a objetos, surgiram processos cada vez mais robustos de desenvolvimento de software, o que culminou no mais conhecido processo de desenvolvimento orientado a objetos, o *Rational Unified Process* (RUP) (KRUCHTEN, 2004). A partir do RUP surgiu o Processo Unificado (PU) que mais tarde foi taxado de processo pesado com o surgimento das

metodologias ágeis. Os responsáveis pelo desenvolvimento dos métodos ágeis buscavam simplicidade e eficiência. Como o próprio manifesto ágil diz, eles estão "descobrindo maneiras melhores de desenvolver software" (BECK, 2014).

Com essa nova abordagem no desenvolvimento de software, o profissionalismo, a melhora nos processos, as avaliações de qualidade dos processos como CMM e MPS.Br, veio a padronização nas organizações. O Governo Federal não ficou de fora, o Tribunal de Contas da União (TCU) realizou uma ampla auditoria na Esfera Federal e constatou que há muitas falhas na governança de TI (TCU, 2008a). Com isso, o Ministério Público do Distrito Federal e Territórios (MPDFT) foi provocado a se adequar e elaborou seu Processo de Desenvolvimento de Software o MPDFT-UP.

O objetivo deste artigo é identificar gargalos, falhas e más práticas no processo atual e propor melhorias para essas situações identificadas no MPDFT-UP. Para isso, é feito um estudo sobre o modelo de processo atual e sobre algumas novas técnicas que estão sendo introduzidas aos poucos no órgão. É feito também um comparativo com o Processo Unificado e com metodologias ágeis e suas práticas.

Este artigo segue a seguinte estrutura, na seção 1 é detalhado o Processo Unificado (PU) que foi derivado do RUP. Na seção 2 é feita uma introdução ao universo ágil, é explicado o que vem a ser o manifesto ágil, a metodologia de gerenciamento de projetos ágeis Scrum e a metodologia de desenvolvimento de software ágil *eXtreme Programming* (XP). O MPDFT é introduzido rapidamente assim como o MPDFT-UP, que é o processo de desenvolvimento utilizado no órgão na seção 3. A avaliação que é objetivo deste artigo encontra-se na seção 4, bem como um detalhamento dos problemas encontrados e algumas sugestões de melhorias para os problemas encontrados. Por fim, são feitas as considerações finais.

## 1 Processo Unificado

Um processo de software é um conjunto de atividades que leva à produção de um produto de software (SOMMERVILLE, 2007). O Processo Unificado (PU) para desenvolvimento de software surgiu como uma alternativa ao já reconhecidamente ineficiente modelo sequencial ou em cascata (JACOBSON; BOOCH; RUMBAUGH, 1999). O modelo em cascata foi o primeiro a ser publicado e teve forte influência dos processos mais gerais de engenharia de sistema (SOMMERVILLE, 2007). Esse modelo (cascata) caracteriza-se pelo encadeamento das fases e pelo fato da fase seguinte não iniciar antes da atual ter terminado. Alguns estudos de sucesso/falha mostram que projetos que utilizaram o modelo em cascata obtiveram alta taxa de falha. Acredita-se que esse modelo ganhou forte adoção baseado em boatos e crenças ao longo dos anos (LARMAN, 2007).

Já o Processo Unificado é fortemente baseado em um modelo iterativo

e incremental. Esse modelo se mostrou mais eficiente em situações que os requisitos não estão completamente definidos e a entrega de resultados deve ser antecipada o máximo possível, além de reduzir a quantidade de defeitos. Larman (2007) cita em seu livro que o desenvolvimento iterativo e evolutivo considera normal que a fase de desenvolvimento comece antes dos requisitos terem sido definidos em detalhes; a realimentação é usada para esclarecer e aperfeiçoar as especificações em evolução.

Para fornecer um melhor entendimento sobre o Processo Unificado é comum dividi-lo em três perspectivas (SOMMERVILLE, 2007):

1. Uma perspectiva dinâmica, que mostra as fases do modelo ao longo do tempo.
2. Uma perspectiva estática, que mostra as atividades realizadas no processo.
3. Uma perspectiva prática, que sugere as boas práticas a serem usadas durante o processo.

Conforme citado anteriormente, o Processo Unificado, assim como o modelo tradicional e vários outros modelos, define um conjunto de fases a serem seguidas. As quatro principais fases são:

1. **Concepção** - O objetivo da fase de concepção é estabelecer um *business case* para o sistema.
2. **Elaboração** - Os objetivos da fase de elaboração são desenvolver um entendimento do domínio do problema, estabelecer um framework de arquitetura para o sistema, desenvolver o plano de projeto e identificar os riscos principais do projeto.
3. **Construção** - A fase de construção está essencialmente relacionada ao projeto, programação e teste de sistema.
4. **Transição** - A fase final do PU está relacionada à transferência do sistema da comunidade de desenvolvimento para a comunidade dos usuários e com a entrada do sistema em funcionamento no ambiente real.

Apesar das fases do PU se parecerem com as do modelo cascata elas não coincidem com as atividades do processo e estão mais estritamente ligadas ao negócio do que a assuntos técnicos (SOMMERVILLE, 2007). Outra característica marcante da estrutura de fases do PU é que elas podem se sobrepor ao longo do desenvolvimento, enquanto no modelo tradicional uma fase não inicia antes da finalização da fase atual.

