

Semantic Versioning versus Breaking Changes: A Study of the Maven Repository

MD 輪講

博士後期課程 3 年 楊 嘉晨

大阪大学大学院 コンピュータサイエンス専攻 楠本研究室

2015 年 05 月 28 日 (木)

- 1 背景
 - 著者情報と出典
 - セマンティックバージョニング
 - SemVer と API の後方互換性

- 2 Research Questions

- 3 調査手法

- 4 統計的な結果

- 5 調査項目の結果

- 6 議論と妥当性への脅威

- 7 結論

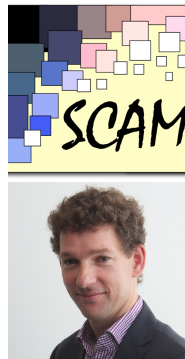
Semantic Versioning versus Breaking Changes: A Study of the Maven Repository¹

出典 SCAM 2014, Victoria, Canada

著者 Steven Raemaekers†,
Arie van Deursen‡,
Joost Visser†

† Software Improvement Group, Amsterdam, The Netherlands

‡ Technical University Delft, The Netherlands



¹ Steven Raemaekers, Arie van Deursen, and Joost Visser. "Semantic versioning versus breaking changes: a study of the Maven Repository". In: *Source Code Analysis and Manipulation (SCAM)*, 2014 IEEE 14th International Working Conference on. IEEE. 2014, pp. 215–224.

セマンティックバージョンング

Background: Semantic Versioning

3.19.2

バージョンナンバーを上げるには、²

major API の変更に **互換性のない** 場合

minor 後方互換性があり **機能性を追加した** 場合

patch 後方互換性を伴う **バグ修正** をした場合

²<http://semver.org/lang/ja/>

セマンティックバージョンング

Background: Semantic Versioning

3.19.2

バージョンナンバーを上げるには、²

major API の変更に **互換性のない** 場合

minor 後方互換性があり **機能性を追加した** 場合

patch 後方互換性を伴う **バグ修正** をした場合

²<http://semver.org/lang/ja/>

セマンティックバージョンング

Background: Semantic Versioning

3.19.2

バージョンナンバーを上げるには、²

major API の変更に **互換性のない** 場合

minor 後方互換性があり **機能性を追加した** 場合

patch 後方互換性を伴う **バグ修正** をした場合

これらルールは既存のソフトウェアに広く使われてあり、全てのソフトウェアに普及すべし

Tom Preston-Werner 氏, Gravatars 及び GitHub の共同創設者

²<http://semver.org/lang/ja/>

セマンティックバージョンング

Background: Semantic Versioning

3.19.2

バージョンナンバーを上げるには、²

major API の変更に **互換性のない** 場合

minor 後方互換性があり **機能性を追加した** 場合

patch 後方互換性を伴う **バグ修正** をした場合

これらルールは既存のソフトウェアに広く使われてあり、全てのソフトウェアに普及すべし

Tom Preston-Werner 氏, Gravatars 及び GitHub の共同創設者

²<http://semver.org/lang/ja/>

セマンティックバージョンング

Background: Semantic Versioning

3.19.2

バージョンナンバーを上げるには、²

major API の変更に **互換性のない** 場合

minor 後方互換性があり **機能性を追加した** 場合

patch 後方互換性を伴う **バグ修正** をした場合

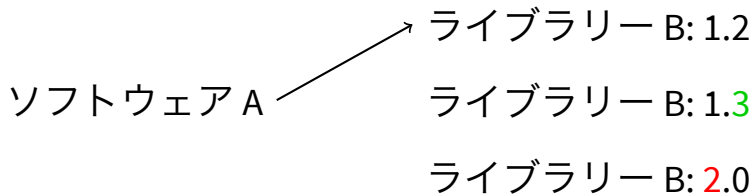
実際、バージョンナンバーは本当にセマンティックバージョンング（以下 SemVer と略称）の原則を守っているか？

