Evolução e Manutenção de software

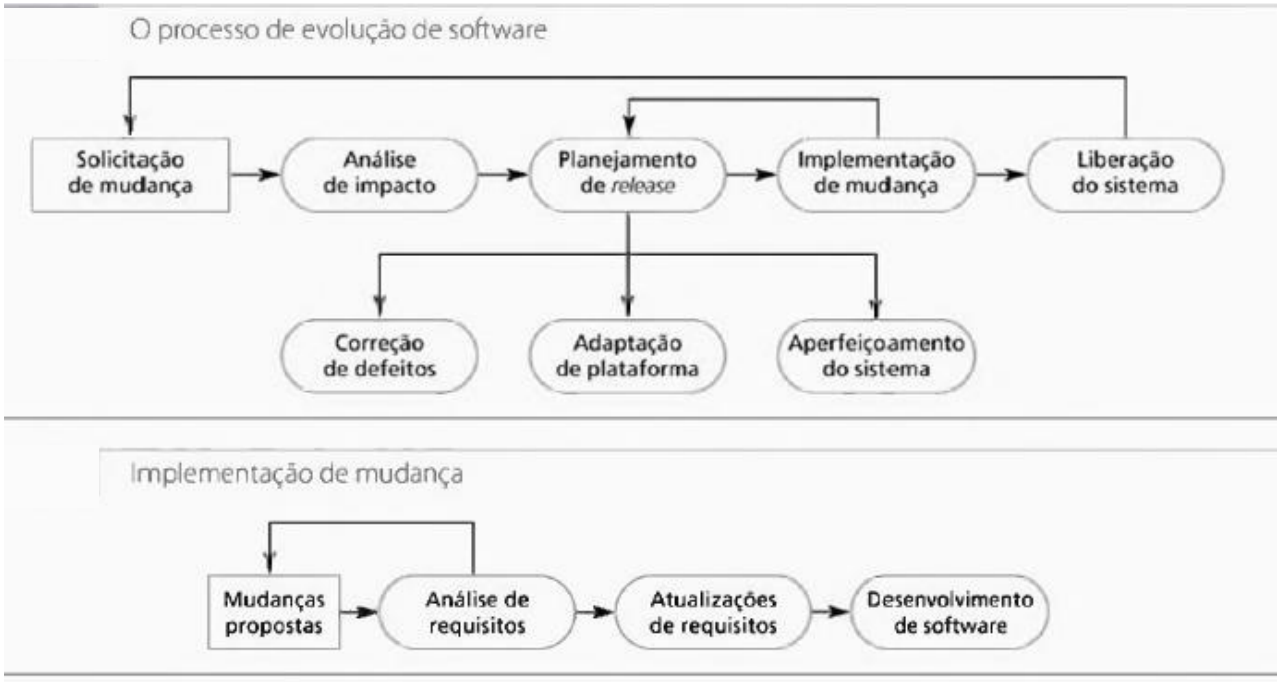
O desenvolvimento de software não é interrompido quando o sistema é entregue, mas continua por toda a vida útil do sistema. Depois que o sistema e implantado, para que ele se mantenha útil é inevitável que ocorram mudanças —, mudanças nos negócios e nas expectativas dos usuários, que geram novos requisitos para o software. Partes do software podem precisar ser modificadas para corrigir erros encontrados na operação e para que o software se adapte às alterações de sua plataforma de hardware e software, bem como para melhorar seu desempenho ou outras características não funcionais.

A evolução dos processos de software pode variar dependendo do tipo de software que esteja sendo mantido, dos processos de desenvolvimento usados em uma organização e das habilidades das pessoas envolvidas. Em algumas organizações, a evolução pode ser um processo informal em que as solicitações de mudança resultam, na maior parte, das conversas entre os usuários do sistema e desenvolvedores. Em outras empresas, é um processo formal com documentação estruturada produzida em cada estágio do processo.

Em todas as organizações, as propostas de mudança no sistema são os acionadores para a evolução. As propostas de mudança podem vir de requisitos já existentes que não tenham sido implementados no release de sistema, solicitações de novos requisitos, relatórios de bugs do sistema apontados pelos stakeholders e novas ideias para melhoria do software vindas da equipe de desenvolvimento. Os processos de identificação de mudanças e de evolução de sistema são cíclicos e continuam durante toda a vida de um sistema.

