# ersionamento Semântico 2.0.0

## Sumário

Dado um número de versão MAJOR.MINOR.PATCH, incremente a:

1. versão Maior(MAJOR): quando fizer mudanças incompatíveis na API,
2. versão Menor(MINOR): quando adicionar funcionalidades mantendo compatibilidade, e
3. versão de Correção(PATCH): quando corrigir falhas mantendo compatibilidade.

Rótulos adicionais para pré-lançamento(pre-release) e metadados de construção(build) estão disponíveis como extensão ao formato MAJOR.MINOR.PATCH.

## Introdução

No mundo de gerenciamento de software existe algo terrível conhecido como inferno das dependências (“dependency hell”). Quanto mais o sistema cresce, e mais pacotes são adicionados a ele, maior será a possibilidade de, um dia, você encontrar-se neste poço de desespero.

Em sistemas com muitas dependências, lançar novos pacotes de versões pode se tornar rapidamente um pesadelo. Se as especificações das dependências são muito amarradas você corre o risco de um bloqueio de versão (A falta de capacidade de atualizar um pacote sem ter de liberar novas versões de cada pacote dependente). Se as dependências são vagamente especificadas, você irá inevitavelmente ser mordido pela ‘promiscuidade da versão’ (assumindo compatibilidade com futuras versões mais do que é razoável). O inferno das dependências é onde você está quando um bloqueio de versão e/ou promiscuidade de versão te impede de seguir em frente com seu projeto de maneira fácil e segura.

Como uma solução para este problema proponho um conjunto simples de regras e requisitos que ditam como os números das versões são atribuídos e incrementados.

Essas regras são baseadas em, mas não necessariamente limitadas às, bem difundidas práticas comumente em uso tanto em softwares fechados como open-source. Para que este sistema funcione, primeiro você precisa declarar uma API pública. Isto pode consistir de documentação ou ser determinada pelo próprio código. De qualquer maneira, é importante que esta API seja clara e precisa. Depois de identificada a API pública, você comunica as mudanças com incrementos específicos para o seu número de versão. Considere o formato de versão X.Y.Z (Maior.Menor.Correção). Correção de falhas (bug fixes) que não afetam a API, incrementa a versão de Correção, adições/alterações compatíveis com as versões anteriores da API incrementa a versão Menor, e alterações incompatíveis com as versões anteriores da API incrementa a versão Maior.

Eu chamo esse sistema de “Versionamento Semântico”. Sob este esquema, os números de versão e a forma como eles mudam transmitem o significado do código subjacente e o que foi modificado de uma versão para a próxima.

## Especificação de Versionamento Semântico (SemVer)

As palavras-chaves “DEVE”, “NÃO DEVE”, “OBRIGATÓRIO”, “DEVERÁ”, “NÃO DEVERÁ”, “PODEM”, “NÃO PODEM”, “RECOMENDADO”, “PODE” e “OPCIONAL” no presente documento devem ser interpretados como descrito na [RFC 2119] (http://tools.ietf.org/html/rfc2119).

1. Software usando Versionamento Semântico DEVE declarar uma API pública. Esta API poderá ser declarada no próprio código ou existir estritamente na documentação, desde que seja precisa e compreensiva.

2. Um número de versão normal DEVE ter o formato de X.Y.Z, onde X, Y, e Z são inteiros não negativos, e NÃO DEVE conter zeros à esquerda. X é a versão Maior, Y é a versão Menor, e Z é a versão de Correção. Cada elemento DEVE aumentar numericamente. Por exemplo: 1.9.0 -> 1.10.0 -> 1.11.0.

3. Uma vez que um pacote versionado foi lançado(released), o conteúdo desta versão NÃO DEVE ser modificado. Qualquer modificação DEVE ser lançado como uma nova versão.

4. No início do desenvolvimento, a versão Maior DEVE ser zero (0.y.z). Qualquer coisa pode mudar a qualquer momento. A API pública não deve ser considerada estável.

5. Versão 1.0.0 define a API como pública. A maneira como o número de versão é incrementado após este lançamento é dependente da API pública e como ela muda.

6. Versão de Correção Z (x.y.Z | x > 0) DEVE ser incrementado apenas se mantiver compatibilidade e introduzir correção de bugs. Uma correção de bug é definida como uma mudança interna que corrige um comportamento incorreto.

7. Versão Menor Y (x.Y.z | x > 0) DEVE ser incrementada se uma funcionalidade nova e compatível for introduzida na API pública. DEVE ser incrementada se qualquer funcionalidade da API pública for definida como descontinuada. PODE ser incrementada se uma nova funcionalidade ou melhoria substancial for introduzida dentro do código privado. PODE incluir mudanças a nível de correção. A versão de Correção deve ser redefinida para 0(zero) quando a versão Menor for incrementada.

8. Versão Maior X (X.y.z | X > 0) DEVE ser incrementada se forem introduzidas mudanças incompatíveis na API pública. PODE incluir alterações a nível de versão Menor e de versão de Correção. Versão de Correção e Versão Menor devem ser redefinidas para 0(zero) quando a versão Maior for incrementada.

