

Business Readiness Rating for Open Source

オープンソースソフトウェアの評価と採用を 促進するためのオープンな基準

サマリー

ソフトウェアを評価することは、企業の IT マネージャーの重要な仕事です。しかし、オープンソースソフトウェアの潜在的ユーザーは、意思決定のための簡単で、効果的で、信頼できる方法を十分持ちあわせていません。また、評価のために広く使われているモデルがありません。オープンソースソフトウェアを評価している会社は単に経験からのみ学んでいるため、オープンソースの採用をむずかしくしています。

SpikeSource 社、カーネギーメロン大学 Open Source Investigation センターとインテル社は、Business Readiness Rating モデルを開発しました。そして、このモデルは IT マネージャーが、迅速にオープンソースソフトウェアについて情報武装された決定を下すことを可能とします。このモデルはまた、ユーザーが自分達の評価をオープンソースコミュニティにフィードバックすることができます。

Business Readiness Rating モデルで使用される計算式は、ある特定の使用状況において、オープンソースソフトウェアの成功する構築にとって最も重要であると考えられる要因に重きを置いています。それらは、機能性、品質、パフォーマンス、コミュニティサイズ、セキュリティ等です。Business Readiness Rating モデルは、オープンであり、柔軟性がありますが、まだ標準化されていません。このモデルは、オープンソースソフトウェアとプロプライエタリーのソフトウェアの組織的かつ透明性のある評価の実現を可能とします。

目次

サマリー	1
1 - 困難なソフトウェアの選択	3
2 - 今日のソフトウェア評価のプラクティス	5
3 - ソフトウェア評価のためのオープンで標準的なモデル	6
過去のオープンソースソフトウェアの評価モデル	7
4 -Business Readiness Rating Model の紹介	8
初期評価（フィルター）	8
メトリックスとカテゴリー	9
機能重視に合わせる	10
モデルの活用	12
5 - 結論	13
Appendix 1: Business Readiness Rating の詳細	14
I. 迅速な評価のためのベストプラクティス	14
II. ターゲットとなる使用方法評価のためのベストプラクティス	16
III. データの収集と処理のためのベストプラクティス	18
メトリックス測定の標準化	18
機能メトリックスの処理	20
加重要因のカテゴリーレート	20
IV. 典型的なメトリックスとスコア	21
Appendix 2: 他の情報	27
I. 成熟したオープンソフトウェアの特徴	27
II. 参照	28

1. 困難なソフトウェア選択

組織内でどのソフトウェアパッケージを採用すべきかを定めることは、困難な仕事といえます。ソフトウェア活用から見込まれる利点は、リスク（例えば互換性の問題、ユーザビリティ、スケーラビリティと合法性等）と表裏一体です。

伝統的に、企業は、以下の制限にもかかわらず、商用またはプロプライエトリーのソフトウェアに依存してきました：

- ・コスト・・・通常、高い。
- ・未公開ソースコード・・・ソフトウェアのセキュリティと品質が未知。
- ・製品ロードマップに対する影響力の欠如・・・ソフトウェアベンダーは、どこを改善すべきかについて選ぶことができます。つまり、顧客のリクエストがソフトウェアベンダーのロードマップに適合しないならば、顧客のリクエストは無視される可能性があります。

これらの要因は、商用（プロプライエトリー）のソフトウェアを使用する上での主な利点を相殺します：

- ・サポート・・・商用ソフトウェアベンダーは、顧客が製品に関する問題を解決することを助けるための専任のサポートスタッフを抱えています。

今日、以下の理由で、オープンソースソフトウェアをビジネスで活用することを検討することがますます増えています：

- ・コスト・・・しばしば無料です。
- ・ソースコードへのアクセス・・・ソースコードが公開されているので、自動コードレビューに役立ちます。従って、成熟したオープンソースソフトウェアプロジェクトは、商用ソフトウェアより安全で、バグも少なくなる傾向にあります。
- ・オープンアーキテクチャー・・・オープンソースソフトウェアは、仮想コミュニティを通して開発されます。開発コミュニティはしばしば地理的に制限されないため、オープンソースプロジェクトは通常、モジュール式になっています。モジュール式のコードは伸長可能で、デバッグが容易です。
- ・品質・・・上述のソースとアーキテクチャーの透明度は、よく管理されたプロジェクトが、成熟かつ高品質の製品を開発する事を可能にします。