Figura 1

Figura 2



A dinâmica da evolução de programas é o estudo da mudança de sistema. Nas décadas de 1970 e 1980, Lehman e Belady (1985) realizaram vários estudos empíricos sobre a mudança de sistema com intenção de compreender mais sobre as características de evolução de software. O trabalho continuou na década de 1990 com Lehman e outros pesquisando o significado de feedback nos processos de evolução (LEHMAN, 1996; LEHMAN et al., 1998; LEHMAN et al., 2001). A partir desses estudos, eles propuseram as 'Leis de Lehman', relativas as mudanças de sistema.

Figura 3

Lei	Descrição
Mudança contínua	Um programa usado em um ambiente do mundo real deve necessariamente mudar, ou se torna progressivamente menos útil nesse ambiente.
Aumento da complexidade	Como um programa em evolução muda, sua estrutura tende a tornar-se mais complexa. Recursos extras devem ser dedicados a preservar e simplicar a estrutura.
Evolução de programa de grande porte	A evolução de programa é um processo de autorregulação. Atributos de sistema como tamanho, tempo entre *releases* e número de erros relatados são aproximadamente invariáveis para cada *release* do sistema.
Estabilidade organizacional	Ao longo da vida de um programa, sua taxa de desenvolvimento é aproximadamente constante e independente dos recursos destinados ao desenvolvimento do sistema.
Conservação da familiaridade	Durante a vigência de um sistema, a mudança incremental em cada *release* é aproximadamente constante.
Crescimento contínuo	A funcionalidade oferecida pelos sistemas tem de aumentar continuamente para manter a satisfação do usuário.
Declínio de qualidade	A qualidade dos sistemas cairá, a menos que eles sejam modificados para refletir mudanças em seu ambiente operacional.
Sistema de *feedback*	Os processos de evolução incorporam sistemas de *feedback* multiagentes, *multiloop*, e você deve tratá-los como sistemas de *feedback* para alcançar significativa melhoria do produto.

Manutenção de Software

O desenvolvimento de software não é interrompido quando o sistema é entregue, mas continua por toda a vida útil do sistema. Depois que o sistema é implantado, para que ele se mantenha útil é inevitável que ocorram mudanças — mudanças nos negócios e nas expectativas dos usuários, que geram novos requisitos para o software.

A manutenção de software é o processo geral de mudança em um sistema depois que ele é liberado para uso.

As alterações feitas no software podem ser simples mudanças para correção de erros de codificação, até mudanças mais extensas para correção de erros de projeto, ou melhorias significativas para corrigir erros de especificação ou acomodar novos requisitos. As mudanças são implementadas por meio da modificação de componentes do sistema existente e, quando necessário, por meio da adição de novos componentes.

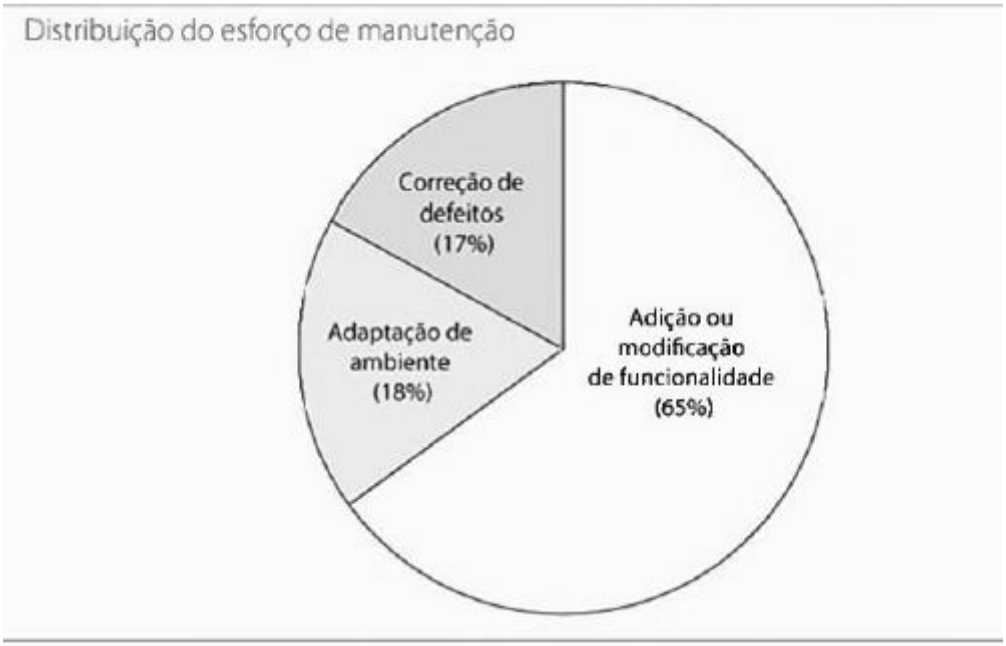
Existem três diferentes tipos de manutenção de software:

1. **Correção de defeitos**. Erros de codificação são relativamente baratos para serem corrigidos; erros de projeto são mais caros, pois podem implicar reescrever vários componentes de programa. Erros de requisitos são os mais caros para se corrigir devido ao extenso reprojeto de sistema que pode ser necessário.
2. **Adaptação ambiental**. Esse tipo de manutenção é necessário quando algum aspecto do ambiente do sistema, como o hardware, a plataforma do sistema operacional ou outro software de apoio sofre uma mudança. O sistema de aplicação deve ser modificado para se adaptar a essas mudanças de ambiente.
3. **Adição de funcionalidade**. Esse tipo de manutenção é necessário quando os requisitos de sistema mudam em resposta às mudanças organizacionais ou de negócios. A escala de mudanças necessárias para o software é, frequentemente, muito maior do que para os outros tipos de manutenção.

Na prática, não existe uma distinção clara entre esses tipos de manutenção. Ao adaptar o sistema a um novo ambiente, pode-se adicionar funcionalidade para tirar proveito de novas características do ambiente. Os defeitos de software são frequentemente expostos porque os usuários usam o sistema de formas inesperadas. Mudar o sistema para acomodar sua maneira de trabalhar é a melhor maneira de corrigir tais defeitos.

As pesquisas em geral concordam que a manutenção de software ocupa uma proporção maior dos orçamentos de TI que o desenvolvimento (a manutenção detém, aproximadamente, dois terços do orçamento, contra um terço para desenvolvimento). Elas também concordam que se gasta mais do orçamento de manutenção na implementação de novos requisitos do que na correção de bugs. A Figura 4 mostra, aproximadamente, a distribuição dos custos de manutenção. As porcentagens específicas variam de uma organização para outra, mas, universalmente, a correção de defeitos de sistema não é a atividade de manutenção mais cara. Evoluir o sistema para lidar com novos ambientes e novos ou alterados requisitos consome mais esforço de manutenção.

Figura 4



Geralmente, é mais caro adicionar funcionalidade depois que um sistema está em operação do que implementar a mesma funcionalidade durante o desenvolvimento. As razões para isso são:

1. **Estabilidade da equipe**. Depois de um sistema ter sido liberado, é normal que a equipe de desenvolvimento seja desmobilizada e as pessoas sejam remanejadas para novos projetos. A nova equipe ou as pessoas responsáveis pela manutenção do sistema não entendem o sistema ou não entendem a estrutura para tomar as decisões de projeto. Antes de implementar alterações é preciso investir tempo em compreender o sistema existente.
2. **Más práticas de desenvolvimento**. O contrato para a manutenção de um sistema é geralmente separado do contrato de desenvolvimento do sistema. O contrato de manutenção pode ser dado a uma empresa diferente da do desenvolvedor do sistema original. Esse fator, juntamente com a falta de estabilidade da equipe, significa que não há incentivo para a equipe de desenvolvimento escrever um software manutenível. Se uma equipe de desenvolvimento pode cortar custos para poupar esforço durante o desenvolvimento, vale a pena fazê-lo, mesmo que isso signifique que o software será mais difícil de mudar no futuro.
3. **Qualificações de pessoal**. A equipe de manutenção é relativamente inexperiente e não familiarizada com o domínio de aplicação. A manutenção tem uma imagem pobre entre os engenheiros de software. É vista como um processo menos qualificado do que o desenvolvimento de sistema e é muitas vezes atribuída ao pessoal mais jovem. Além disso, os sistemas antigos podem ser escritos em linguagens obsoletas de programação. A equipe de manutenção pode não ter muita experiencia de desenvolvimento nessas linguagens e precisa primeiro aprender para depois manter o sistema.
4. **Idade do programa e estrutura**. Com as alterações feitas no programa, sua estrutura tende a degradar. Consequentemente, como os programas envelhecem, tornam-se mais difíceis de serem entendidos e alterados. Alguns sistemas foram desenvolvidos sem técnicas modernas de engenharia de software. Eles podem nunca ter sido bem-estruturados e talvez tenham sido otimizados para serem mais eficientes do que inteligíveis. As documentações de sistema podem ter-se perdido ou ser inconsistentes. Os sistemas mais antigos podem não ter sido submetidos a

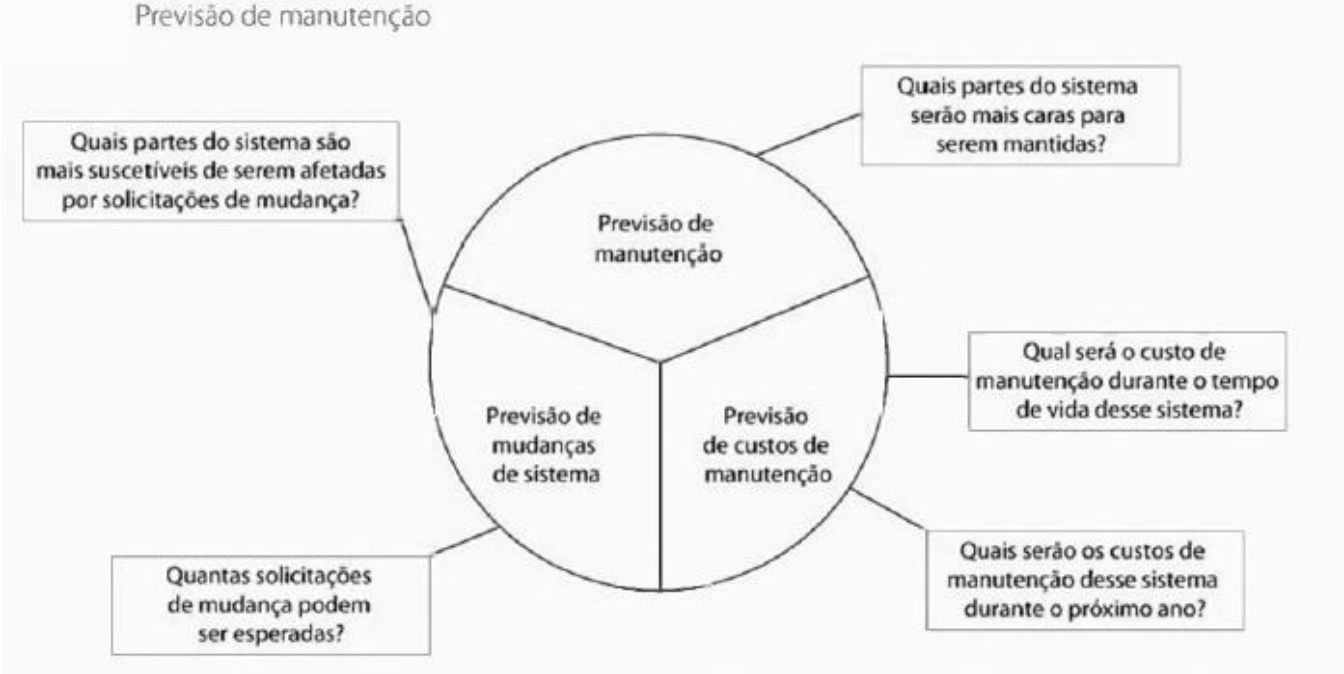
um gerenciamento rigoroso de configuração, então se desperdiça muito tempo para encontrar as versões certas dos componentes do sistema para a mudança.