Outra perspectiva, a estática, é organizada em disciplinas ou fluxos de trabalho (*workflows*). Uma disciplina é uma sequência de atividades que

produz um resultado de valor observável (CORPORATION, 1998). Por isso a disciplina é a parte responsável por uma das quatro perguntas básicas de um processo (quem, o quê, como e quando), o quando. É a disciplina que faz com que as outras perguntas sejam respondidas no momento adequado do ciclo de vida do processo. O PU possui nove disciplinas núcleo, seis agrupadas com o título de engenharia e outras três sob o título de suporte.

**Disciplinas de Engenharia**

1. **Modelagem de negócios** - Os processos de negócios são modelados usando casos de uso de negócios.
2. **Requisitos** - Os agentes que interagem com o sistema são identificados e os casos de uso são desenvolvidos para modelar os requisitos de sistema.
3. **Análise e projeto (design)** - Um modelo de projeto é criado e documentado usando modelos de arquitetura, modelos de componente, modelos de objeto e modelos de sequência.
4. **Implementação** - Os componentes de sistema são implementados e estruturados em subsistemas de implementação.
5. **Teste** - O teste é um processo iterativo realizados em conjunto com a implementação.
6. **Implantação** - Uma versão do produto é criada, distribuída aos usuários e instalada no local de trabalho.

**Disciplinas de Apoio/Suporte**

1. **Gestão de Configuração e Mudança** - Disciplina responsável por gerenciar as mudanças do sistema.
2. **Gerência de Projeto** - Disciplina que gerencia o desenvolvimento do sistema.
3. **Ambiente** - Disciplina relacionada à disponibilização de ferramentas apropriadas de software para a equipe de desenvolvimento.

A terceira perspectiva do Processo Unificado é a prática, que descreve boas práticas de engenharia de software recomendadas para uso em desenvolvimento de sistemas (SOMMERVILLE, 2007). São seis:

1. **Desenvolver software iterativamente** - Planejar os incrementos de software com base nas prioridades do cliente e desenvolver e entregar antes as características de sistema de maior prioridade no processo de desenvolvimento.

2. **Gerenciar requisitos** - Documentar explicitamente os requisitos do cliente e manter acompanhamento das mudanças desses requisitos.

3. **Usar arquiteturas baseadas em componentes** - Estruturar a arquitetura de sistemas com componentes.

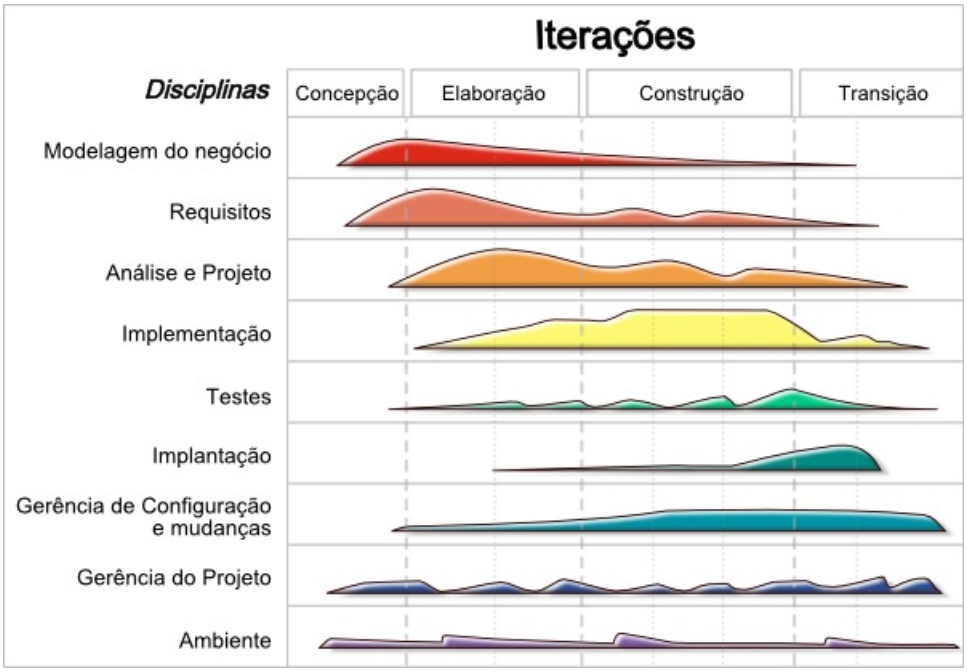
4. **Modelar o software visualmente** - Usar modelos gráficos de UML para apresentar as visões estática e dinâmicas do software.

5. **Verificar a qualidade do software** - Garantir que o software atenda aos padrões de qualidade da organização.

6. **Controlar as mudanças de software** - Gerenciar as mudanças de software, usando um sistema de gerenciamento de mudanças e procedimentos e ferramentas de gerenciamento de configuração.

A separação em perpectivas visa dar uma abordagem mais didática, mas o comum é que elas sejam visualizadas sob uma única ótica, ou seja, na prática as perspectivas estão sobrepostas conforme Figura 1.