これらの利点にもかかわらず、いくつかの問題は、オープンソースソフトウェア採用を妨げます。オープンソースプロジェクトの数は、沢山あり、またプロジェクトは、個人ベースの低品質なプロジェクトから、高品質の企業ソリューションレベルのプロジェクトまであります。SourceForge だけで、100,000 以上のオープンソースプロジェクトがあり、また CodeHaus、Tigris、Java.net、ObjectWeb、OpenSymphony のような他のオープンソースの宝庫にはまだたくさんのプロジェクトがあります。

オープンソースソフトウェアのユーザーや潜在的採用者は、以下の困難に直面します：

- ・選択・・・ソフトウェアのカテゴリによっては、選択は実質的に無限です。
- ・サポート・・・多くのオープンソースパッケージは、専門的にサポートされません。
- ・寿命・・・大部分のオープンソースプロジェクトが商業的な会社によって支えられていないので、将来のリリースはコミュニティの努力に依存します。
- ・不安定性・・・多くのオープンソースプロジェクトは、コミュニティの中での牽引や注目を得るため、「**Release Early, Release Often**」の パラダイムに陥りがちです。従い、オープンソース界の唯一の恒常的なものは、変化です。オープンソースソフトウェアの多くの潜在的採用者は、オープンソース界で急速な最新版と一般的なソフトウェアパッケージの変更を追って、実行する準備ができていません。
- ・未熟なプロジェクトのための低品質なコード・・・コンセプトフェーズの間、オープンソースプロジェクトは、数人の道楽者（趣味とする人達）または、何かを創造する事に情熱をもったお金に困った IT スタッフによって開発されます。これらの初期の開発者は、完全な製品を届けるために必要な資源または経験が不足しているかもしれません。最初の開発者が彼らの問題を解決するためのオープンソースコンポーネントを作るならば、プロジェクトは急激に孤児になるかもしれません。新しいプロジェクトは、正式なソフトウェア工学またはテストなしで開発されるかもしれません。

その結果、オープンソースソフトウェアの品質において、非常に広い継続があります。広く採用されたオープンソースプロジェクトは、しばしば高品質のソフト製品に進化します・・・しばしば、彼らのライバルである商用製品よりも。しかし、未熟なオープンソースソフト製品は、採用者に対して利益よりも多くのリスクをもたらすかもしれません。オープンソースソフトウェアの成熟度を評価するための広く採用され、標準化された方法が本当に必要です。（本論文の Appendix2 に、我々は成熟したオープンソースソフトウェアの特徴の多くをリストしています。）

「ユーザーがあなたにそれをサポートするよう頼むまで、それは楽しみです。」・・・ビル・ジョイ氏

「オープンソースは、しばしば商用（プロプライエタリー）のソフトウェアを打ち破ることができます。しかし、これはいつも真実ではありません。」・・・ジャン・ミシェルダール氏、ニコラス・ジュリアン氏

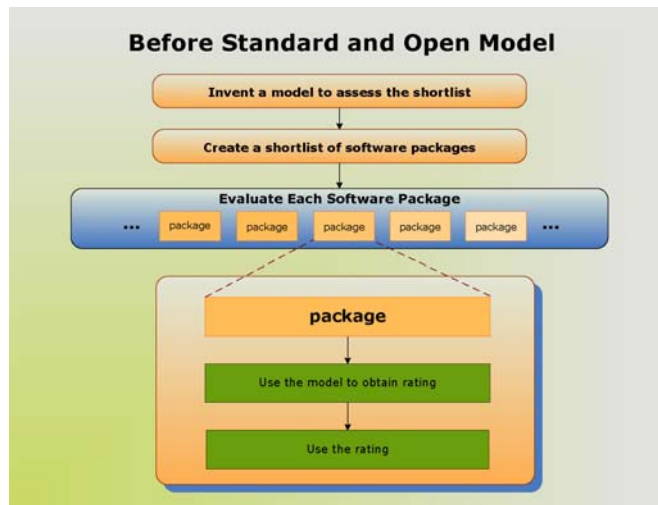
「全ての **Open Source** イニシアティブにとって、最初の年は、最も大きなハードルです。」・・・CapGemini 社

2. 今日のソフトウェア評価のプラクティス

どのソフトウェアを活用すべきか決めるためには、多くの選択肢をリストアップすることが必要です。それから、重要なクライテリアを満たすショートリストにそれらの選択肢を選別するための迅速な評価が必要です。ショートリストを作成することは、あり余る程のオープンソースソフトウェア選択肢と将来のバージョンについての不確実性のために、脅威になるかもしれません。それでも、ショートリストなしでは、不適当なプロジェクトを評価する上で、多くの無駄な時間とエネルギーが使われます。

