PROCESSOS DE SOFTWARE

Engenharia de Software I

José Roberto Holanda[1]
Luan Henrique dos Santos Silva[2]
Luana Oliveira Santos[3]

RESUMO

Este trabalho visa apresentar o conceito de processo de software, conjuntos de padrões, atividades e métodos para a produção de software, por meio do resumo e análise de um capítulo do livro técnico sobre Engenharia de Software, de Ian Sommerville, tal atividade faz parte das avaliações da disciplina de Engenharia de Software I do IFPA. Partindo disso, temos por objetivo introduzir os conceitos dos modelos de processo de software mais conhecidos, suas diferenças, vantagens e desvantagens, as atividades de processo de software fundamentais, bem como a organização e melhoria dos processos.

Com isso, buscamos clarear para o leitor a concepção do processo de software, de forma expositiva, discorrendo todas as principais ideias desta área.

Palavras-chave: Processo de Software. Modelos de Processo. Produção de Software.

1. INTRODUÇÃO

Processo de software é um grupo de atividades relacionadas entre si, que resultam na produção de um sistema de software. Existem diversos tipos e modelos de processo de software, é difícil dizer qual deles é o melhor, pois dependem muito do projeto e do contexto em que serão aplicados, dessa forma, não há um processo universal que serve para qualquer situação (embora existam algumas tentativas para isso).

Contudo, os processos de software, quaisquer que sejam, devem incluir quatro fundamentos:

- Especificação: Sua funcionalidade e restrições impostas pelo cliente por meio dos requisitos (planejamento do projeto).

[1] Graduando do Curso de Tecnologia em Análise e Desenvolvimento de Sistemas. E-mail: jsrobert1069@gmail.com
[2] Graduando do Curso de Tecnologia em Análise e Desenvolvimento de Sistemas. E-mail: luanhsantos14@gmail.com
[3] Graduando do Curso de Tecnologia em Análise e Desenvolvimento de Sistemas. E-mail: luaolivsant@gmail.com

- Desenvolvimento: A parte de criação, a construção do sistema que atenderá aos requisitos.
- Validação: Verificação da parte de desenvolvimento em relação à especificação.
- Evolução: Softwares dificilmente permanecem estáticos, conforme surgem novas demandas para o sistema, é preciso que passe pelo processo novamente, melhorando o projeto.

Um processo de software é uma ativade complexa, pois há muitas fases a serem percorridas pela equipe de projeto, e cada fase gera um resultado diferente, é preciso ter um nível de organização altíssimo para ter uma boa relação entre os diferentes grupos participantes do processo e manter os requisitos e condições do cliente de acordo.

Mesmo não havendo um processo universal, é comum ter práticas que se repetem em diferentes modelos, como um planejamento, abordagem focada nos requisitos e o própio gerenciamento da atividade. Ademais, é possível que sejam mesclados diferentes modelos na construção de um sistema, pois segundo Sommerville (2007):

> O termo sistema é usado universalmente. Falamos de sistemas de computadores, sistemas operacionais, sistemas de pagamento, sistema educacional, sistema de governo, etc. Existem obviamente, usos bastante diferentes para a palavra sistema.

Existem sistemas tão complexos, que adotam dois ou mais tipos de modelo durante suas construções, cabe a equipe de planejamento saber articular esses métodos de trabalho, para que um complemente o outro, ao invés de atrapalhar.

2. ANÁLISE E COMENTÁRIO DO CONTEÚDO

2.1. MODELOS DE PROCESSO DE SOFTWARE

2.1.1. MODELO CASCATA

Entende-se por cascata, um método de desenvolvimento em que a próxima parte de um projeto só deve ser realizada se anterior já estiver finalizada, uma sequência de construções uma após a outra, a próxima fase sempre aproveita, e usa de base, a anterior.