Previsão de Manutenção

Prever o número de solicitações de mudança para um sistema requer uma compreensão do relacionamento entre o sistema e seu ambiente externo. Alguns sistemas possuem um relacionamento muito complexo com seu ambiente externo, e as mudanças nesse ambiente inevitavelmente resultam em alterações. Para avaliar os relacionamentos entre um sistema e seu ambiente, deve-se avaliar:

1. O número e o complexidade das interfaces de sistema. Quanto maior o número de interfaces e mais complexas elas forem, maior a probabilidade de serem exigidas as alterações de interface quando novos requisitos forem propostos.
2. O número de requisitos inerentemente voláteis de sistema. Os requisitos que refletem as políticas e procedimentos organizacionais são provavelmente mais voláteis do que requisitos baseados em características estáveis de domínio.
3. Os processos de negócio em que o sistema é usado. Como processos de negócios evoluem, eles geram solicitações de mudança de sistema. Quanto mais processos de negócios usarem um sistema, maior a demanda por mudanças.

Figura 5



Reengenharia de software

O processo de evolução de sistema envolve a compreensão do programa que tem de ser mudado e, em seguida, a implementação dessas mudanças. No entanto, muitos sistemas, especialmente sistemas legados mais velhos, são difíceis de serem compreendidos e mudados. Para fazer com que os sistemas legados de software sejam mais fáceis de serem mantidos, é preciso aplicar reengenharia nesses sistemas visando a melhoria de sua estrutura e inteligibilidade. A reengenharia pode envolver a redocumentação de sistema, a refatoração da arquitetura de sistema, a mudança de linguagem de programação para uma linguagem moderna e modificações e atualizações da estrutura e dos dados de sistema.

Existem dois benefícios importantes na reengenharia, em vez de substituição:

1. **Risco reduzido**. Existe um alto risco em desenvolver novamente um software crítico de negócios. Podem ocorrer erros na especificação de sistema ou pode haver problemas de desenvolvimento. Atrasos no início do novo software podem significar a perda do negócio e custos adicionais.
2. **Custo reduzido**. O custo de reengenharia pode ser significativamente menor do que o de desenvolvimento de um novo software. Ulrich (1990) cita um exemplo de um sistema comercial cujos custos de reimplementação foram estimados em 50 milhões de dólares. O sistema foi reconstruído com sucesso por 12 milhões de dólares.

Segundo Somerville, com a tecnologia moderna de software, o custo relativo de reimplementação é provavelmente inferior a esse, mas ainda consideravelmente superior aos custos da reengenharia.

A Figura 6 é um modelo geral de processo de reengenharia. A entrada para o processo é um programa legado, e a saída, uma versão melhorada e reestruturada do mesmo programa. As atividades desse processo de reengenharia são:

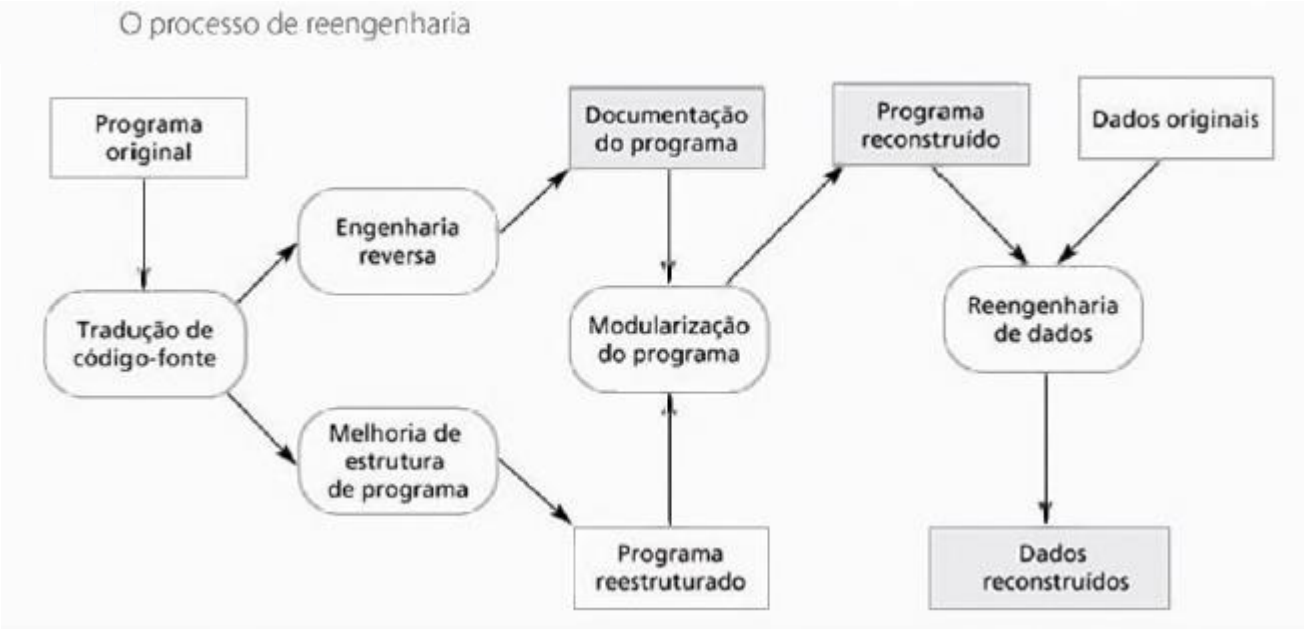
1. **Tradução de código-fonte**. Usando uma ferramenta de tradução, o programa é convertido a partir de uma linguagem de programação antiga para uma versão mais moderna da mesma linguagem ou em outra diferente.
2. **Engenharia reversa**. 0 programa é analisado e as informações são extraídas a partir dele. Isso ajuda a documentar sua organização e funcionalidade. Esse processo também é completamente automatizado.
3. **Melhoria de estrutura de programa**. A estrutura de controle do programa é analisada e modificada para que se torne mais fácil de ler e entender. Isso pode ser parcialmente automatizado, mas, normalmente, alguma intervenção manual é exigida.
4. **Modularização de programa**. Partes relacionadas do programa são agrupadas, e onde houver redundância, se apropriado, esta é removida. Em alguns casos, esse estágio pode envolver refatoração de arquitetura (por exemplo, um sistema que usa vários repositórios de dados diferentes pode ser refeito para usar um único repositório). Esse é um processo manual.
5. **Reengenharia de dados**. Os dados processados pelo programa são alterados para refletir as mudanças de programa. Isso pode significar a redefinição dos esquemas de banco de dados e a conversão do banco de dados existente para a nova estrutura. Normalmente devem-se limpar os dados, o que envolve encontrar e corrigir erros, remover registros duplicados etc. Ferramentas são disponíveis para dar suporte a reengenharia de dados.

A reengenharia de programa pode não exigir necessariamente todas as etapas da Figura 6. Caso se utilize o ambiente de desenvolvimento da linguagem de programação, não é necessário tradução do código-fonte. Se puder fazer automaticamente a reengenharia, a recuperação de documentação por meio da engenharia reversa pode ser desnecessária. A reengenharia de dados só e necessária se as estruturas de dados de programa mudarem durante a reengenharia de sistema.

O problema com a reengenharia de software é que existem limites práticos para o quanto se pode melhorar um sistema por meio da reengenharia. Não é possível, por exemplo, converter um sistema escrito por meio de uma abordagem funcional para um sistema orientado a objetos. As principais mudanças de arquitetura ou a reorganização radical do sistema de gerenciamento de dados não podem

ser feitas automaticamente, pois são muito caras. Embora a reengenharia possa melhorar a manutenibilidade, o sistema reconstruído provavelmente não será tão manutenível como um novo sistema, desenvolvido por meio de métodos modernos de engenharia de software.

Figura 6



Referências

SOMMERVILLE, Ian. Engenharia de software. 10. ed. São Paulo: Pearson, 2019.