INTEGRAÇÃO DE SISTEMAS LEGADOS

Prof. Rodrigo Cantú Polo

CONVERSA INICIAL

Nesta aula, no primeiro tema, serão abordados os conceitos de sistemas e plataformas legadas. No segundo tema, discutiremos sobre a importância de sistemas legados. Na sequência, tema 3, serão descritos os riscos envolvidos na substituição de sistemas legados. No quarto tema, serão discutidos os tipos de sistemas computacionais definidos por Manny Lehman, enquanto que, no quinto e último tema, descreveremos as oitos leis de Lehman, que compreendem a evolução de sistemas.

CONTEXTUALIZANDO

Sistemas legados costumam ser vistos como sistemas críticos para o funcionamento de processos importantes de uma organização. São sistemas concebidos há muito tempo e passaram por um grande processo de evolução durante sua existência para atender seus usuários de forma satisfatória. O significado da palavra “legado”, segundo o dicionário Aurélio (Ferreira, 2001, p. 452), é:

1. Deixar (em legado).
2. Deixar por herança.
3. Transmitir.
4. Enviar como legado.

Em computação, de forma geral, a palavra “legado” para sistemas segue o mesmo conceito, isto é, especifica sistemas que foram “deixados como herança” ou “transmitidos” para outras pessoas trabalharem na sua manutenção.

Com o tempo, este último aspecto (outras pessoas trabalharem na sua manutenção) pode causar vários problemas decorrentes do desconhecimento do seu funcionamento completo, da sua arquitetura, da tecnologia geralmente defasada utilizada no seu desenvolvimento, entre outros motivos. A substituição de tais sistemas costuma ser considerada. Todavia, em geral, eles são críticos e complexos, de modo que não compensa substituí-los.

No tema a seguir, você conhecerá algumas definições sobre sistemas legados.

TEMA 1 – DEFINIÇÃO DE SISTEMAS LEGADOS

Para Sommerville (2011), sistemas legados são sistemas antigos ainda úteis e, algumas vezes, críticos para a operação de negócios. Podem ter sido implementados com o uso de linguagens e tecnologias antigas, ou se utilizar de outros sistemas legados. Normalmente, sua estrutura se encontra degradada pelas mudanças ocorridas ao longo do tempo e sua documentação está perdida ou desatualizada. Em geral, são sistemas críticos para o negócio da organização, e são mantidos porque sua substituição é arriscada demais.

Para Feathers (2004), um código legado que compõe um sistema legado consiste em todo o código que não possui testes automatizados. Os testes automatizados, por sua vez, garantem que o código de um sistema esteja funcionando, de modo que são essenciais quando o código sofre modificações. Se um conjunto de modificações causar algum defeito previsto por um teste automatizado, tal defeito será indicado pelo teste. Isso faz com que os desenvolvedores tenham mais confiança em suas modificações do sistema, principalmente os integrantes mais novos, que ainda estão aprendendo o seu funcionamento.

Os sistemas legados, muitas vezes, são executados em sistemas operacionais obsoletos, chamados de plataformas legadas, como descrito a seguir.

1.1 Definição de plataformas legadas

Uma plataforma legada costuma ser um sistema operacional que não mais está em utilização, que foi substituído por uma versão mais nova, ou, ainda, por uma tecnologia mais moderna. Pode ser também, além do sistema operacional, o conjunto de compiladores, drivers e programas de suporte para manter o sistema legado. Muitas organizações têm plataformas legadas que suportam sistemas críticos. Com o lançamento do *Windows 10*, por exemplo, o *Windows 8* se tornou uma plataforma legada. Antes, o *Windows 8* tornou o *Windows 7* uma plataforma legada e assim sucessivamente.

Pode ser difícil, ou até mesmo impossível, executar novas aplicações em plataformas legadas. Em alguns casos, uma nova versão de um programa pode funcionar em um sistema operacional mais antigo, mas sem garantias de que algum efeito indesejado não possa acontecer. Não é incomum que a versão nova

de um programa funcione parcialmente, ou simplesmente não funcione em um sistema operacional legado. O computador pode até parar de funcionar quando a nova versão do programa é executada. Em geral, é mais provável encontrar problemas deste tipo à medida que a diferença de idade entre a aplicação e o sistema operacional aumenta.