良いショートリストはできるだけ多くの現実的実行可能なソフトウェアの選択肢を含まなければならない、一方で適当でないものを除外しなければなりません。あるアプリケーションカテゴリーにおいては、ソフトウェアの特性または標準は、役に立つフィルター(例えばそれがサポートするアプリケーションの主要な言語またはデータベース)であるかもしれません。しかし、多くのアプリケーションカテゴリーに

においては、そのようなフィルターは、手に入れるのが難しいです。「プログラミング言語」のような単純なフィルターを、<http://www.cmsmatrix.org> にリストされている 375 の考えられる Content Management パッケージに適用した後でさえ、リストはまだ多くを含んでおり、ロングリストです。



ショートリストを作成した後、IT スタッフは、リストに載っているソフトウェアパッケージをより完全に評価しなければなりません。大部分の評価者は、自分自身の評価方法を発明しなければなりません。そして、評価データへのアクセスまたは彼らの仲間の方法なしで、他が同じソフトウェアパッケージを評価したときでも、彼らは各々のパッケージを再評価しなければなりません。

実際には、正式な評価方法論なしで、多くのソフトウェア評価プロジェクトは、「臨機応変」で、実行されます。「臨機応変」の方法は、彼らの評価において、誤っているか、または不完全であるかもしれません。そして、評価の正確さを確認することはとても難しいです。不正確で不完全な評価メカニズムは、誤った決定と製品選択に至ります。そして、それは「臨機応変」を危険にします。

3. ソフトウェア評価のためのオープンで標準化されたモデル

どのように、ビジネスユーザーや、開発者や IT エンジニアは、より簡単に、どのオープンソースソフトウェアを使用すべきかについて、決めることができるのでしょうか？どのように、彼らの目的のために、彼らが考慮しているソフトウェアが成熟していて、「準備ができているビジネスか」どうか、自信をもって決めることができるのでしょうか？

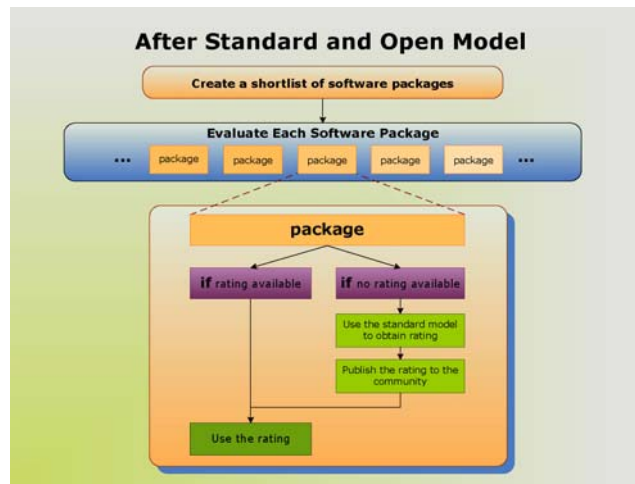
我々は、ソフトウェアを評価するためのオープンで標準化されたモデルを使うことが、評価の簡便さと正確さを強化し、オープンソースソフトウェアの採択を速めることであると提唱します。

その上、広く採用されて議論の必要がない、オープンで標準的な評価モデルによって、オープンソースソフトウェアユーザーが、評価結果を共有することができます。なぜ「標準化」なのでしょう？

標準化されたモデルは、評価レーティングの共通な理解を可能にします。なぜ「オープン」なのでしょう？オープンモデルは、評価プロセスにおける「信頼」を推進します。また、将来の変化に関して柔軟なことを確実にします。正確さのためのオープンな評価モデルの有効性は、ストレートです：モデルの潜在的採用者は、それを見て、それについてコメントして、それを改善します。

そのようなモデルは、良いソフトウェア・レーティング・モデルの重要な必要条件を含みます・・・そして、それは **Complete** (完全性)、**Simple** (単純さ)、**Adaptable** (適応性) と **Consistent** (一貫性) (CSAC) です：

- **Complete**・・・製品レーティングモデルの第一の必要条件は、製品のすべての突出した特性をハイライトするモデルの能力です。製品のレーティングが決してまぎらわしくないように、これは必要です。
- **Simple**・・・広く受け入れられるために、モデルは簡単に理解されなければならず、比較的使い易くなければなりません。さらにまた、評価と用語は、カスタマーフレンドリーでなければなりません。しかし、モデルの完全性は、より高い優先順位となります。
- **Adaptable**・・・ソフトウェア産業の急速な変化のために、今日作られたソフトウェアレーティングモデルは、将来とは無関係かもしれません。コンセプトステージの間、