²<http://semver.org/lang/ja/>

SemVer と互換性

Semantic Versioning and API Compatibility

ソフトウェアやライブラリーのユーザーにとって、
API の 後方互換性 は重要である



SemVer の原則が守っていれば、安心して依存関係のライブラリーをアップグレードすることができる

SemVer と互換性

Semantic Versioning and API Compatibility

ソフトウェアやライブラリーのユーザーにとって、
API の 後方互換性 は重要である

ライブラリー B: 1.2

ソフトウェア A  ライブラリー B: 1.3

ライブラリー B: 2.0

SemVer の原則が守っていれば、安心して依存関係のライブラリーをアップグレードすることができる

SemVer と互換性

Semantic Versioning and API Compatibility

ソフトウェアやライブラリーのユーザーにとって、
API の 後方互換性 は重要である

ライブラリー B: 1.2

ソフトウェア A ライブラリー B: 1.3

→ ライブラリー B: 2.0

SemVer の原則が守っていれば、安心して依存関係のライブラリーをアップグレードすることができる

SemVer と互換性

Semantic Versioning and API Compatibility

ソフトウェアやライブラリーのユーザーにとって、
API の 後方互換性 は重要である

ライブラリー B: 1.2

ソフトウェア A ライブラリー B: 1.3

→ ライブラリー B: 2.0

SemVer の原則が守っていれば、安心して依存関係のライブラリーをアップグレードすることができる

1 背景

2 Research Questions

- 調査目的と対象
- 調査項目

3 調査手法

4 統計的な結果

5 調査項目の結果

6 議論と妥当性への脅威

7 結論

調査目的と対象

Goal and Research Targets

目的 バージョン番号を上げる際に API 互換性が SemVer 原則に従われているかどうか

対象 Maven 中央リポジトリ³にあるほぼ全ての OSS

Maven 中央リポジトリを対象とする原因

- ・ プロジェクト間の依存関係が記述される
- ・ 一箇所に集まる
- ・ バージョン番号も明記されています

³<http://search.maven.org/>

調査目的と対象

Goal and Research Targets

目的 バージョン番号を上げる際に API 互換性が SemVer 原則に従われているかどうか

対象 Maven 中央リポジトリ³にあるほぼ全ての OSS

Maven 中央リポジトリを対象とする原因

- ・ プロジェクト間の依存関係が記述される
- ・ 一箇所に集まる
- ・ バージョン番号も明記されています

³<http://search.maven.org/>

調査項目

Research Questions

- RQ1 互換性に関する SemVer 原則はどのくらい従われている？
- RQ2 SemVer 原則に従うソフトウェアは時間に亘って増えているか？
- RQ3 新しいバージョンへの依存関係はどう更新されてるか？最新版を使わぬ要因は何がある？
- RQ4 廃止予定 (deprecation) のタグは実際にどう使われているのか？

廃止予定 @deprecated

API 互換性を破る前に、幾つかのバージョンに予め廃止を予定したメソッド等につけるアノテーション

調査項目

Research Questions

- RQ1 互換性に関する SemVer 原則はどのくらい従われている？
- RQ2 SemVer 原則に従うソフトウェアは時間に亘って増えているか？
- RQ3 新しいバージョンへの依存関係はどう更新されてるか？最新版を使わぬ要因は何がある？
- RQ4 廃止予定 (deprecation) のタグは実際にどう使われているのか？

廃止予定 @deprecated

API 互換性を破る前に、幾つかのバージョンに予め廃止を予定したメソッド等につけるアノテーション

1 背景

2 Research Questions

3 調査手法

- 調査手法の概要
- Maven 中央リポジトリ
- API の後方互換性を判断する基準
- バージョン番号の比較
- ソースコードの比較と廃止パターン

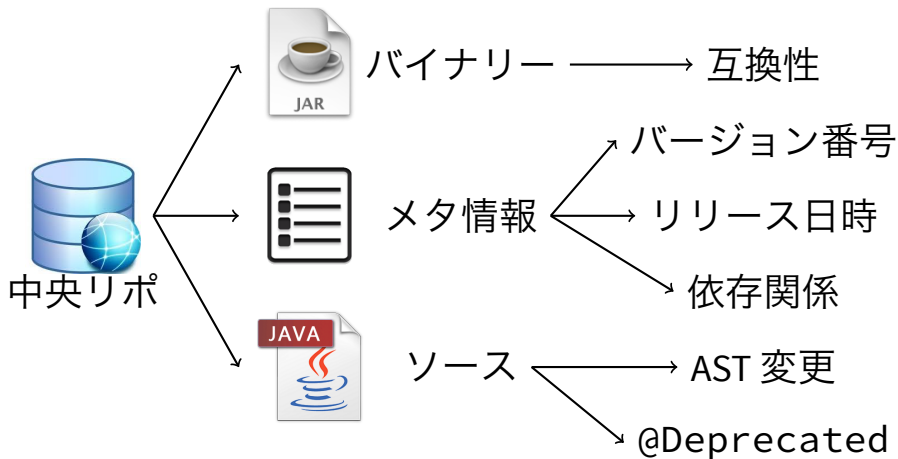
4 統計的な結果

5 調査項目の結果

6 議論と妥当性への脅威

7 結論

調査手法の概要



Maven 中央リポジトリ

2011 年 7 月 11 日の時点の Maven 中央リポジトリのスナップショット⁴を取ってきて⁵、pom.xml を解析し、プロジェクト間に依存関係でリンク付け⁶

	プロジェクト数	バイナリー jar	ソース jar
規模	22,205	148,253	101,413

平均、プロジェクト毎に 6.7 リリース

⁴<http://juliusdavies.ca/2013/j.emse/bertillonage/maven.tar.gz>

⁵ Julius Davies et al. "Software bertillonage: finding the provenance of an entity". In: *Proceedings of the 8th working conference on mining software repositories*. ACM. 2011, pp. 183–192.