O processo de desenvolvimento de software apresenta uma série de estágios que "cascatam" de um para o outro (um só acontece se o anterior acontecer), definindo assim o modelo em cascata ou ciclo de vida do software. Esse processo é dirigido por um plano, em que é necessário planejar e criar um cronograma de todas as atividades antes de começar o desenvolvimento propriamente dito. Os estágios do modelo em cascata são:

- Análise e definição dos requisitos: os usuários informo como desejam o sistema (serviços, restrições e metas) e os engenheiros especificam e refinam esses requisitos.
- Projeto do sistema e do software: os requisitos refinados são então divididos em requisitos do sistema de hardware e software. O projeto de software envolve a identificação e a descrição das abstrações fundamentais do sistema.
- Implementação e teste de unidade: o projeto começa a ser desenvolvido como um conjunto ou unidade de programa, verificados individualmente, se satisfazem a especificação.

- Integração e teste de sistema: as unidades de programa individuais começam a ser integradas como um sistema de fato, e são testados em conjunto, garantindo a especificação. Após a testagem, o software é entregue ao cliente
- Operação e manutenção: geralmente é a fase mais longa do ciclo de vida o sistema é colocado em uso. A manutenção corrige os erros que não foram descobertos antes e aperfeiçoam serviços à medida que novos requisitos são descobertos.

O resultado de cada fase consiste em um ou mais documentos que são aprovados. No desenvolvimento de hardware (altos custos de produção) faz sentido a fase seguinte é só começar após a anterior terá terminado, mas no desenvolvimento de software as fases se sobrepõe e geram feedback de informações entre elas tornando o processo não linear na prática. Os documentos antes produzidos devem ser mudados de acordo com a aprovação do cliente, mas isso atrasa o processo ou congela-o, gerando sistemas mal estruturados. Em sua fase final de utilização, erros e omissões podem ser descobertos gerando a necessidade do sistema evoluir. Pela pouca flexibilidade, o modelo em cascata é adequado somente para alguns tipos de sistema como sistemas embarcados software, sistemas críticos e grande sistemas de software que fazem parte de sistemas mais amplos. O modelo em cascata não é recomendado para situações em que a comunicação informal é possível e quando os requisitos mudam rapidamente.

2.1.2 DESENVOLVIMENTO INCREMENTAL

Criar uma implementação inicial, obter feedback dos usuários e fazer o software evolui por novas versões. As atividades de especificação, desenvolvimento e validação são intercaladas, em vez de separadas. É a abordagem mais comum no desenvolvimento de software hoje podendo ser por plano (incrementos são identificados antecipadamente) ou ágil (incrementos iniciais identificados mas os finais dependen do progresso). É uma abordagem melhor que a em cascata em projetos com requisitos propensos a mudar. Reflete a maneira como solucionamos os problemas, pois, raramente elaboramos uma solução completa para o problema com antecedência, mas caminhamos passo a passo e retrocedemos quando erramos. Também é mais barato fazer alterações nesse modo que no cascata. Geralmente a cada incremento ao sistema são inseridas novos funcionalidades, a versão inicial conta com os requisitos mais importantes o urgentes.

Vantagens do desenvolvimento incremental em relação ao cascata:

- O custo de implementação das mudanças nos requisitos é reduzido, menor documentação para ser refeita.
- Obtenção de feedback do cliente sobre o trabalho de desenvolvimento é mais fácil.
- A entrega e a implantação antecipadas de um software útil para o cliente são possíveis, mesmo se toda a funcionalidade não tiver sido incluída.

Os problemas do desenvolvimento incremental são o de existirem procedimentos burocráticos evoluídos, podendo gerar incompatibilidade entre burocracia e processo ágil.

Além disso o processo não é visível, pois não é econômico produzir documentos que reflitam cada versão do sistema. E a estrutura do sistema tende a se degradar a medida que os incrementos são adicionados, para isso devem ser refatorados regularmente.

Ademais, isso torna o desenvolvimento incremental inviável para sistemas grandes, que precisam de um framework ou arquitetura estáveis, de vida longa, que devem ser planejados antecipadamente.