モデルの全ての将来の潜在的な使用法を捕えることは、不可能です。したがって、我々は、適応性・・・そしてオープン性・・・で我々のモデルを造っていくよう努めます。モデルが拡張を必要とするとき、現在のモデルが混乱なしで、ひとつを加えることは簡単です。

- **Consistent**・・・モデルが生み出すスケールとレーティングは、モデルの異なるターゲット使用法を通して一貫性がなければいけません。2つのカテゴリからの2つのソフトウェアパッケージのための比較評価は、“equal business readiness”を意味しなければなりません。

*過去のオープンソースソフトウェアの評価モデル

異なるアプローチが、ソフトウェアを評価するために存在します。少なくとも2つの団体は、オープンソースソフトウェアの適合性を評価するために、オープンソース採用者に方法論を提供しようとしています：Navicasoft社のバーナード・ゴールドデン氏によって開発された **Open Source Maturity Model**・・・”Succeeding Open Source”（2004年発刊）と、そして、CapGemini社による **Open Source Maturity Model** (seriouslyopen.org から入手可能)。これらのモデルは、優れた開拓者的な努力の賜物です。

新しいモデルを提唱するにあたり、我々は十分なソフトウェア評価のために、類似したコンセプトを使います。しかしながら、我々はソフトウェアの **Readiness** を評価するため、特に運営とサポートの観点で、より詳細な評価データとスコアを提供します。評価のための特定の領域の分離と、特定の使用に基づく加重評価を提供する事は、バーナード・ゴールドデン氏と CapGemini のモデルの共通のコンセプトです。我々はそのコンセプトを拡張し、評価データから点数付けまでを明確にマッピングする評価システムのための科学的なモデルを提供します。我々のモデルの紹介において、我々はできるだけ多くの評価状況において広く採用可能で簡単に使えるソフトウェア評価についての方法論のアイデアを拡張することを期待します。

4. Business Readiness Rating Model 紹介

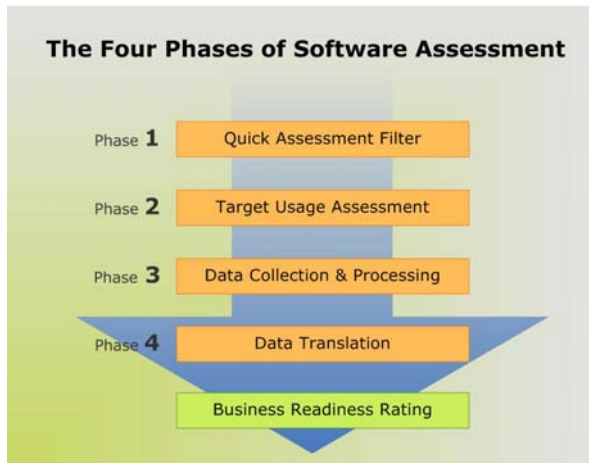
SpikeSource 社、カーネギーメロン大学 Open Source Investigation センターとインテル社は、オープンソースソフトウェアの評価を単純化することを望みます。そのために、我々は標準化されオープンなモデル・・・Business Readiness Rating (BRR)・・・を提唱致します。

この Business Readiness Rating モデルは、IT マネージャーがどのオープンソースソフトウェアが彼らのニーズに最もふさわしいかについて判断することを援助することを目的とします。オープンソースユーザーはまた、オープンソースの有効なサイクルと「参加の構造」を続けながら、彼らの評価レーティングを潜在的採用者と共有することができます。

このモデルで、我々は評価データの異なるタイプを標準化して、それらをカテゴリーに分類するための提案を提供します。ソフトウェアがかなえなければいけないいかなる使用必要条件にでも、この評価モデルの採択を見込むために、我々は評価のプロセスを 4 つの段階に分けます。

最初に、ソフトウェアパッケージを採用するか、除外するか迅速な評価、そして実行可能な候補のショートリストの作成をします。次に、カテゴリーまたはメトリックスの重要度をランク付けし、データを処理し、最後にデータを BRR に翻訳します。ソフトウェア構成要素の Business Readiness Rating は、1（受け入れ難い）－5（素晴らしい）構成されます。

下記のセクションでは、我々は Business Readiness Rating 概念と、モデルの使用法の概要を紹介します。モデルの使用法に関してのより広範囲な情報は、Appendix1 で参照することができます。