⁶ Steven Raemaekers, Arie van Deursen, and Joost Visser. "The maven repository dataset of metrics, changes, and dependencies". In: *Proceedings of the 10th Working Conference on Mining Software Repositories*. IEEE Press. 2013, pp. 221–224.

API の後方互換性を判断する基準

Determining backward incompatible API changes

厳密に API 互換性の有無を判断することは困難

- 意味的に解析や、実際にビルド・テストする必要
代わりに本研究ではバイナリー互換性を判断基準とする

Java 言語仕様にバイナリー互換性の定義^a

^a<http://docs.oracle.com/javase/specs/jls/se7/html/jls-13.html>

API に変更する前後に、リンクする際にエラーが起こらないことはバイナリー互換性があること

本研究に使った定義

再度コンパイルし直す必要がない API の変更^a

^a<http://wiki.eclipse.org/EvolvingJava-basedAPIs>

API の後方互換性を判断する基準

Determining backward incompatible API changes

厳密に API 互換性の有無を判断することは困難

- 意味的に解析や、実際にビルド・テストする必要
代わりに本研究では**バイナリー互換性**を判断基準とする

Java 言語仕様にバイナリー互換性の定義^a

^a<http://docs.oracle.com/javase/specs/jls/se7/html/jls-13.html>

API に変更する前後に、リンクする際にエラーが起こらないことはバイナリー互換性があること

本研究に使った定義

再度コンパイルし直す必要がない API の変更^a

^a<http://wiki.eclipse.org/EvolvingJava-basedAPIs>

Clirr: 公開 API に変更を検出ツール⁷

Clirr: Tool to Extract API changes

入力: バイナリー Jar ファイル、出力: 公開 API 違い

互換性がない	互換性がある
メソッドの削除	メソッドの追加
クラスの削除	クラスの追加
フィールドの削除	フィールドの追加
引数の型が変更	親クラスレスとに追加
フィールドの型が変更	定数の値が変更
総数：23	総数：20

安全側に立って、検出した互換性がない修正は絶対正しいが、互換性がある変更に間違いがある

⁷<http://clirr.sourceforge.net>

バージョン番号の比較

Determining subsequent versions and update types

以下の要素でライブラリーのバージョンを特定

groupId	artifactId	version
org.springframework	spring-core	2.5.6

バージョン番号の比較は Artifact API⁸ を使います

フォーマットは MAJOR.MINOR.PATCH、PATCH がない場合に 0 とみなす

プレリリース (1.2.3-beta1 や 1.2.3-snapshot) は対象から外す

⁸<http://maven.apache.org/ref/3.1.1/maven-artifact>

バージョン番号の比較

Determining subsequent versions and update types

以下の要素でライブラリーのバージョンを特定

groupId	artifactId	version
org.springframework	spring-core	2.5.6

バージョン番号の比較は Artifact API⁸ を使います

フォーマットは MAJOR.MINOR.PATCH、PATCH がない場合に 0 とみなす

プレリリース (1.2.3-beta1 や 1.2.3-snapshot) は対象から外す

⁸<http://maven.apache.org/ref/3.1.1/maven-artifact>

バージョン番号の比較

Determining subsequent versions and update types

以下の要素でライブラリーのバージョンを特定

groupId	artifactId	version
org.springframework	spring-core	2.5.6

バージョン番号の比較は Artifact API⁸ を使います

フォーマットは MAJOR.MINOR.PATCH、PATCH がない場合に 0 とみなす

プレリリース (1.2.3-beta1 や 1.2.3-snapshot) は対象から外す

⁸<http://maven.apache.org/ref/3.1.1/maven-artifact>

バージョン番号の比較

Determining subsequent versions and update types

以下の要素でライブラリーのバージョンを特定

groupId	artifactId	version
org.springframework	spring-core	2.5.6

バージョン番号の比較は Artifact API⁸ を使います

フォーマットは MAJOR.MINOR.PATCH、PATCH がない場合に 0 とみなす

プレリリース (1.2.3-beta1 や 1.2.3-snapshot) は対象から外す

⁸<http://maven.apache.org/ref/3.1.1/maven-artifact>

ソースコードの比較と廃止パターン

ソースコードはどのくらい変更したのかを
ChangeDistiller⁹で比較します

入力：2つバージョンのソースコード

IEEE Transactions on 33.11 (2007), pp. 725–743.

ソースコードの比較と廃止パターン

ソースコードはどのくらい変更したのかを
ChangeDistiller⁹で比較します

入力：2つバージョンのソースコード

廃止 API はキーワード `@Deprecated` で検索し、該当するソースファイルを JDT で解析して、廃止 API のパターンを検出します。

IEEE Transactions on 33.11 (2007), pp. 725–743.