2.1.3 INTEGRAÇÃO E CONFIGURAÇÃO

Reúso de software: Acontece quando as pessoas que trabalham no projeto conhecem ou procuram algum código similar para o projeto. O reúso informal já ocorre desde os anos 2000, mas foi "estabilizado", as abordagens contam com bases de componentes reusáveis e um framework de integração. Os componentes mais reusados são:

- Sistemas de aplicação *stand-alone*, usados em ambiente particular devem ser adaptados para uma aplicação específica.
- Coleções de objetos, pacotes a serem integrados a um framework de componentes.
- *Web services*, desenvolvidos com os padrões de serviços disponíveis na internet.

Os estágios em integração e configuração são:

- Especificação dos requisitos: Requisitos iniciais propostos, não precisam ser muito detalhados.
- Descoberta e avaliação do software: É feita uma busca em softwares reutilizáveis com base na descrição dos requisitos e verifica-se se são úteis ou não.
- Refinamento dos requisitos: Feito em cima do software em reúso com as devidas adaptações.
- Configuração da aplicação: Verifica-se se há uma aplicação de prateleira que satisfaça os requisitos, a fim de ser incluída.
- Adaptação e integração dos componentes: Caso não haja reutilizáveis, novos componentes são desenvolvidos e integrados ao sistema final.

Essa aboragem reduz a criação de software, diminuindo custos e riscos, entretanto podem haver inconcessões quanto aos requisitos e o sistema pode acabar não satisfazendo as necessidades reais dos usuários. Ademais, a evolução do sistema se perde, pois os componentes melhorados estão sob outro controle agora.

2.2 ATIVIDADES DO PROCESSO

2.2.1 ESPECIFICAÇÃO DE SOFTWARE

É o processo de definir quais os serviços são necessários e identificar as restrições da aplicação, é um estágio crítico e cuidadoso, visto que influencia diretamente as outras etapas e na implementação do sistema.

Antes de iniciar esse projeto de especificação, é possível realizar um estudo de marketing para avaliar se há demanda para a aplicação.

Esta etapa visa um documento de requisitos que atendam os stakeholders e normalmente é apresentado em dois níveis: o nível do usuário, que recebe especificações superficiais, e o de desenvolvedores, que recebem um documento mais detalhado.

As três principais atividades desse processo são:

- Elicitação e análise: processo de pesquisa com potenciais usuários, no qual podem ser desenvolvidos protótipos do projeto para compreender as necessidades.
- Especificações: atividade de construção do software com base nas declarações dos clientes e da funcionalidade necessária.

- Validação: confere o realismo, consistência e integridade, visando a correção de erros na documentação.

Essas etapas estão entrelaçadas, fazendo com que durante esses processos surjam novas necessidades e especificações. Em alguns métodos a especificação ocorre durante o desenvolvimento, os requisitos são apresentados previamente e a elicitação vem de usuários que fazem parte do time de desenvolvimento.

2.2.2 IMPLEMENTAÇÃO DE SOFTWARE

Nesse estágio ocorre a criação de um executável a ser entregue para o cliente, essa etapa é dividida em duas partes: projeto, o qual refere-se ao design, e programação. Esses processos podem ser realizados de maneira intercalada e sem documentação formal.

O projeto é uma descrição da estrutura de software a ser implementado, dos modelos e dados utilizados pelo sistema, das interfaces, componentes e até o algoritmo a ser usado e com detalhes que são acrescentados à medida que desenvolvem o projeto

A maioria dos softwares interagem com outros, desde o sistema operacional e banco de dados até **middleware**, compondo a "plataforma de software", que é essencial para que os projetistas decidam como integrar a aplicação. Alguns dos processos são:

1. *Projeto de arquitetura:* identificação da estrutura global e componentes principais, observando seu funcionamento e distribuição.
2. *Projeto banco de dados:* no qual são projetadas as estruturas e como devem ser representadas, dependendo da definição entre um banco de dados já existente e a criação de um novo.
3. *Projeto de interface:* são definidas as interfaces e componentes, se feito de forma precisa, um componente pode ser utilizado sem que seja necessário saber como foi implementado, posteriormente são projetados separadamente.
4. *Seleção e projeto:* Nessa etapa há busca por componentes que poderão ser utilizados novamente depois, se não for possível, são projetados novos. Nesse estágio o projeto é uma descrição simples dos componentes e os detalhes são definidos pelo programador.