Figura 1 – Processo Unificado: Disciplinas vs Fases.




Fonte: http://www.sutilab.com/cc/?aula=41

Essa visão de duas dimensões, para fases e disciplinas, foi uma grande evolução na forma de desenvolver software, pois enquanto este visa entregar

um produto de trabalho, um artefato, aquele busca atingir um objetivo de desenvolvimento. Apesar de parecer pouco isso foi uma mudança significativa na forma de escrever software. O PU proporcionou também um modelo flexível e adaptável que foi base para muitos outros.

## 2 Métodos Ágeis

Por volta da década de 1990 houve um movimento envolvendo desenvolvedores que não concordavam com o caminho que estava sendo tomado pelos processos de software. A burocracia, o peso do processo em detrimento do próprio produto a ser desenvolvido, o foco em artefatos que não necessariamente agregam valor, todos esses fatores foram motivos para a migração que levou a outro extremo, a redução brusca da burocracia. Isso fez com que esse grupo de desenvolvedores se reunissem e elaborassem uma séries de princípios a serem seguidos. Eles são conhecidos como os doze princípios do software ágil (BECK, 2014).

1. Nossa maior prioridade é satisfazer o cliente através da entrega contínua e adiantada de software com valor agregado.
2. Mudanças nos requisitos são bem-vindas, mesmo tardiamente no desenvolvimento. Processos ágeis tiram vantagem das mudanças visando vantagem competitiva para o cliente.
3. Entregar frequentemente software funcionando, de poucas semanas a poucos meses, com preferência à menor escala de tempo.
4. Pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto.
5. Construa projetos em torno de indivíduos motivados. Dê a eles o ambiente e o suporte necessário e confie neles para fazer o trabalho.
6. O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é através de conversa face a face.
7. Software funcionando é a medida primária de progresso.
8. Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente.
9. Contínua atenção à excelência técnica e bom design aumenta a agilidade.
10. Simplicidade – a arte de maximizar a quantidade de trabalho não realizado – é essencial.
11. As melhores arquiteturas, requisitos e designs emergem de equipes auto-organizáveis.

12. Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz e então refina e ajusta seu comportamento de acordo.

Esses princípios foram a base para o que mais tarde ficou conhecido como manifesto ágil (BECK, 2014). O manifesto ágil prega alguns valores descritos a seguir.

> 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** mais que processos e ferramentas
>
> **Software em funcionamento** mais que documentação abrangente
>
> **Colaboração com o cliente** mais que negociação de contratos
>
> **Responder a mudanças** mais que seguir um plano
>
> Ou seja, mesmo havendo valor nos itens à direita, valorizamos mais os itens à esquerda.

Essa foi praticamente a base para todas as metodologias consideradas ágeis atualmente. Todas tentam de maneiras diferentes fornecer os valores que foram definidos no manifesto ágil.

## 2.1 *eXtreme Programming* - XP

Um dos responsáveis pela revolução ágil foi Kent Beck, ao escrever o livro *Extreme Programming Explained* em 1999. Ele publicou a provavelmente mais famosa metodologia ágil até hoje, a Programação Extrema (*eXtreme Programming* - XP) (BECK, 1999). Seguindo o entendimento que a sociedade define valores para si mesma que conflitam com valores individuais, pois o que é pensado de forma coletiva, em geral, priva as pessoas de certas liberdades. A XP seguiu esse mesmo entendimento, que são necessários valores para indicar se o que está sendo feito é para um bem comum ou por interesses individuais. Surgiram então os valores da *eXtreme Programming*.

1. Comunicação
2. Simplicidade
3. Feedback
4. Coragem
5. Respeito

Na primeira edição do livro citado somente os quatro primeiro valores da XP foram descritos, só na segunda edição é que o respeito foi introduzido, pois foi percebido que ele é um valor que está permeando os outros e é fundamental para o sucesso do projeto (BECK, 2000).

O fato dos valores serem preservados ao longo do desenvolvimento é um bom indicativo de sucesso. No entanto, os valores eram muito vagos e foi necessário atacar de uma forma mais concreta, sugiram então as práticas. A XP possui um conjunto de práticas ou princípios, que são os principais responsáveis pela conformidade com o conceito de desenvolvimento ágil.

**Princípios ou Práticas da XP**

**Planejamento incremental** - Os requisitos são registrados em cartões de histórias e as histórias a serem incluídas em um release são determinadas pelo tempo disponível e sua prioridade relativa.

**Pequenos releases** - O conjunto mínimo útil de funcionalidade que agrega valor ao negócio é desenvolvido primeiro.

**Projeto Simples** - É realizado um projeto suficiente para atender aos requisitos atuais e nada mais.

**Desenvolvimento test-fist** - Um framework automatizado de teste unitário é usado para escrever os testes para uma nova parte da funcionalidade antes que esta seja implementada.

**Refactoring** - Espera-se que todos os desenvolvedores recriem o código continuamente tão logo os aprimoramentos do código forem encontrados.

**Programação em pares** - Os desenvolvedores trabalham em pares, um verificando o trabalho do outro e fornecendo apoio para realizar sempre um bom trabalho.