1 背景

2 Research Questions

3 調査手法

4 統計的な結果

- バージョン文字列のパターン
- 互換性が有無の API 変更

5 調査項目の結果

6 議論と妥当性への脅威

7 結論

バージョン文字列のパターン

#	Pattern	Example	#Single	#Pairs	Incl.
1	MAJOR.MINOR	2.0	20,680	11,559	yes
2	MAJOR.MINOR.PATCH	2.0.1	65,515	50,020	yes
3	#1 or #2 with nonnum. chars	2.0.D1	3,269	2,150	yes
4	MAJOR.MINOR-prerelease	2.0-beta1	16,115	10,756	no
5	MAJOR.MINOR.PATCH-pre.	2.0.1-beta1	12,674	8,939	no
6	Other versioning scheme	2.0.1.5.4	10,930	8,307	no
		Total	129,138	91,731	

69% のバージョン文字列は SemVer のフォーマットになっています。

22.3% のバージョン文字列はプレリリース（4 と 5）である。

29% のライブラリーは 1 つのリリースのみである。

互換性が有無の API 変更

Breaking changes			Non-breaking changes		
#	Change type	Frequency	#	Change type	Frequency
1	Method has been removed	177,480	1	Method has been added	518,690
2	Class has been removed	168,743	2	Class has been added	216,117
3	Field has been removed	126,334	3	Field has been added	206,851
4	Parameter type change	69,335	4	Interface has been added	32,569
5	Method return type change	54,742	5	Method removed, inherited still exists	25,170
6	Interface has been removed	46,852	6	Field accessibility increased	24,954
7	Number of arguments changed	42,286	7	Value of compile-time constant changed	16,768
8	Method added to interface	28,833	8	Method accessibility increased	14,630
9	Field type change	27,306	9	Addition to list of superclasses	13,497
10	Field removed, previously constant	12,979	10	Method no longer final	9,202

互換性が有無の API 変更

Breaking changes			Non-breaking changes		
#	Change type	Frequency	#	Change type	Frequency
1	Method has been removed	177,480	1	Method has been added	518,690
2	Class has been removed	168,743	2	Class has been added	216,117
3	Field has been removed	126,334	3	Field has been added	206,851
4	Parameter type change	69,335	4	Interface has been added	32,569
5	Method return type change	54,742	5	Method removed, inherited still exists	25,170
6	Interface has been removed	46,852	6	Field accessibility increased	24,954
7	Number of arguments changed	42,286	7	Value of compile-time constant changed	16,768
8	Method added to interface	28,833	8	Method accessibility increased	14,630
9	Field type change	27,306	9	Addition to list of superclasses	13,497
10	Field removed, previously constant	12,979	10	Method no longer final	9,202

	Contains at least 1 breaking change				
Update type	Yes	%	No	%	Total
Major	4,268	35.8%	7,624	64.2%	11,892
Minor	10,690	35.7%	19,267	64.3%	29,957
Patch	9,239	23.8%	29,501	76.2%	38,740
Total	24,197	30.0%	56,392	70.0%	80,589

Major と Minor の使い分けは **されていない** !

互換性が有無の API 変更

Breaking changes			Non-breaking changes		
#	Change type	Frequency	#	Change type	Frequency
1	Method has been removed	177,480	1	Method has been added	518,690
2	Class has been removed	168,743	2	Class has been added	216,117
3	Field has been removed	126,334	3	Field has been added	206,851
4	Parameter type change	69,335	4	Interface has been added	32,569
5	Method return type change	54,742	5	Method removed, inherited still exists	25,170
6	Interface has been removed	46,852	6	Field accessibility increased	24,954
7	Number of arguments changed	42,286	7	Value of compile-time constant changed	16,768
8	Method added to interface	28,833	8	Method accessibility increased	14,630
9	Field type change	27,306	9	Addition to list of superclasses	13,497
10	Field removed, previously constant	12,979	10	Method no longer final	9,202

	Contains at least 1 breaking change				
Update type	Yes	%	No	%	Total
Major	4,268	35.8%	7,624	64.2%	11,892
Minor	10,690	35.7%	19,267	64.3%	29,957
Patch	9,239	23.8%	29,501	76.2%	38,740
Total	24,197	30.0%	56,392	70.0%	80,589

Major と Minor の使い分けは **されていない** !

1 背景

2 Research Questions

3 調査手法

4 統計的な結果

5 調査項目の結果

- RQ1: SemVer 原則が従われているか
- RQ2: 時間を亘って変わるか
- RQ3: 依存関係がどう更新されるか
- RQ4: 廃止予定タグが使われているか

6 議論と妥当性への脅威

7 結論

RQ1: SemVer 原則が従われているか

Type	#Breaking		#Non-break.		Edit script		Days	
	μ	σ^2	μ	σ^2	μ	σ^2	μ	σ^2
Major	58.3	337.3	90.7	582.1	50.0	173.0	59.8	169.8
Minor	27.4	284.7	52.2	255.5	52.7	190.5	76.5	138.3
Patch	30.1	204.6	42.8	217.8	22.7	106.5	62.8	94.4
Total	32.0	264.3	52.2	293.3	37.2	152.3	67.4	122.9