Essas etapas levam às saídas do projeto, os quais são detalhadamente descritos na documentação, e que podem ser diagramas, mas se forem utilizados métodos ágeis, as saídas não serão documentos de especificação separados, mas serão representados no código. A etapa de programação é um processo individual do programador que não possui uma forma genérica a ser seguida. Apesar de não possuir forma correta de ser feita, um passo seguido por todos os desenvolvedores é a depuração, que visa a identificação e correção de defeitos.

2.2.3 VALIDAÇÃO DE SOFTWARE

A verificação e validação buscam comprovar a funcionalidade de acordo com as especificações exigidas e expectativas do cliente, a principal forma é executando o programa, mas pode ser necessário a revisão de toda a estrutura e código.

Exceto em pequenas aplicações, no qual os componentes são testados de forma individual e com dados reais, normalmente de usuários selecionados, no chamado "teste-beta".

1. *Teste de componente:* Os componentes são testados de maneira individual e sem as demais partes do sistema, podendo ser as classes de objetos ou agrupamento coerentes desses componentes.
2. *Teste de sistema:* São integrados para criar um sistema completo, nessa fase encontram-se erros de interações imprevistas entre os componentes e problemas de interface, essa fase busca mostrar que o sistema atende todos os requisitos funcionais e não funcionais.
3. *Teste do cliente:* Nessa fase o cliente utiliza seus dados para teste da aplicação, fazendo a verificação antes que o programa se torne operacional, demonstrando se o produto satisfaz ou não as necessidades do cliente.

Normalmente os bugs e problemas na UI são identificados ainda na fase de construção do código durante o processo de depuração, fazendo-se necessário repetir o mesmo procedimento buscando eficiência e funcionalidade do mesmo.

Em alguns casos é necessária uma equipe independente de testadores que seguem um plano de testes que foi desenvolvido a partir da especificação e do projeto do sistema.

2.2.4 EVOLUÇÃO DO SOFTWARE

A flexibilidade do software é uma das razões para que ele seja incorporado a grandes e complexos sistemas, visto que sua manutenção ou alteração é um processo mais barato se comparado aos valores de modificações de hardware.

A manutenção e o desenvolvimento são etapas que possuem cada vez menos distinções, pois são vistos como processos evolutivos da engenharia de software, no qual este é alterado continuamente seguindo as necessidades e requisitos do cliente.

2.3 LIDANDO COM MUDANÇAS

A mudança é inevitável em todos os grandes projetos de software. Os requisitos do sistema mudam à medida que as empresas reagem a pressões externas, à concorrência e a mudanças nas prioridades da gestão. Ao passo que novas tecnologias são disponibilizadas, novas abordagens de projeto e de implementação se tornam possíveis. Portanto, seja qual for o modelo de processo de software utilizado, é essencial que ele consiga apoiar as mudanças no software que está sendo desenvolvido.

A mudança eleva os custos de desenvolvimento de software, já que isso normalmente significa que o trabalho já concluído precisará ser refeito: isso é retrabalho. Por exemplo, se os relacionamentos entre os requisitos em um sistema forem analisados e novos requisitos forem identificados, parte ou toda a análise de requisitos deve ser refeita. Então, pode ser necessário reprojetar o sistema para entregar os novos requisitos, mudar quaisquer programas que tenham sido desenvolvidos e testar o sistema novamente.