*初期フィルター

オープンソースソフトウェアのコンポーネントのビジネスへの準備状況を評価するために、ユーザーはそのコンポーネントのいくつかの量的かつ質的な特性を見ることから始めます。初期の評価段階において、単純なフィルターによって、採用者がコンポーネントを採用するかしないかを、自信を持って決めることができます。

我々は、この時期においてフィルターとして使うことができるいくつかの指標を認識し

ています。

- ・ソフトウェアのライセンス／法律状況は、どうですか？
- ・それは、標準に従っていますか？
- ・参照可能な採用者かユーザーがいますか？
- ・製品開発と関連するサポート可能な安定した組織はありますか？
- ・開発・実行言語はなんですか？
- ・希望する言語のローカリゼーションはなされていますか？
- ・ソフトウェアを第三者がレビューしますか？
- ・ソフトウェアについての本は出版されていますか？
- ・ガートナー社や IDC 社のような業界アナリストが推奨していますか？

初期評価のための我々のフィルタークライテリアのリストは、すべてを網羅しているわけではありません。ユーザーは、彼らが評価している特定のソフトウェアパッケージや状況にとって重要であるフィルターを付け加えるかもしれません。我々は、より精巧なクイック Assessment ガイドラインを Appendix1 に含めます。

*メトリックスとカテゴリー

迅速な評価プロセスを完了した後に、より詳細な評価段階のために、どのメトリックスとカテゴリーを使うべきか調べることは、重要です。

我々は、オープンソースソフトウェアプロジェクトの評価可能な特性を、メトリックスとして定義します。以下は例です：ソフトウェアについて出版された本の数、プロジェクトに貢献している人数、テストのレベル等。Business Readiness Rating を標準化するために、これらのメトリックスの生のデータを正規化することは、重要です。ソフトウェアパッケージのダウンロードの数のような量的メトリックスは、正規化するのが比較的簡単です。質的なメトリックスも、正規化する必要があります。この過程は主観的でありえますが、我々は本文の Appendix1 でそのようなメトリックスを定量化して、正規化する方法を提案します。

評価プロセスを利害または評価カテゴリーの領域にて系統立てることは、重要です。我々は、ソフトウェアを評価するために、12のカテゴリーを定めました：

評価カテゴリー	説明
機能	どの程度ソフトウェアは、平均的なユーザーの必要条件を満たしますか？
ユーザビリティ	ユーザーインターフェースはどの位いいですか？ エンドユーザにとって、どの位簡単に使えますか？どの位簡単にインストールし、設定し、構築し、保守することができますか？

評価カテゴリー	説明
品質	デザイン、コードとテストは、どの程度の品質ですか？それらは、どれくらい完全で、誤りがないですか？
セキュリティ	どれくらい、ソフトウェアはセキュリティ問題を取り扱いますか？それは、どれくらい安全ですか？
パフォーマンス	パフォーマンスはどうですか？
拡張性	どの位の規模まで使えますか？
アーキテクチャー	どれくらいよく、ソフトウェアは設計されますか？それは、どれくらいモジュール式で、ポータブルで、柔軟で、伸長可能で、オープンで、統合するのが簡単ですか？
サポート	ソフトウェアのコンポーネントはどの程度サポートされますか？
ドキュメント	ソフトウェアのドキュメントはどうですか？
採用度	コンポーネントは、コミュニティ、市場、業界においてどの程度採用されていますか？
コミュニティ	そのソフトウェアのコミュニティは、どの程度活発で、活動的ですか？
プロフェッショナリズム	開発プロセスやプロジェクト組織の専門性のレベルはどの程度ですか？

我々は、カテゴリーレーティングとして、特定の観点からソフトウェアの評価を定めます。カテゴリーレーティングは、同じ観点を評価するいくつかのメトリックスを分類することによって得られます。1つのカテゴリーの評価がどのように計算されるかは、別のカテゴリーが評価される方法と異なるかもしれませんが、結果は同じスケール（1～5）を使わなければなりません。1つのメトリックスは、異なる見方で、いくつかのカテゴリーの一因になるかもしれません：例えば、“Fedora”の6ヵ月のリリースサイクルは、コミュニティの「活発度」の高水準を示しますが、安定性における低レベルを示します。

*機能重視に合わせる

機能重視・・・コンポーネントと、その使用セッティングの合体

我々のモデルにおいて、カテゴリーレーティングは、ソフトウェアの使用必要条件により、その重要性は異なるレベルであるかもしれません。使用必要条件は、2つの要因に由来することができます：