SemVer を従われていれば、Minor/Patch に互換性がない変更はないはずなのに、実際 Minor/Patch に平均 30 ぐらいの互換性がない変更がある。

編集の大きさから見ると、Minor は Major より大きい
が、Patch は相対的に小さい。

リリースの間隔から見ると、Minor は Major より時間
がかかる。

RQ1:SemVer 原則が従われているか

Type	#Breaking		#Non-break.		Edit script		Days	
	μ	σ^2	μ	σ^2	μ	σ^2	μ	σ^2
Major	58.3	337.3	90.7	582.1	50.0	173.0	59.8	169.8
Minor	27.4	284.7	52.2	255.5	52.7	190.5	76.5	138.3
Patch	30.1	204.6	42.8	217.8	22.7	106.5	62.8	94.4
Total	32.0	264.3	52.2	293.3	37.2	152.3	67.4	122.9

SemVer を従われていれば、Minor/Patch に互換性がない変更はないはずなのに、実際 Minor/Patch に平均 30 ぐらいの互換性がない変更がある。

編集の大きさから見ると、Minor は Major より大きい
が、Patch は相対的に小さい。

リリースの間隔から見ると、Minor は Major より時間
がかかる。

RQ1:SemVer 原則が従われているか

Type	#Breaking		#Non-break.		Edit script		Days	
	μ	σ^2	μ	σ^2	μ	σ^2	μ	σ^2
Major	58.3	337.3	90.7	582.1	50.0	173.0	59.8	169.8
Minor	27.4	284.7	52.2	255.5	52.7	190.5	76.5	138.3
Patch	30.1	204.6	42.8	217.8	22.7	106.5	62.8	94.4
Total	32.0	264.3	52.2	293.3	37.2	152.3	67.4	122.9

SemVer を従われていれば、Minor/Patch に互換性がない変更はないはずなのに、実際 Minor/Patch に平均 30 ぐらいの互換性がない変更がある。