Duas abordagens relacionadas podem ser utilizadas para reduzir os custos de retrabalho:

1. Antecipação da mudança.

O processo de software inclui atividades que podem antecipar ou prever possíveis mudanças antes da necessidade de um retrabalho. Um Protótipo do sistema pode ser desenvolvido para exibir aos clientes algumas características principais do sistema. experimentar o protótipo e refinar seus requisitos.

2. Tolerância à mudança.

O processo e o software são projetados de modo que as mudanças no sistema possam ser feitas com facilidade. Isso envolve, normalmente, alguma forma de desenvolvimento incremental. As mudanças propostas podem ser implementadas em incrementos que ainda não foram desenvolvidos.

Maneiras de lidar com as mudanças e com as variações nos requisitos do sistema:

1. Prototipação do sistema:

Uma versão ou parte do sistema é desenvolvida rapidamente para verificar os requisitos do cliente e a viabilidade de algumas decisões de projeto Essa é uma maneira de antecipar a mudança, já que permite aos usuários experimentarem o sistema antes da entrega e. assim, refinar seus requisitos.

2. Entrega incremental:

O sistema é fornecido para o cliente em incrementos, a fim de que comentários e experimentações sejam feitos. Essa é uma maneira de antecipar as mudanças e aumentar a tolerância a elas, evitando o comprometimento prematuro com os requisitos do sistema como um todo e permitindo que as mudanças sejam incorporadas aos incrementos finais a um custo relativamente baixo.

2.3.1. PROTOTIPAÇÃO

O protótipo é uma versão inicial de um sistema utilizado para demonstrar conceitos, experimentar opções de projeto e descobrir mais sobre o problema e suas possíveis soluções.

Um protótipo de software pode ser utilizado em um processo de desenvolvimento para ajudar a antecipar as mudanças que podem ser necessárias:

1. No processo de engenharia de requisitos, um protótipo pode ajudar na elicitação e validação dos requisitos do sistema.
2. No processo de projeto do sistema, um protótipo pode ser utilizado para explorar soluções de software e no desenvolvimento de uma interface com o usuário para o sistema.

O **próximo estágio** no processo é decidir o que colocar e, talvez ainda mais importante, o que deixar de fora do sistema prototipado. Para reduzir os custos de prototipação e acelerar o cronograma de entrega;

O **estágio final** do processo é a avaliação do protótipo. Nessa etapa, deve ser realizado o treinamento dos usuários, e os objetivos do protótipo devem ser usados para a criação de um plano de avaliação. Os usuários precisam de tempo para se acostumar com um sistema novo e estabelecer um padrão normal de uso.

Os testadores do protótipo podem não ser usuários típicos do sistema. Pode não haver tempo suficiente para treinar os usuários durante a avaliação do protótipo. Se o protótipo for lento, os avaliadores poderão ajustar sua maneira de trabalhar, a fim de evitar as características do sistema com tempos de resposta lentos.

2.3.2. ENTREGA INCREMENTAL

A entrega incremental é uma abordagem para o desenvolvimento de software em que alguns dos incrementos desenvolvidos são fornecidos para o cliente e implantados para uso em seu ambiente de trabalho. Em um processo de entrega incremental, os clientes definem quais serviços são mais e menos importantes para eles. Uma série de incrementos é definida para entrega, e cada um deles proporciona um subconjunto da funcionalidade do sistema. A alocação dos serviços aos incrementos depende da prioridade do serviço; são implementados e entregues, em primeiro lugar, os de maior prioridade.

A entrega incremental tem uma série de vantagens:

1. Os clientes podem usar incrementos iniciais como protótipos e adquirir experiência;
2. Os clientes não precisam esperar até o sistema inteiro ser fornecido, o que permite usar o software imediatamente;
3. O processo mantém os benefícios do desenvolvimento incremental, sendo fácil incorporar mudanças no sistema;
4. Os serviços de prioridade mais elevada são fornecidos primeiro, consequentemente recebem a maioria dos testes. estarão menos propensos a encontrar falhas.

No entanto, também existem problemas na entrega incremental.