**Propriedade coletiva** - Os pares de desenvolvedores trabalham em todas as áreas do sistema, de tal maneira que não se formem ilhas de conhecimento, com todos os desenvolvedores de posse de todo o código.

**Integração contínua** - Tão logo o trabalho em uma tarefa seja concluído, este é integrado ao sistema como um todo.

**Ritmo sustentável** - Grandes quantidades de horas extras não são consideradas aceitáveis, pois, no médio prazo, há uma redução na qualidade do código e na produtividade.

**Cliente on-site** - Um representante do usuário final do sistema (o cliente) deve estar disponível em tempo integral para apoiar a equipe de XP.

Além das práticas a XP ainda define um conjunto de atividades que servem de orientação ao longo do desenvolvimento.

**Atividades da XP**

**Programar** - Ao fim do dia deve haver um programa.

**Testar** - Funcionalidades de software que não podem ser demonstradas por testes automatizados simplesmente não exitem.

**Escutar** - Os desenvolvedores devem ouvir o que os sistemas precisam fazer dos clientes.

**Projetar** - Projetar é criar uma estrutura que organize a lógica no sistema.

## 2.2 *Scrum*

Por volta de 2001 Ken Schwaber escreveu o livro *Agile Software Development with Scrum*, que seria o responsável pela iniciação da metodologia de gerenciamento de projetos ágil denominada *Scrum*. O seu nome é baseado no jogo de *Rugby*, além de alguns outros conceitos. O próprio autor define o *Scrum* como um framework e não uma metodologia, uma vez que ele não é um processo prescritivo, ou seja, não define o que deve ser feito em todas as circunstâncias (SCHWABER; BEEDLE, 2002). É fortemente embasado no processo de controle empírico, pois acredita que tarefas complexas possuem um grau tão auto de imprevisibilidade que é inviável tentar definir todo o processo.

Além disso, o Scrum define três papeis. Todas as responsabilidades de gerenciamento do projeto são divididas entre esses três papeis (SCHWABER; BEEDLE, 2002).

**Papéis do Scrum**

**Product Owner** - É o representante de todos os interessados no projeto e do sistema resultante.

**Time** - É o responsável pelo desenvolvimento das funcionalidades.

**ScrumMaster** - É o responsável pela correta implementação do processo Scrum.

Não existem só pessoas que estão classificadas em um dos três papeis, é normal haver outros interessados, contudo somente as pessoas que se encaixam nos papeis têm a total responsabilidade de fazer o necessário para o sucesso do projeto (SCHWABER; BEEDLE, 2002).

O *Scrum* define ainda alguns artefatos que são utilizados durante o gerenciamento dos projetos.

**Artefatos do Scrum**

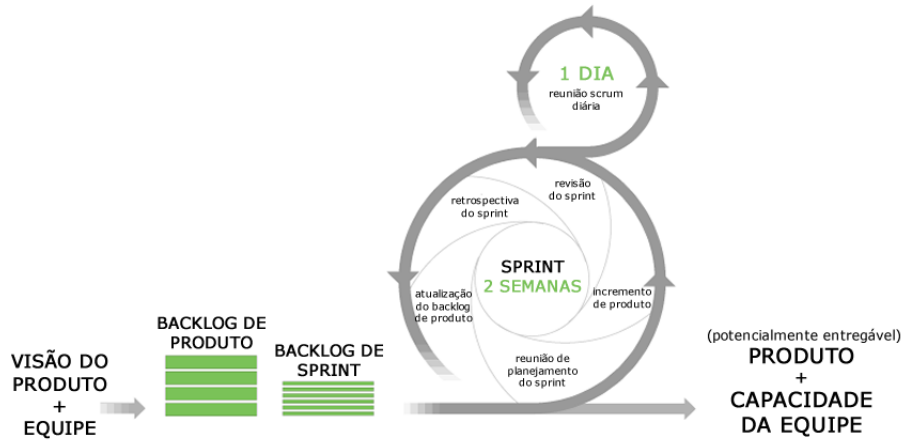
**Product Backlog** - É onde estão definidos os requisitos do sistema ou o produto sendo desenvolvido.

**Sprint Backlog** - Define o trabalho, ou tarefas, selecionadas do product backlog a serem implementadas pelo time na próxima Sprint.

**Burndown Chart** - O gráfico que mostra o conjunto de trabalho restante em relação ao tempo.

O Scrum segue um ciclo de desenvolvimento bem definido, conforme pode ser visualizado na Figura 2.

Figura 2 – Ciclo de Vida do Scrum.




Fonte: http://www.semeru.com.br/blog/category/daily-scrum-meeting/

# 3 MPDFT

O Ministério Público do Distrito Federal e Territórios (MPDFT) é um dos integrantes do Ministério Público da União (MPU). É o responsável pela defesa dos direitos dos cidadãos e dos interesses da sociedade. Dotado de poderes definidos na Constituição de 1988 que garantem a independência dos três poderes instituídos - Executivo, Legislativo e Judiciário - além da independência funcional, a indivisibilidade e a unidade (MPDFT, 2014a).