編集の大きさから見ると、Minor は Major より大きい
が、Patch は相対的に小さい。

リリースの間隔から見ると、Minor は Major より時間
がかかる。

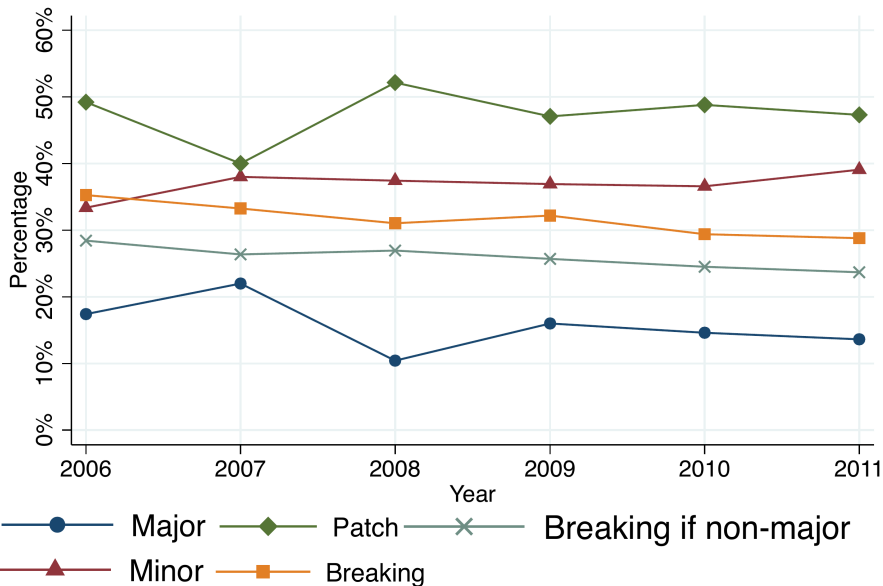
後方互換性に関する
SemVer 原則は実際
に**守られていない**！

Minor リリースと Patch リリースには、互換性のない変更が**平均 30 個**ぐらいあります。

後方互換性に関する
SemVer 原則は実際
に**守られていない**！

Minor リリースと Patch リリースに
は、互換性のない変更が**平均 30
個**ぐらいあります。

RQ2: 時間を亘って変わるか



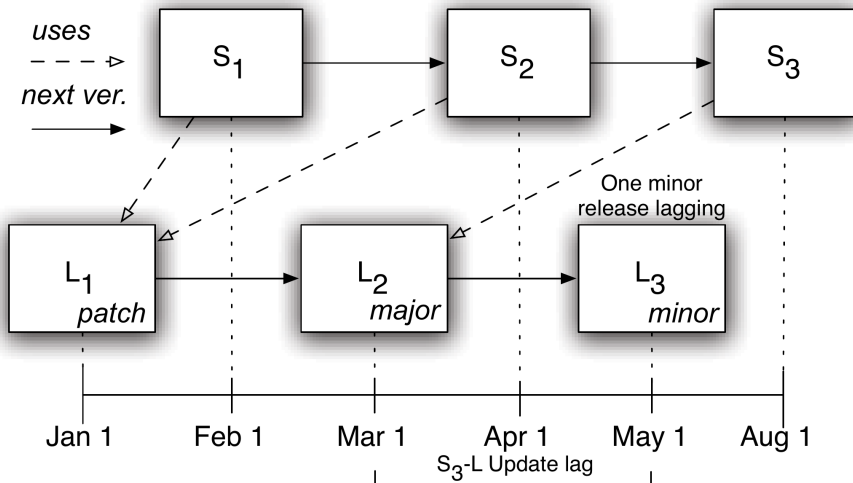
SemVer 原則を守る
ことは近年少しくら
い増えている

Minor と Patch リリースに互換性の
ない変更の割合は、2006 年
の28.4%から、2011 年に23.7%になり
ました。

SemVer 原則を守る
ことは近年少しくら
い増えている

Minor と Patch リリースに互換性の
ない変更の割合は、2006 年
の28.4%から、2011 年に23.7%になり
ました。

RQ3: 依存関係がどう更新されるか



RQ3-1: 依存関係が伴う更新される数

互換性がない更新に着目する理由は、依存関係の更新を困難に持たすから、ここで依存関係の更新に関しても調べました。
ソフトウェア S の更新される際に、依存関係の記述にライブラリー L を伴って更新された数を調べた。

	<i>Update L</i>			
<i>Update S</i>	<i>Major</i>	<i>Minor</i>	<i>Patch</i>	<i>Total</i>
<i>Major</i>	543	189	82	814
<i>Minor</i>	651	791	227	1,669
<i>Patch</i>	150	54	297	501
<i>Total</i>	1,344	1,034	606	2,984

RQ3-2: 依存関係が伴う更新の遅延

更新の遅延: S が L を依存して、 L の最新版 L_n がリリースされたが、その後にリリースされた S の最新版は古い L_o に依存したまま場合、一回の遅延と見なし、遅延の長さは L_o と L_n の間のリリースの数として定義。

	<i>min</i>	<i>p25</i>	<i>p50</i>	<i>p75</i>	<i>p90</i>	<i>p95</i>	<i>p99</i>	<i>max</i>
Major	0	0	0	0	1	1	4	22
Minor	0	0	0	1	2	4	6	101
Patch	0	0	0	1	5	6	13	46

更新の遅延は互換性のない変更の数とソースコードに対する変更の激しさの関係を、スピアマン相関係数で評価

	Breaking changes	Edit script size
Major versions lagging	0.0772	-0.0701
Minor versions lagging	0.1440	0.1272
Patch versions lagging	0.0190	0.0199

RQ3 に対する回答

依存関係の更新は多くの場合に、ソフトウェアの Major リリースにライブラリーの Major リリースを更新される

依存関係が更新するのに遅延が存在され、Patch リリースによく発生する

更新の遅延は変化の激しさとの相関は弱いけど存在する

RQ3 に対する回答

依存関係の更新は多くの場合に、ソフトウェアの Major リリースにライブラリーの Major リリースを更新される

依存関係が更新するのに遅延が存在され、Patch リリースによく発生する

更新の遅延は変化の激しさとの相関は弱いけど存在する

RQ3 に対する回答

依存関係の更新は多くの場合に、ソフトウェアの Major リリースにライブラリーの Major リリースを更新される

依存関係が更新するのに遅延が存在され、Patch リリースによく発生する

更新の遅延は変化の激しさとの相関は弱いけど存在する

廃止予定タグの正しい使い方

バージョン: 1.0

```
1 class foo{
2
3     public void
4     function(){
5         //...
6     };
7
8
9
10
11 }
```

バージョン: 1.1

```
1 class foo{
2 +   @Deprecated
3     public void
4     function(){
5         //...
6     };
7 +   public Object
8 +   functionEx(){
9 +       //...
10  +   };
11 }
```

バージョン: 2.0

```
1 class foo{
2 -   @Deprecated
3 -   public void
4 -   function(){
5 -       //...
6 -   };
7   public Object
8   functionEx(){
9       //...
10  };
11 }
```

RQ4: 廃止予定タグが使われているか

22,205 プロジェクトの内、1196 (5.4%) は少なくとも一回メソッドに廃止予定のタグがつけられていた

#	v1 (maj.)	v2 (min.)	v3 (min.)	v4 (maj.)	c	i	Freq.	%
1	pr m1	pr m1	pr m1	pr m1	y	n	63,698	24.34
2	pr m2	pr m2	pr @d m2	pr @d m2	y	n	113	0.04
3	pu m3	pu m3	pu m3	pu m3	y	n	110,613	42.27
4	pu m4	pu @d m4	pu @d m4	pu @d m4	y	y	793	0.30
5	pu m5	pu m5	-	-	n	y	86,449	33.03
6	pu m6	pu @d m6	-	-	n	y	0	0
7	pu m7	pu m7	pu m7	pu @d m7	n	y	0	0
8	pu m8	pu @d m8	pu @d m8	-	y	y	0	0
9	pu m9	pu @d m9	pu m9	pu m9	n	y	16	0.01