9. Uma versão de Pré-Lançamento (pre-release) PODE ser identificada adicionando um hífen (dash) e uma série de identificadores separados por ponto (dot) imediatamente após a versão de Correção. Identificador DEVE incluir apenas caracteres alfanuméricos e hífen [0-9A-Za-z-]. Identificador NÃO DEVE ser vazio. Indicador numérico NÃO DEVE incluir zeros à esquerda. Versão de Pré-Lançamento tem precedência inferior à versão normal a que está associada. Uma versão de Pré-Lançamento (pre-release) indica que a versão é instável e pode não satisfazer os requisitos de compatibilidade pretendidos, como indicado por sua versão normal associada. Exemplos: 1.0.0-alpha, 1.0.0-alpha.1, 1.0.0-0.3.7, 1.0.0-x.7.z.92.

10. Metadados de construção(Build) PODE ser identificada por adicionar um sinal de adição (+) e uma série de identificadores separados por ponto imediatamente após a Correção ou Pré-Lançamento. Identificador DEVE ser composto apenas por caracteres alfanuméricos e hífen [0-9A-Za-z-]. Identificador NÃO DEVE ser vazio. Metadados de construção PODEM ser ignorados quando se determina a versão de precedência. Assim, duas versões que diferem apenas nos metadados de construção, têm a mesma precedência. Exemplos: 1.0.0-alpha+001, 1.0.0+20130313144700, 1.0.0-beta+exp.sha.5114f85.

11. A precedência refere como as versões são comparadas com cada outra quando solicitado. A precedência DEVE ser calculada separando identificadores de versão em Maior, Menor, Correção e Pré-lançamento, nesta ordem (Metadados de construção não figuram na precedência). A precedência é determinada pela primeira diferença quando se compara cada identificador da esquerda para direita, como se segue: Versões Maior, Menor e Correção são sempre comparadas numericamente. Example: 1.0.0 < 2.0.0 < 2.1.0 < 2.1.1. Quando Maior, Menor e Correção são iguais, a versão de Pré-Lançamento tem precedência menor que a versão normal. Example: 1.0.0-alpha < 1.0.0. A precedência entre duas versões de Pré-lançamento com mesma versão Maior, Menor e Correção DEVE ser determinada comparando cada identificador separado por ponto da esquerda para direita até que seja encontrada diferença da seguinte forma: identificadores consistindo apenas dígitos são comparados numericamente e identificadores com letras ou hífen são comparados lexicalmente na ordem de classificação ASCII. Identificadores numéricos sempre têm menor precedência do que os não numéricos. Um conjunto maior de campos de pré-lançamento tem uma precedência maior do que um conjunto menor, se todos os identificadores anteriores são iguais. Example: 1.0.0-alpha < 1.0.0-alpha.1 < 1.0.0-alpha.beta < 1.0.0-beta < 1.0.0-beta.2 < 1.0.0-beta.11 < 1.0.0-rc.1 < 1.0.0.

# Por que usar Versionamento Semântico?

Esta não é uma ideia nova ou revolucionária. De fato, você provavelmente já faz algo próximo a isso. O problema é que “próximo” não é bom o bastante. Sem a aderência a algum tipo de especificação formal, os

números de versão são essencialmente inúteis para gerenciamento de dependências. Dando um nome e definições claras às ideias acima, fica fácil comunicar suas intenções aos usuários de seu software. Uma vez que estas intenções estão claras, especificações de dependências flexíveis (mas não tão flexíveis) finalmente podem ser feitas.

Um exemplo simples vai demonstrar como o Versionamento Semântico pode fazer do inferno de dependência uma coisa do passado. Considere uma biblioteca chamada “CaminhaoBombeiros”. Ela requer um pacote versionado dinamicamente chamado “Escada”. Quando CaminhaoBombeiros foi criado, Escada estava na versão 3.1.0. Como CaminhaoBombeiros utiliza algumas funcionalidades que foram inicialmente introduzidas na versão 3.1.0, você pode especificar, com segurança, a dependência da Escada como maior ou igual a 3.1.0 porém menor que 4.0.0. Agora, quando Escada versão 3.1.1 e 3.2.0 estiverem disponíveis, você poderá lança-los ao seu sistema de gerenciamento de pacote e saberá que eles serão compatíveis com os softwares dependentes existentes.

Como um desenvolvedor responsável você irá, é claro, querer certificar-se que qualquer atualização no pacote funcionará como anunciado. O mundo real é um lugar bagunçado; não há nada que possamos fazer quanto a isso senão sermos vigilantes. O que você pode fazer é deixar o Versionamento Semântico lhe fornecer uma maneira sensata de lançar e atualizar pacotes sem precisar atualizar para novas versões de pacotes dependentes, salvando-lhe tempo e aborrecimento.

Se tudo isto soa desejável, tudo que você precisar fazer para começar a usar Versionamento Semântico é declarar que você o esta usando e então, seguir as regras. Adicione um link para este website no seu README para que outros saibam as regras e possam beneficiar-se delas.

# FAQ