O Departamento de Tecnologia da Informação (DTI) do MPDFT é o responsável por toda a informatização do órgão. Várias equipes cuidam desde contratações envolvendo TI até desenvolvimento de soluções de software passando por administração da infraestrutura de informática (redes de computadores, banco de dados, segurança etc).

## 3.1 MPDFT-UP

O Processo Unificado do Ministério Público do Distrito Federal e Territórios (MPDFT-UP) (MPDFT, 2014b) é o resultado das recomendações do Tribunal de Conta da União (TCU) presentes no acórdão nº 1.603/2008 (TCU, 2008b) e publicadas no Diário Oficial da União (TCU, 2008a) que visam melhorias na governança de tecnologia da informação na Administração Pública Federal. No item 9.1.4 desse acórdão é proposta a utilização de medidas que estimulem a adoção de metodologias de desenvolvimento de sistemas.

Sabendo que os processos evoluem para explorar as capacidades das pessoas em uma organização e as características específicas dos sistemas que estão sendo desenvolvidos (SOMMERVILLE, 2007), o MPDFT-UP foi desenvolvido baseando-se no Processo Unificado e sendo, portanto um processo iterativo e incremental. A utilização do Processo Unificado não é tão crítica em relação a necessidades e pressões de mercado, mas por se tratar de um órgão público é necessária especial atenção à documentação dos sistemas desenvolvidos, formalização de demandas, designação de responsáveis pelo desenvolvimento e pelos aceites.

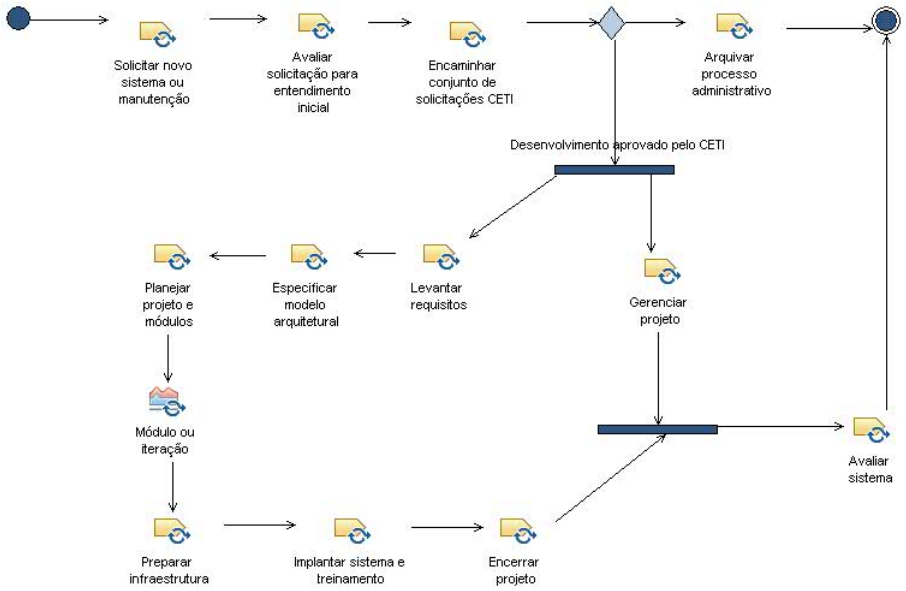
A Portaria Normativa nº 22 de 2009 é a que regula a metodologia de desenvolvimento de sistemas de informação do MPDFT (MPDFT, 2009). O MPDFT-UP define oito etapas para o desenvolvimento de software, que são os processos responsáveis pela organização do fluxo de trabalho.

**Etapas que compõem a metodologia de desenvolvimento de sistemas**

1. Entendimento inicial
2. Planejamento e gerência de projeto
3. Levantamento e análise de requisitos
4. Implementação e manutenção
5. Testes
6. Homologação
7. Treinamento e implantação
8. Avaliação

Apesar de existirem oito etapas somente seis estão detalhadas na documentação do MPDFT-UP (MPDFT, 2014b). A etapa de entendimento inicial e de avaliação não são detalhadas. Na Figura 3 é possível visualizar o fluxo de trabalho macro do processo. Além das etapas anteriores alguns papéis também foram definidos na Tabela Tabela 1.

Figura 3 – Fluxo de trabalho do MPDFT-UP.




Fonte: MPDFT (2014b)

Tabela 1 – Papéis do MPDFT-UP.

| | |
|---|---|
| Planejamento e gerência de projeto | Líder de projeto<br>Responsável técnico |
| Levantamento e análise de requisitos | Analista de negócio e requisitos de sistemas |
| Implementação e manutenção | Administrador de dados<br>Arquiteto de sistemas<br>Desenvolvedor |
| Testes | Analista de teste<br>Executor de teste |
| Homologação | Usuário gestor |
| Treinamento e implantação | Implantador<br>Pesquisador |

Fonte: Produzido pelo autor.

# 4 Avaliação

Este artigo tem por objetivo avaliar se a metodologia de desenvolvimento de software do Ministério Público do Distrito Federal de Territórios está sendo eficiente e eficaz. Os pontos a serem avaliados foram identificados através de uma pesquisa *in loco* e da observação das atividades diárias envolvendo desenvolvimento de software.