1,2,3 private や廃止と関係ないメソッドは対象外

5,6,7 間違い使い方

4,8 正しい使い方

9 廃止予定がキャンセルした

RQ4: 廃止予定タグが使われているか

22,205 プロジェクトの内、1196 (5.4%) は少なくとも一回メソッドに廃止予定のタグがつけられていた

#	v1 (maj.)	v2 (min.)	v3 (min.)	v4 (maj.)	c	i	Freq.	%
1	pr m1	pr m1	pr m1	pr m1	y	n	63,698	24.34
2	pr m2	pr m2	pr @d m2	pr @d m2	y	n	113	0.04
3	pu m3	pu m3	pu m3	pu m3	y	n	110,613	42.27
4	pu m4	pu @d m4	pu @d m4	pu @d m4	y	y	793	0.30
5	pu m5	pu m5	-	-	n	y	86,449	33.03
6	pu m6	pu @d m6	-	-	n	y	0	0
7	pu m7	pu m7	pu m7	pu @d m7	n	y	0	0
8	pu m8	pu @d m8	pu @d m8	-	y	y	0	0
9	pu m9	pu @d m9	pu m9	pu m9	n	y	16	0.01

1,2,3 private や廃止と関係ないメソッドは対象外

5,6,7 間違い使い方

4,8 正しい使い方

9 廃止予定がキャンセルした

RQ4: 廃止予定タグが使われているか

22,205 プロジェクトの内、1196 (5.4%) は少なくとも一回メソッドに廃止予定のタグがつけられていた

#	v1 (maj.)	v2 (min.)	v3 (min.)	v4 (maj.)	c	i	Freq.	%
1	pr m1	pr m1	pr m1	pr m1	y	n	63,698	24.34
2	pr m2	pr m2	pr @d m2	pr @d m2	y	n	113	0.04
3	pu m3	pu m3	pu m3	pu m3	y	n	110,613	42.27
4	pu m4	pu @d m4	pu @d m4	pu @d m4	y	y	793	0.30
5	pu m5	pu m5	-	-	n	y	86,449	33.03
6	pu m6	pu @d m6	-	-	n	y	0	0
7	pu m7	pu m7	pu m7	pu @d m7	n	y	0	0
8	pu m8	pu @d m8	pu @d m8	-	y	y	0	0
9	pu m9	pu @d m9	pu m9	pu m9	n	y	16	0.01

1,2,3 private や廃止と関係ないメソッドは対象外

5,6,7 間違い使い方

4,8 正しい使い方

9 廃止予定がキャンセルした

RQ4: 廃止予定タグが使われているか

22,205 プロジェクトの内、1196 (5.4%) は少なくとも一回メソッドに廃止予定のタグがつけられていた

#	v1 (maj.)	v2 (min.)	v3 (min.)	v4 (maj.)	c	i	Freq.	%
1	pr m1	pr m1	pr m1	pr m1	y	n	63,698	24.34
2	pr m2	pr m2	pr @d m2	pr @d m2	y	n	113	0.04
3	pu m3	pu m3	pu m3	pu m3	y	n	110,613	42.27
4	pu m4	pu @d m4	pu @d m4	pu @d m4	y	y	793	0.30
5	pu m5	pu m5	-	-	n	y	86,449	33.03
6	pu m6	pu @d m6	-	-	n	y	0	0
7	pu m7	pu m7	pu m7	pu @d m7	n	y	0	0
8	pu m8	pu @d m8	pu @d m8	-	y	y	0	0
9	pu m9	pu @d m9	pu m9	pu m9	n	y	16	0.01

1,2,3 private や廃止と関係ないメソッドは対象外

5,6,7 間違い使い方

4,8 正しい使い方

9 廃止予定がキャンセルした

RQ4: 廃止予定タグが使われているか

22,205 プロジェクトの内、1196 (5.4%) は少なくとも一回メソッドに廃止予定のタグがつけられていた

#	v1 (maj.)	v2 (min.)	v3 (min.)	v4 (maj.)	c	i	Freq.	%
1	pr m1	pr m1	pr m1	pr m1	y	n	63,698	24.34
2	pr m2	pr m2	pr @d m2	pr @d m2	y	n	113	0.04
3	pu m3	pu m3	pu m3	pu m3	y	n	110,613	42.27
4	pu m4	pu @d m4	pu @d m4	pu @d m4	y	y	793	0.30
5	pu m5	pu m5	-	-	n	y	86,449	33.03
6	pu m6	pu @d m6	-	-	n	y	0	0
7	pu m7	pu m7	pu m7	pu @d m7	n	y	0	0
8	pu m8	pu @d m8	pu @d m8	-	y	y	0	0
9	pu m9	pu @d m9	pu m9	pu m9	n	y	16	0.01