Historicamente, aplicações foram desenvolvidas para sistemas operacionais de fabricantes específicos. Atualmente, muitas companhias estão migrando suas aplicações para novas linguagens de programação e sistemas operacionais que seguem padrões abertos, ou amplamente utilizados. A intenção é possibilitar a atualização de programas sem precisar reimplementá-los totalmente, e fazer com que qualquer organização possa utilizar as suas aplicações em qualquer sistema operacional.

No próximo tema, segue uma descrição sobre a importância de sistemas legados para as organizações.

TEMA 2 – IMPORTÂNCIA DE SISTEMAS LEGADOS

Por conta do tempo e do esforço requeridos para desenvolver um sistema complexo, grandes sistemas computacionais costumam durar muito tempo. Por exemplo, sistemas militares são, em geral, desenvolvidos para durar vinte anos, e grande parte do controle de tráfego aéreo mundial ainda é baseado em *software*s e processos operacionais originalmente desenvolvidos nas décadas de 1960 e 1970. Algumas vezes, é muito caro e arriscado descartar sistemas como estes depois de alguns anos de uso. Seu desenvolvimento continua com mudanças para acomodar novos requisitos, novas plataformas etc.

Muitos bancos se sustentam em sistemas críticos feitos no fim dos anos setenta e no começo dos anos oitenta. Toda a vez que um cliente de um banco realiza uma transferência bancária, a transação passa por um sistema legado. Quanto ao risco de substituição de algum sistema como esse, considere, por exemplo, o sistema de contas de clientes, um dos primeiros sistemas bancários. Políticas organizacionais e procedimentos podem depender deste tipo de sistema. Se algum banco fosse substituir um sistema como esse, que pode estar sendo executado por um mainframe, um sério problema poderia acontecer se a substituição não fosse feita de forma apropriada. Procedimentos de negócio teriam de ser modificados, e isso poderia causar vários problemas para os funcionários e os auditores do banco.

Os sistemas SCADA (**S**upervisory **C**ontrol **A**nd **D**ata **A**cquisition) são utilizados na supervisão e no controle de concessionárias de energia elétrica, distribuição de água, gás, processos industriais, entre outros. As concessionárias de energia elétrica, por exemplo, costumam ter versões antigas em operação nas subestações. Isso acontece porque o custo e o risco de atualizações nestes sistemas costumam ser altos e, muitas vezes, não compensa atualizá-los, até porque os equipamentos da subestação com os quais estes sistemas se comunicam não são atualizados com frequência, podendo até ter décadas de idade.

O programa do ônibus espacial da NASA utilizou muita tecnologia da década de setenta. A substituição dos equipamentos não era uma opção por conta do alto custo da certificação para habilitação de voos. O hardware original completou o requerimento de certificação para voos, mas qualquer novo equipamento teria de passar por todo o procedimento novamente. Tal processo, longo e detalhado, requeria testes extensivos em novos componentes, antes que algum deles pudesse ser utilizado no programa do ônibus espacial. Além disso, todo o sistema do programa, incluindo veículos de terra e de lançamento, foi feito para que os seus componentes, em totalidade, trabalhassem com um sistema fechado. Como as especificações não mudavam, todos os sistemas e componentes certificados se saíram bem nos papéis a eles projetados. Mesmo depois que o programa foi desativado, em 2010, a NASA acreditou que compensava continuar utilizando várias peças da tecnologia da década de setenta, em vez de atualizar os sistemas e refazer a certificação dos novos componentes.

Como foi comentado anteriormente, substituir ou atualizar um sistema legado não costuma ser uma tarefa fácil. Algumas vezes, simplesmente não compensa efetuar a atualização ou substituição. A seguir, são comentados os motivos pelos quais estes sistemas possuem tais características.

TEMA 3 – RISCOS DE SUBSTITUIÇÃO DE SISTEMAS LEGADOS

Organizações atualizam seus ativos de TI por novos sistemas regularmente. Todavia, retirar sistemas legados de operação e substituí-los por *software*s mais modernos envolve riscos significativos ao negócio. Muitos gerentes tentam minimizar os riscos, evitando incertezas no funcionamento de

novos sistemas. Substituir um sistema legado é uma estratégia arriscada pelos seguintes motivos:

- Raramente existe uma especificação completa do sistema legado. A especificação original, na maioria dos casos, foi perdida. Se uma especificação existe, é improvável que incorpore detalhes de todas as mudanças de sistema feitas ao longo do tempo. Logo, não há uma forma simples de especificar um novo sistema que seja funcionalmente idêntico ao sistema em uso.
- Os processos de negócio e as formas como seus sistemas legados operam estão, quase sempre, fortemente ligados. Estes processos foram feitos para se beneficiarem dos serviços de tais sistemas, evitando suas fraquezas. Se um sistema for substituído, seus processos também terão de ser modificados, com consequências imprevisíveis para o negócio.
- Regras importantes de negócio podem estar embutidas no *software* e não estar documentadas em lugar algum. Uma regra de negócio é uma validação que se aplica a alguma rotina de processamento de parte do negócio. A retirada dessa validação pode ter consequências imprevisíveis. Por exemplo, uma companhia de seguros pode ter regras embutidas no sistema para avaliar os riscos de uma aplicação de uma nova apólice. Se estas regras não forem mantidas, a companhia pode, inadvertidamente, aceitar apólices de alto risco, o que possibilita resultar em altas coberturas no futuro.
- Desenvolvimento de novos sistemas costuma ser um processo arriscado, com alta probabilidade de ocorrência de problemas inesperados. Um novo sistema pode não ser entregue no prazo, ou o seu preço final pode acabar sendo mais alto que o estimado inicialmente.

Apesar dos riscos envolvidos na substituição desses sistemas, os custos elevados de manutenção podem acelerar seu processo. Sistemas legados com mais de alguns anos de idade são caros de manter pelos seguintes motivos:

1. Diferentes partes do sistema podem ter sido implementadas por times diferentes, sem um estilo de programação, o que dificulta o entendimento do código.
2. Parte do sistema (ou todo ele) pode ter sido implementado com linguagens de programação obsoletas. Pode ser difícil de achar pessoas que tenham

conhecimento de tais linguagens, e a terceirização desse trabalho, que tende a ser cara, será requerida.

3. Documentação do sistema está frequentemente inadequada e desatualizada. Em determinados casos, a única documentação é o código fonte. Algumas vezes, o código fonte pode ter sido perdido, e apenas os binários do sistema estão disponíveis.
4. Muitos anos de manutenção podem ter corrompido a estrutura do sistema, tornando-o difícil de entender. Novos programas podem ter sido adicionados e integrados com outras partes do sistema de forma não convencional.
5. O sistema pode ter sido otimizado para uso de espaço de armazenamento, ou de velocidade de execução, em detrimento de inteligibilidade. Isso causa dificuldades para programadores que aprenderam técnicas de engenharia de *software* modernas, e que não têm a vivência de truques de programação, utilizados antigamente.
6. Os dados processados pelo sistema podem estar mantidos em vários arquivos diferentes, com estruturas incompatíveis. É possível haver duplicação de dados, que podem estar desatualizados, incompletos ou incorretos.

TEMA 4 – SISTEMAS *P*, *S* E *E*

A dinâmica de evolução de programas é o estudo das mudanças de sistema. Nas décadas de 70 e 80, Lehman e Belady (1985) fizeram vários estudos empíricos de mudanças de sistema para entender melhor as características da evolução de *software*. O trabalho continuou nos anos 90. Lehman e colegas investigaram o significado da realimentação na evolução de sistemas. Desses estudos, Lehman categoriza todos os sistemas de *software* como parte de um de três tipos mapeados: tipo **S**, tipo **P** ou tipo **E**.

4.1 Sistemas do tipo *S*

Sistemas do tipo S, ou sistemas de tipo estático, podem ser descritos formalmente em um conjunto de especificações cuja solução é bem compreendida. Essencialmente, estes tipos de sistema permitem que as pessoas envolvidas entendam o problema e saibam exatamente o que é necessário para resolvê-lo. A parte estática destes sistemas se refere às suas especificações, que

não mudam. Alguns exemplos: programas de calculadora simples, AIs que encapsulam protocolos de comunicação, sistemas de cálculo de amostragens de normas industriais etc. Sistemas desse tipo são, em geral, os mais simples dos três tipos e são menos sujeitos a forças evolucionárias.

Figura 1 – Sistema do tipo *S*

Fonte: Pfleeger (2004).