## 4.1 Problemas Identificados

### 4.1.1 Falta de Domínio do Processo

Um dos pontos críticos identificados é justamente a falta de conhecimento adequado sobre o processo do órgão. Antes de tecer qualquer comentário sobre as várias situações analisadas, esta é de longe a mais significativa. A maioria dos envolvidos sabe da existência do MPDFT-UP, mas seguer conhecem as etapas do processo. Quase toda a aplicação do processo de desenvolvimento é feita por meio de orientação dos integrantes mais antigos. Isso é extremamente problemático, pois situações tratadas no fluxo do processo não são eliminadas, além de serem incluídos vícios no trabalho diário. Não é que o processo não é seguido por falta de interesse, os integrantes das equipes de desenvolvimento não são incentivados ao uso.

### 4.1.2 Estrutura Organizacional

A cultura, o estilo e a estrutura organizacional influenciam a maneira como os projetos são executados (PMI, 2008, pag. 30). A estrutura organizacional do MPDFT pode ser considerada mista, mas mantém características mais funcionais do que projetizadas. Assim, várias dificuldades das organizações matriciais fracas ou funcionais surgem. Torna-se dificil de prever e acompanhar o ciclo de vida do software uma vez que cada setor tem prioridades próprias. A comunicação entre os setores é prejudicada, pois cada setor é uma caixa preta onde é muito difícil obter informações precisas.

### 4.1.3 Diferenças nos Processos

Haver diferenças entre os processos não implica necessariamente em problemas, ajustes podem ser feitos para melhor adequar um processo ao órgão. Conforme citado por Larman (2007), o PU é completamente adaptável e aberto a boas práticas de outros metodos. A introdução do PU não visa diminuir o valor desses outros métodos - muito pelo contrário (LARMAN, 2007). Contudo, os ajustes feitos no processo do MPDFT acabaram ofuscando alguns procedimentos. Das nove disciplinas originais do Processo Unificado, somente cinco são contempladas explicitamente no MPDFT-UP conforme Tabela 2. Disciplinas importantíssimas são tratadas de maneira reduzida dentro de outras etapas do processo do órgão.

A modelagem de negócio, que é uma disciplina do PU, foi em parte absorvida pela etapa de levantamento e análise de requisitos do MPDFT-UP. A etapa de entendimento inicial não é bem detalhada na documentação existente, mas infere-se que esteja mais relacionada a requisitos. A etapa de implementação e manutenção do MPDFT-UP é a que mais absorveu as atividades de análise e projeto, contudo a análise e o projeto de software propriamente ditos não são citados explicitamente no processo do órgão, somente o desenvolvimento da arquitetura, o projeto do banco de dados e a prototipação são citadas. A

Tabela 2 – Disciplinas: Processo Unificado vs MPDFT-UP.

| Processo Unificado | MPDFT-UP |
|---|---|
| Modelagem de negócios | - |
| - | Entendimento inicial |
| Requisitos | Levantamento e análise de requisitos |
| Análise e projeto | - |
| Implementação | Implementação e manutenção |
| Teste | Testes |
| - | Homologação |
| Implantação | Treinamento e implantação |
| - | Avaliação |
| Gestão de configuração e mudança | - |
| Gerência de projeto | Planejamento e gerência de projeto |
| Ambiente | - |

Fonte: Produzido pelo autor.

homologação é a etapa do processo do MPDFT que é responsável por obter a aceitação formal por parte do usuário gestor e não possui equivalente no PU.

A disciplina de gestão de configuração e mudança é uma das mais afetadas pela estrutura de processos do MPDFT, essa disciplina não existe diretamente no MPDFT-UP, mas em alguns momentos é citada em outras etapas. Por exemplo, na etapa de homologação é realizado o teste de homologação junto ao usuário gestor e caso seja encontrado algum problema é feita uma solicitação de mudança. Logo, deve haver algum mecanismo previsto para gerenciar as solicitações de mudança.

Por fim, a disciplina de ambiente é outra que no dia a dia do órgão não é tão problemática, mas para a área de desenvolvimento é bastante negligenciada. Há pouca padronização de ferramentas, frameworks e APIs utilizadas. Essas poucas não estão corretamente documentadas, estão guardadas nas cabeças de algumas pessoas, o que dificulta a disseminação do conhecimento sobre o que deve ser utilizado em cada situação.

### 4.1.4 Dificuldades nas Abordagens Ágeis

Há ainda algumas tentativas tímidas, porém com enorme potencial, de abordagens agilistas em meio aos processos de desenvolvimento de software do órgão. Há algumas equipes utilizando, de maneira experimental, *Scrum* e *Kanban*. Além de tentativas de adotar práticas do XP como a integração contínua. Contudo, novos problemas trazem novas dificuldades e uma das mais críticas em relação às abordagens ágeis é a falta de comprometimento do cliente. Conforme explicado anteriormente, um dos fatores cruciais para o sucesso de metodologias ágeis é o envolvimento do cliente e isso não é, em regra, respeitado no MPDFT. No Ministério Público do Distrito Federal e Territórios a atividade fim está relacionada à fiscalização da aplicação da Lei. Logo, os membros, oriundos da área de direito, são o topo da cadeia. Eles nem

sempre estão interessados nas técnicas, tecnologias, metodologias, e etapas necessárias para a construção de um sistema. Portanto, é muito difícil contar com a disponibilidade do maior interessado no trabalho final. Isso por si só já é um fator que pesa bastante na adoção de várias práticas agilista, mas o envolvimento do cliente é crucial para qualquer abordagem e não é por isso que o desenvolvimento de software deve parar. Como relatado anteriormente, a estrutura organizacional, que influencia na organização das equipes, também dificulta abordagens ágeis mais eficientes uma vez que o modelo adotado é mais voltado para uma organização funcional do que projetizada.