1,2,3 private や廃止と関係ないメソッドは対象外

5,6,7 間違い使い方

4,8 正しい使い方

9 廃止予定がキャンセルした

RQ4 に対する回答

プログラマーは廃止予定のタグを殆ど使われていない。

使われていても、
正しく使われていない。

RQ4 に対する回答

プログラマーは廃止予定のタグを殆ど使われていない。

使われていても、
正しく使われていない。

1 背景

2 Research Questions

3 調査手法

4 統計的な結果

5 調査項目の結果

6 議論と妥当性への脅威

- SemVer 原則を守れない原因
- リリースの間隔と編集の規模
- 初期開発段階のリリース
- 妥当性への脅威

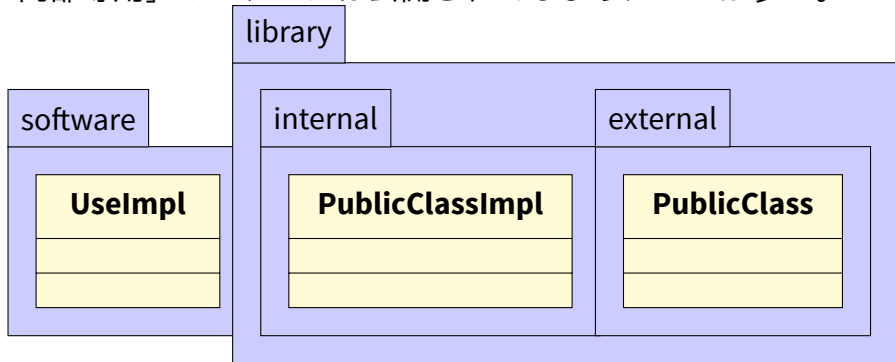
7 結論

SemVer 原則を守れない原因

Low adherence explained

Java のモジュールシステムに可視性の設定はプログラマーの必要に満たせない。

「内部専用」のパッケージは公開されてしまったことが多い。



実際の使い方と潜在的な使い方の違いもあります

リリースの間隔と編集の規模

直感で考えると、Major リリースは Minor リリースより時間がかかるはずなのに、実際のデータから見る限り、**Minor リリースのほうが時間がかかります。**

推測: Major リリースは意図的互換性を破るために作ることが多く、機能追加はその後にある Minor リリースに行う。

ソースコードの編集の規模からもこの傾向が見えます: Minor リリースのほうが編集が多くされます。

リリースの間隔と編集の規模

直感で考えると、Major リリースは Minor リリースより時間がかかるはずなのに、実際のデータから見る限り、**Minor リリースのほうが時間がかかります。**

推測: Major リリースは意図的互換性を破るために作ることが多く、機能追加はその後にある Minor リリースに行う。

ソースコードの編集の規模からもこの傾向が見えます: Minor リリースのほうが編集が多くされます。

リリースの間隔と編集の規模

直感で考えると、Major リリースは Minor リリースより時間がかかるはずなのに、実際のデータから見る限り、**Minor リリースのほうが時間がかかります。**

推測: Major リリースは意図的互換性を破るために作ることが多く、機能追加はその後にある Minor リリースに行う。

ソースコードの編集の規模からもこの傾向が見えます: Minor リリースのほうが編集が多くされます。

初期開発段階のリリース

SemVer の説明によると「Major 番号は 0 の場合 (0.y.z)、互換性を守らないことは許されます」

今回の調査にこのルールを考慮していない。数だけを調べたら、Major 番号は 0 のリリース数は 10.44% (13,162 / 126,070) があります。

初期開発段階のリリース

SemVer の説明によると「Major 番号は 0 の場合 (0.y.z)、互換性を守らないことは許されます」

今回の調査にこのルールを考慮していない。数だけを調べたら、Major 番号は 0 のリリース数は 10.44% (13,162 / 126,070) があります。

妥当性への脅威

内部的妥当性

記録されたライブラリーのリリース日時が間違えた可能性がある。

- 2,321, 1.5% のバイナリーファイルのリリース日時は 2005 年 11 月 5 日になって、データから省いた
- リリース日時の順位とバージョン番号によるソートした順位と一致しないことはある。

外部的妥当性

Maven にある Java で書かれた OSS だけ対象にした

実験結果の再現性

スーパーコンピュータで、100 計算ノード、合計六ヶ月の計算時間を使って分析した

1 背景

2 Research Questions

3 調査手法

4 統計的な結果

5 調査項目の結果

6 議論と妥当性への脅威

7 結論

結論

この研究は 22,000 超える Maven にある OSS ライブラリーを対象として、バージョン番号と互換性の関係について、SemVer を基準として調査した。発見したことは：

- 互換性がない変更がよくある： $\frac{1}{3}$ のリリース
- major バージョンであるかどうかと関わらず、互換性がない変更は同じぐらいに存在する
- 互換性がない変更の存在は新しいライブラリーを使う遅延に影響は小さい
- 廃止予定のタグはあまり使われていない、使った場合でも正しく使われていない