4.2 Sistemas do tipo *P*

Nem sempre é fácil ou possível descrever completamente um problema do mundo real. Em muitos casos, a solução teórica existe, mas é impraticável ou impossível de ser implementada. Logo, sistemas do tipo P, ou sistemas de tipo prático são, em essência, sistemas em que o problema é conhecido, mas que não se conhecem formas precisas de resolvê-lo. Sistemas deste tipo requerem uma aproximação iterativa em sua descoberta na identificação de características relevantes do problema, para que seja possível encontrar a melhor solução. Exemplos deste tipo de sistema são lógicas de inteligência artificial para identificar qual é a melhor jogada a ser feita em um jogo de xadrez. Utilizando a tecnologia atual, é impossível implementar essa solução de forma completa. O número de movimentos é muito grande para ser avaliado em um tempo aceitável. Logo, deve-se desenvolver uma solução aproximada, que seja prática de se construir e utilizar.

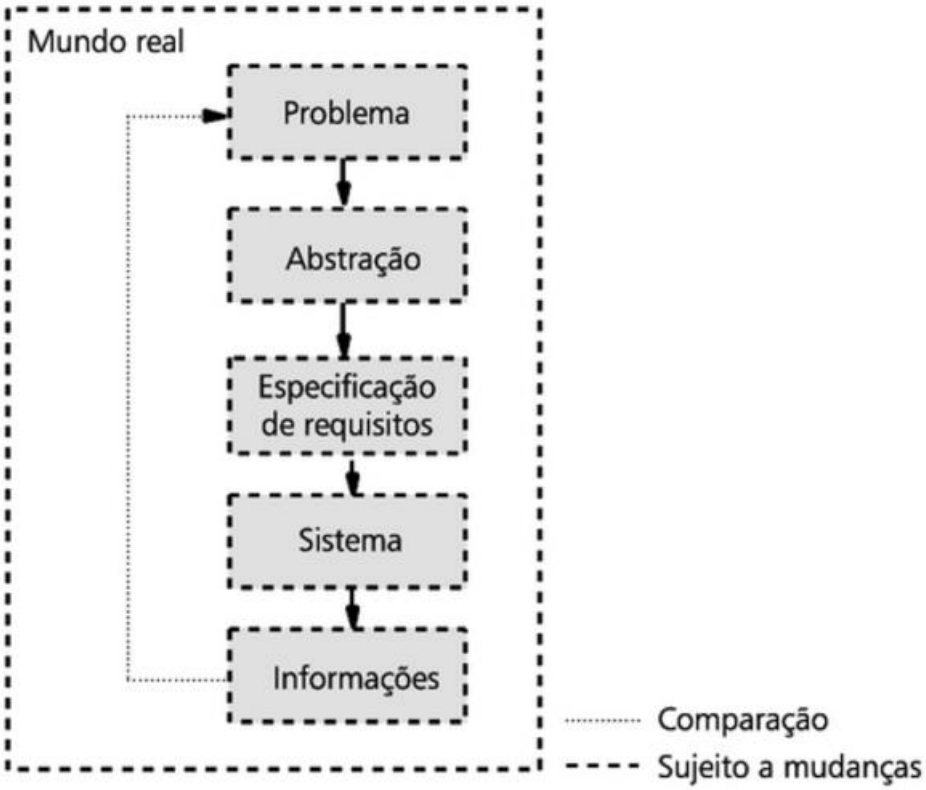
Figura 2 – Sistema do tipo *P*

Fonte: Pfleeger (2004).

4.3 Sistemas do tipo *E*

Sistemas do tipo E, ou sistemas embutidos, são os que caracterizam a grande maioria dos sistemas em uso atualmente. O termo “embutido” se refere à noção de que estes sistemas servem para modelar processos do mundo real e que, pelo seu uso, se tornam componentes do mundo, o qual têm a intenção de modelar. Em outras palavras, de forma oposta a um simples programa de calculadora (um exemplo de sistema do tipo S), sistemas deste tipo se tornam componentes de processos do mundo real, que falhariam sem sua existência. Um exemplo de sistema fortemente embutido no mundo real é o de controle de tráfego aéreo de um aeroporto. Sem um sistema como este, a tarefa de organizar o pouso e a decolagem seguros de aviões seria impossível. Nesse exemplo, fica bem claro como o sistema se tornou um componente do mundo real. Como o mundo real está em constante mudança, estes sistemas devem continuar sendo atualizados para, portanto, continuarem úteis.

Figura 3 – Sistema do tipo *E*



Fonte: Pfleeger (2004).