## 4.2 Sugestões e Melhorias

### 4.2.1 Capacitação do Integrantes

Já que existe um processo bem definido e uma fase de integração ou ambientação dos novos integrantes do órgão, uma das ações tomadas pode ser a inclusão de treinamentos não só no processo, mas nas tecnologias que serão utilizadas, sistemas a serem mantidos etc. A inclusão de um treinamento é fundamental para a posterior cobrança por parte dos membros das equipes de desenvolvimento.

### 4.2.2 Organização Projetizada

Ao se escolher o tipo de estrutura organizacional se está, na verdade, priorizando o que será mantido. Uma estrutura funcional prioriza os produtos, já uma estrutura projetizadas prioriza o desenvolvimento de projetos (LARSON ERIK W., 1989). Como há muitos projetos novos sendo desenvolvidos e outros esperando na fila para desenvolvimento futuro, priorizar o andamento dos projetos existentes é fundamental. A diminuição de gargalos e melhorias no processo passam necessariamente por uma reestruturação das equipes e da forma de trabalho atual.

### 4.2.3 Melhorias no Processo

Primeiro, a modelagem de negócio é uma disciplina que pode ser muito bem empregada no entendimento do negócio atual do cliente, que é a área fim do MPDFT.

A disciplina de análise e projeto realmente pode continuar fazendo parte da implementação, principalmente, com a adoção mais agressiva de metodologias ágeis.

Deve haver uma sintonia maior entre as áreas de implementação e teste, o trabalho entre os responsáveis pelas duas disciplinas pode acontecer de uma forma mais adequada para a produtividade. A implementação pode ter como saída código coberto por testes e os responsáveis pela disciplina de teste pode fornecer o roteiro de teste das funcionalidades desenvolvidas. Assim, as duas áreas ganham, a implementação por conseguir cobrir cenários não

planejados e enviar uma funcionalidade mais robusta para os responsáveis pelo teste e o teste por receber algo mais maduro e, eventualmente, até com testes automatizados.

Treinamento e implantação, principalmente em se tratando de implantação, há pouca documentação sobre os ambientes de desenvolvimento, teste, homologação e produção. Pouca documentação sobre passos a serem seguidos para publicar corretamente uma aplicação em outro ambiente, formato de e-mail de liberação de versão etc.

As disciplinas de gestão de configuração e mudança e ambiente, que não estão presentes no MPDFT-UP deveriam ganhar mais espaço, principalmente, por serem disciplinas de apoio, ou seja, trabalham em prol de todo o processo. Uma política de gestão de configuração minimizaria o tempo de aprendizado de um novo integrante, já que ele não vai precisar aprender a forma que uma nova equipe faz controle de código. A padronização do que é armazenado em controle de versão e estrutura de diretórios também é importante. Para ambiente, é bom estar escrito qual versão das ferramentas são utilizadas no órgão.

Limites bem definidos entre os modelos de processos é algo muito bom do ponto de vista didático, mas, no mundo prático, é inviável se utilizar somente de um modelo de desenvolvimento de software. Conforme explicado anteriormente, os modelos possuem vantagens e desvantagens, o que os tornam interessantes sob determinadas situações. Contudo, instituições sólidas buscam uma padronização no modo de trabalho, pois um modo de trabalho uniforme viabiliza o aprimoramento dos processso de software, no qual a diversidade de processos de software ao longo da organização é reduzida (SOMMERVILLE, 2007). Isso visa melhorias na comunicação e redução na curva de aprendizado de novos colaboradores.

Tendo em vista que o Processo Unificado é bastante flexível e aberto é incentivada a inclusão de práticas interessantes de outros métodos iterativos tais como: eXtreming Programming (XP) (BECK, 1999) e Scrum (SCHWABER; BEEDLE, 2002). Assim, é possível obter vantagem dos dois mundos.

### 4.2.4 Revisões Periódicas

Outra melhoria que pode ser feita no processo é a revisão periódica do próprio processo, das tecnologias e das práticas adotadas. Muito do que é utilizado atualmente no MPDFT é defasado tecnologicamente. Tanto em termos procedimentais quanto tecnológicos.

## Considerações finais

Grande parte dos problemas relacionados à utilização do Processo Unificado como base para o desenvolvimento/manutenção de software se deve à tentativa de tornar um processo já existente em nível executado e às vezes até

definido em outro processo. O que geralmente, e erroneamente, leva a equipe da qualidade a forçar a utilização de um novo e teoricamente melhor modelo. Assim, em vez de analisar como o software é tratado em uma organização e simplesmente documentar, a equipe de qualidade tenta sair de algo executado de forma meramente empírica, em muitos casos, para algo altamente documentado e, muitas vezes, burocrático. Essa é uma grande queixa por parte dos envolvidos no uso do processo e um dos principais responsáveis por um dos grandes mitos envolvendo o PU, o processo é pesado e, portanto, burocrático.