1. Os usuários precisam de toda a funcionalidade do sistema antigo e, normalmente, não estão dispostos a experimentar um novo sistema que esteja incompleto;
2. Pode ser difícil identificar os recursos comuns necessários para todos os incrementos;
3. Na abordagem incremental, não há especificação completa do sistema até o incremento final ser especificado. Isso requer uma nova forma de contrato, à qual grandes clientes, a exemplo de órgãos governamentais, podem encontrar dificuldade para se adaptar.

Para alguns tipos de sistemas, o modelo incremental de desenvolvimento e de entrega não é a melhor abordagem.

2.4 MELHORIA DE PROCESSO

Hoje em dia. há uma demanda constante da indústria por software melhor e mais barato, que deve ser fornecido em prazos ainda mais apertados. Consequentemente, muitas empresas se voltaram para a melhoria do processo de software como uma maneira de melhorar a qualidade de seu software, reduzindo custos ou acelerando seus processos de desenvolvimento.

A melhoria dos processos significa compreender os processos existentes e modificá-los para aumentar a qualidade do produto e/ou reduzir os custos e o tempo de desenvolvimento.

São empregadas duas abordagens bem diferentes para melhoria e mudança de processos:

1. A abordagem de maturidade do processo, que se concentra na melhoria dos processos e do gerenciamento do projeto e na introdução de práticas recomendadas de engenharia de software em uma organização. Os objetivos primários dessa abordagem são a maior qualidade do produto e a previsibilidade do processo.
2. A abordagem ágil, que se concentra no desenvolvimento iterativo e na redução dos custos gerais do processo de software. As características primárias dos métodos ágeis são a entrega rápida da funcionalidade e a rapidez de resposta para os requisitos mutáveis do cliente.

A melhoria geral do processo é cíclica, Os estágios nesse processo são:

1. Medição do processo:
Essas medições formam um ponto de partida que permite decidir se as melhorias do processo foram eficazes. À medida que se introduzem mais melhorias, os mesmos atributos devem ser reavaliados;
2. Análise do processo:
O processo atual é avaliado a fim de identificar os pontos fracos e os gargalos;
3. Mudança no processo:
As mudanças são propostas para abordar alguns dos pontos fracos identificados. Elas são introduzidas e o ciclo continua coletando dados sobre sua eficácia.

Os níveis do modelo de maturidade do processo são:

1. Inicial
As metas associadas à área de processo são cumpridas e o escopo do trabalho a ser realizado é definido e comunicado para os membros do time.

2. Gerenciado
Nesse nível, as metas associadas à área de processo são cumpridas e as políticas organizacionais estão implantadas, definindo quando cada processo deve ser utilizado. Deve haver planos de projeto documentados que definam as metas do projeto.

3. Definido
Esse nível se concentra na padronização organizacional e na implantação dos processos.

4. Gerenciado quantitativamente
Nesse nível, há uma responsabilidade organizacional de empregar o método estatístico entre outros métodos quantitativos para controlar os subprocessos.

5. Em otimização
Nesse nível mais alto, a organização deve usar as medições de processos e produtos para orientara melhoria dos processos. As tendências devem ser analisadas e adaptados às necessidades da empresa

3. CONSIDERAÇÕES FINAIS

Portanto, processos de software são atividades cruciais para a produção de um software de qualidade e possuem modelos como representações abstratas do processo. Outrossim, modelos de processo genéricos descrevem a organização dos processos de software, aliados à união de requisitos, design, validação e outras subatividades necessárias para a implementação e evolução do software posteriormente.

REFERÊNCIAS BIBLIOGRÁFICAS

SOMMERVILLE, Ian. Processos de Software. In: SOMMERVILLE, Ian. **Engenharia de Software**. 10. ed. São Paulo: Pearson, 2019. Cap. 2. p. 29-50.

SOMMERVILLE, Ian. **Engenharia de Software**. 8. ed. Tradução Selma Shin Melnikoff; Reginaldo Arakaki; Edilson de Andrade Barbosa. São Paulo: Pearson, 2007.