Os sistemas do tipo E têm alto grau de evolução ao longo do tempo. Para este tipo de sistemas, Lehman formulou oito leis de evolução, as quais serão descritas no próximo tema. Sistemas do tipo S são sistemas estáticos, em que o problema e a sua solução não se modificam ao longo do tempo. Como este tipo de sistema não evolui, as leis de Lehman não se aplicam a eles. O sistema P também não está sujeito às leis de Lehman porque a solução para o problema a ser por eles resolvido não é especificada formalmente e pode variar, inclusive em função da capacidade de processamento do computador no qual o sistema é executado.

TEMA 5 – LEIS DE LEHMAN

Com a observação da evolução de grandes sistemas computacionais, Lehman e Belady propuseram um conjunto de leis, conhecidas como leis de Lehman, que abrangem mudanças em sistemas do tipo E.

Lehman e Belady argumentam que estas leis são, em geral, verdadeiras para todos os tipos de grandes sistemas organizacionais. São sistemas em que os requisitos mudam para refletir necessidades de negócio. Novas entregas são essenciais para que o sistema possa prover valor de negócio.

Quadro 1 – Leis de Lehman

Lei	Descrição
Mudança contínua	Um programa usado em um ambiente do mundo real deve necessariamente mudar, ou se torna progressivamente menos útil neste ambiente.
Aumento da complexidade	Como um programa em evolução muda, sua estrutura tende a se tornar mais complexa. Recursos extras devem ser dedicados a preservar e simplificar a estrutura.
Evolução de programa de grande porte	A evolução de programa é um processo de autorregulação. Atributos de sistema, como tamanho, tempo entre entregas e número de erros relatados são aproximadamente invariáveis para cada entrega do sistema.
Estabilidade organizacional	Ao longo da vida de um programa, sua taxa de desenvolvimento é aproximadamente constante e independe dos recursos destinados ao desenvolvimento do sistema.
Conservação da familiaridade	Durante a vigência de um sistema, a mudança incremental em cada entrega é aproximadamente constante.
Crescimento contínuo	A funcionalidade oferecida pelos sistemas tende a 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 é preciso tratá-los como sistemas de feedback para alcançar significativa melhoria do produto.

A **primeira lei** (mudança contínua) especifica que a manutenção de sistemas é um processo inevitável. À medida que o ambiente do sistema muda, novos requisitos emergem e o sistema deve ser modificado. Quando o sistema modificado é reintroduzido no ambiente, acabam surgindo novas mudanças de ambiente, e o processo de evolução recomeça.

A **segunda lei** (aumento da complexidade) especifica que, à medida que um sistema é modificado, sua estrutura tende a ficar degradada. Isso se aplica a todos os tipos de sistema, não só os de *software*, e ocorre porque uma estrutura feita para determinado propósito está sendo adaptada para um propósito diferente. A degradação, se não for tratada, faz com que seja cada vez mais difícil efetuar mudanças no programa. Pequenas mudanças reduzem a extensão da degradação estrutural, minimizando os riscos de ocorrência de sérios problemas no sistema. Grandes mudanças aumentam a probabilidade de ocorrência de novas falhas, inibindo futuras alterações. A única forma de evitar isso é investir em manutenção preventiva, melhorando a estrutura do *software* sem adicionar novas funcionalidades. Isto significa custos adicionais, além dos de desenvolvimento de novas funcionalidades de sistema.

A **terceira lei** (evolução de programa de grande porte) sugere que grandes sistemas têm uma dinâmica própria, estabelecida no início do processo de desenvolvimento. Isso determina a tendência da maior parte do processo de manutenção e limita o número de possíveis mudanças no sistema. Os fatores estruturais, que afetam a terceira lei, vêm da complexidade de grandes sistemas, citado pela segunda lei.

A **quarta lei** (estabilidade organizacional) sugere que a maior parte dos projetos de desenvolvimento ocorre de forma que mudanças administrativas, ou no time de desenvolvimento, têm efeitos imperceptíveis na evolução a longo prazo do sistema. Isso é consistente com a terceira lei, que considera a evolução de programas largamente independente de decisões de gerenciamento.

A **quinta lei** (conservação da familiaridade) se refere a incrementos de mudança em cada entrega do sistema. A adição de novas funcionalidades em um sistema inevitavelmente introduz novas falhas. Para cada nova funcionalidade adicionada, mais falhas surgirão. Um grande incremento de funcionalidades em uma versão significa que uma nova versão deve ser liberada com a correção dos defeitos encontrados (ao menos, os mais críticos). Esta lei sugere que não se deve liberar grandes incrementos de funcionalidade em uma versão sem considerar a necessidade de correção das falhas encontradas.