Conforme explicado nas seções anteriores, não há certos ou errados no mundo de desenvolvimento de software. É tudo uma questão de adaptação das técnicas/práticas no dia a dia. É completamente possível utilizar o Processo Unificado de forma ágil ou tornar uma metodologia ágil lenta e burocrática. Portanto, não é necessário mudar completamente o processo de desenvolvimento de software do MPDFT, basta adaptá-lo às reais necessidades do órgão.

Ao avaliar os problemas e as sugestões de melhoria será possível evoluir a forma de trabalho atual e obter grande retorno por parte dos participantes e do próprio processo. Mas, as melhorias não devem parar com este artigo, novos problemas surgirão e novas soluções serão propostas, pois um processo de desenvolvimento de software trata de processos, pessoas e ferramentas, logo, é algo vivo e deve estar em constante evolução.

## Referências


BECK, K. Embracing change with extreme programming. *Computer*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 32, n. 10, p. 70–77, out. 1999. ISSN 0018-9162. Disponível em: <http://dx.doi.org/10.1109/2.796139>. Citado 2 vezes nas páginas 10 e 19.

BECK, K. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 2000. (An Alan R. Apt Book Series). ISBN 9780201616415. Disponível em: <http://books.google.com.br/books?id=G8EL4H4vf7UC>. Citado na página 11.

BECK, K. *Manifesto para o desenvolvimento ágil de software*. 2014. Manifesto para o desenvolvimento ágil de software. Disponível em: <http://www.agilemanifesto.org/iso/ptbr/>. Acesso em: 20 mar. 2014. Citado 3 vezes nas páginas 5, 9 e 10.

CORPORATION, R. S. *Rational Unified Process - Best Practices for Software Development Teams*. 1998. Citado na página 7.

JACOBSON, I.; BOOCH, G.; RUMBAUGH, J. *The Unified Software Development Process*. Pearson Education, Limited, 1999. (Addison-Wesley Object Technology). ISBN 9780321822000. Disponível em:

<http://books.google.com.br/books?id=t4UVnwEACAAJ>. Citado na página 5.

KRUCHTEN, P. *The Rational Unified Process: An Introduction*. Addison-Wesley, 2004. (The Addison-Wesley object technology series). ISBN 9780321197702. Disponível em: <http://books.google.com.br/books?id=RYCMx6o47pMC>. Citado na página 4.

LARMAN, C. *Utilizando UML e Padrões*. 3. ed. São Paulo: Bookman, 2007. ISBN 9788560031528. Disponível em: <http://books.google.com.br/books?id=ZHtcynS03DIC>. Citado 3 vezes nas páginas 5, 6 e 16.

LARSON ERIK W., G. D. H. Significance of project management structure on development sucess. *IEEE Transactions on Engineering Management*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 36, n. 2, p. 119 – 125, maio 1989. Citado na página 18.

MPDFT, M. P. do Distrito Federal e T. *Portaria Normativa nº 22*. 2009. Citado na página 14.

MPDFT, M. P. do Distrito Federal e T. *MPDFT*. 2014. Disponível em: <http://www.mpdft.mp.br/>. Acesso em: 03 mar. 2014. Citado na página 13.

MPDFT, M. P. do Distrito Federal e T. *MPDFT-UP*. 2014. Processo de Desenvolvimento de Sistemas do MPDFT. Disponível em: <https://intranet.mpdft.mp.br/mpdft-up/>. Acesso em: 03 mar. 2014. Citado 2 vezes nas páginas 14 e 15.

PMI Project Management Institute. *Um Guia do Conjunto de Conhecimentos em Gerenciamento de Projetos*: Guia pmbok. [S.l.]: Project Management Institute, Inc., 2008. (P M series). ISBN 978-1-933890-70-8. Citado na página 16.

SCHWABER, K.; BEEDLE, M. *Agile software development with scrum*. Prentice Hall, 2002. (Series in agile software development). ISBN 9780130676344. Disponível em: <http://books.google.com.br/books?id=BpFYAAAAYAAJ>. Citado 2 vezes nas páginas 12 e 19.

SOMMERVILLE, I. *Engenharia de software*. 8. ed. São Paulo: Pearson Addison-Wesley, 2007. Citado 5 vezes nas páginas 5, 6, 7, 14 e 19.

TCU, T. de Contas da U. 2008. 243 p. Acórdão nº 1.603/2008. Disponível em: <http://pesquisa.in.gov.br/imprensa/jsp/visualiza/index.jsp?jornal=1&pagina=243&data=18/08/2008>. Acesso em: 23 mar. 2014. Citado 2 vezes nas páginas 5 e 14.

TCU, T. de Contas da U. 2008. Acórdão nº 1.603/2008. Disponível em: <http://www.tcu.gov.br/Consultas/Juris/Docs/judoc%5CAcord%

5C20080814%5C008-380-2007-1-GP.doc>. Acesso em: 23 mar. 2014. Citado na página 14.