A **sexta lei** (crescimento contínuo) descreve que o tamanho de um sistema deve aumentar durante o seu ciclo de vida para que continue relevante para os problemas de negócio que se propõe a resolver. Esta lei está relacionada à segunda lei, em que existe um relacionamento direto entre a quantidade de funcionalidades de um sistema e a complexidade requerida para suportá-las. Esta lei reafirma a segunda de forma que, sem o devido cuidado, a expansão do sistema pode ter efeitos negativos na sua compreensão, afetando a sua capacidade de evolução.

A **sétima lei** (declínio de qualidade) diz que modificações feitas com baixa qualidade tendem a introduzir defeitos e atendimento parcial a especificações de requisitos. Sem um esforço direto e rigoroso em preservar a integridade estrutural do sistema, minimizando os efeitos que a evolução contínua terá no seu aumento de tamanho, complexidade e perda de inteligibilidade, o resultado será o declínio da sua qualidade.

A **oitava** e última **lei** (sistema de feedback) descreve que, para que a evolução contínua de um sistema seja sustentável, deve haver formas de monitorar o seu desempenho. Esta lei se refere à importância de se obter métricas

de desempenho de manutenção do sistema. Mesmo que os valores nominais não tenham valor aparente, a sua análise de tendência ao longo do tempo pode indicar a forma como a evolução do sistema está se comportando.

As observações de Lehman devem ser consideradas no planejamento do processo de manutenção de um sistema legado. Porém, existe a possibilidade de as considerações do negócio ignorarem tais observações. Por exemplo, por razões de marketing, pode ser necessário fazer várias mudanças no sistema em uma única versão. As consequências de uma ação como esta são que uma ou mais versões subsequentes podem ser necessárias para a correção de defeitos. Isso geralmente é observado em *software*s para computação pessoal, em que uma grande versão de determinada aplicação tipicamente é seguida de uma grande atualização de correção de falhas.

FINALIZANDO

Nesta aula, foram descritos os conceitos de sistemas e de plataformas legadas, estas que costumam se referir a sistemas operacionais legados. A importância de sistemas legados também foi abordada, com exemplos de alguns ainda utilizados atualmente na área militar, no controle de tráfego aéreo, em bancos, sistemas SCADA e exploração espacial. Comentamos sobre os riscos de substituição destes sistemas, assim como os seus elevados custos de manutenção ao longo do tempo. Foram descritos os tipos de sistemas de larga escala, sendo eles os tipos *S* (estático), *P* (prático) e *E* (embutido com o mundo real), de acordo com Lehman. Por fim, abordamos as leis de Lehman, sobre a evolução de sistemas do tipo *E*. É importante entender as definições sobre os tipos de sistemas *E*, bem como suas leis de evolução, para entender como os sistemas evoluem e se tornam sistemas legados com o passar do tempo.

REFERÊNCIAS

FEATHERS, M. C. **Working effectively with legacy code**. São Paulo: Pearson Prentice Hall, 2004.

FERREIRA, A. B. de H. Legado. **Aurélio**: o dicionário da língua portuguesa. Rio de Janeiro: Nova Fronteira, 2001.

HEIN, A. M. How to assess heritage systems in the early phases? **Secesa**, 2014. Disponível em: <https://www.academia.edu/8441540/How_to_Assess_Heritage_Systems_in_the_Early_Phases>. Acesso em: 14 set. 2017.

KARCH, E. Lehman's Laws of *software* evolution and the staged-model. **Karchworld Identity**, 11 abr. 2011. Disponível em: <https://blogs.msdn.microsoft.com/karchworld_identity/2011/04/01/lehmans-laws-of-*software*-evolution-and-the-staged-model/>. Acesso em: 14 set. 2017.

LEGACY platform (legacy operating system). **WhatIs.com**. Disponível em: <http://whatis.techtarget.com/definition/legacy-platform-legacy-operating-system>. Acesso em: 14 set. 2017.

PFLEEGER, S. L. **Engenharia de *software*:** teoria e prática. 2. ed. São Paulo: Pearson Prentice Hall, 2004.

SISTEMAS legados. Disponível em: <https://ifs.host.cs.st-andrews.ac.uk/Books/SE9/Web/LegacySys/>. Acesso em: 14 set. 2017.

SOMMERVILLE, I. **Engenharia de *software*.** Trad. Kalinka Oliveira e Ivan Bosnic. 9. ed. São Paulo: Pearson Prentice Hall, 2011.