1. コンポーネントのタイプからの使用必要条件

同じ機能を実行するコンポーネントの集合体、例えば Mail Transfer Agent (MTA)、ウェブコンテナ、ウェブブラウザ、オフィススイート。ユーザは、機能性の損失または獲得なしで、コンポーネント内でコンポーネントを交換するかもしれません。

2. 使用環境から

以下は、ソフトウェアコンポーネントが評価される使用環境です。

・ミッションクリティカルでの使用

ソフトウェアは、24 時間 7 日間を稼動していなければなりません。会社のビジネスは

そのソフトウェアに依存しているか、あるいは、そのソフトウェアは、会社によって、ソフトウェアに依存している顧客に提供される製品／システムの一部です。例：Apache (sendmail)。

・ルーチンでの使用

ソフトウェアは、内部的に使われます。アップデートのために回避すること、ないし多少待つ事は、ビジネスに大きな影響を及ぼすことなく許容できます。

・社内での開発／独立系ソフトウェアベンダー

開発グループは、内部使用のため、または顧客のために製品やサービスへの包含のために、社内システムと統合するオープンソースソフトウェアを評価するかもしれません。開発グループは、進行中のサポートに対して責任があるという予想のもとに、ソースコードを修正しているかもしれません。

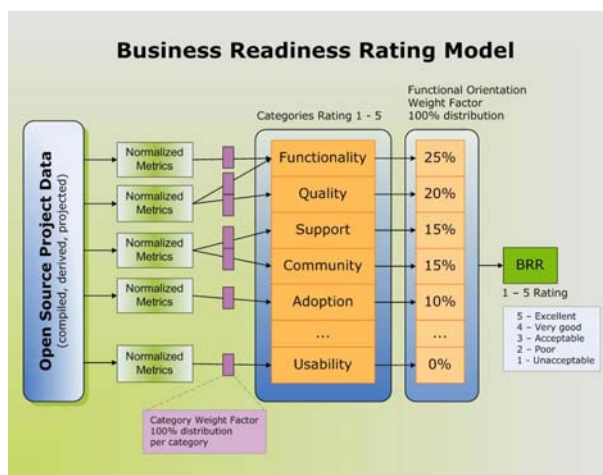
・実験

製品指向ではない目的でのソフトウェアの調査。例は、類似した製品の比較調査、構築の詳細を研究するためのソフトウェアのコンポーネントのモジュールの調査を含みます。そして、研究の結果は、上記の役割のうちの 1 つで使用されるソフトウェアにつながるかもしれません。

使用環境（例えば更なる開発、調査研究やミッションクリティカルインフラストラクチャー）とソフトウェアパッケージのコンポーネント（例えばウェブブラウザまたはデータベース）の並列は、ビジネスレディネスに重要な影響を及ぼします。我々は、要因の組合せをソフトウェアの機能的オリエンテーションとして定義します。

ドキュメンテーション、コードの品質や採用のレベルのような分野のためのカテゴリ一評価は、ユーザーに、理解するのが簡単で正常化されて標準化された数字を提供することができますが、これらの数字は異なる機能的なオリエンテーションと使用必要条件のためのものを意味するかもしれません。オープンソースソフトウェアのコンポーネントは、異なる用途において異なるレベルのビジネスレディネスを備えています。1つのオープンソースパッケージは、企業内の普通の使用のためのビジネスレディネスと考えられるかもしれません。しかし、それはミッションクリティカルなシステムでのビジネスレディネスではまだないかもしれません。

ソフトウェアコンポーネントのための Business Readiness Rating は、ソフトウェアのタイプと使用法によって適切にカテゴリ一評価を加重化することによって計算されます。各々の機能は、限られた数のカテゴリ一レーティングだけに集



申します;ソフトウェアの機能の **Business Readiness Rating** は、それらの選ばれたカテゴリーレーティングに基づいて計算されます。これは、**Business Readiness Rating** が評価で最も重要なカテゴリーを反映することを確実にします。特定の機能（重み要因を選ぶと方法）のために最も影響力があるカテゴリーレーティングを選ぶ方法や、加重化要素を選ぶ方法のベストプラクティスは、**Appendix 1** に記述されています。

***モデルの活用**

ソフトウェアの機能の重要性に従って、迅速評価、メトリックスとカテゴリーの定義付けとランキングを行う事は、ソフトウェアの **Business Readiness Rating** を計算するためのモデルの各々の段階においてとるべきアクションとステップに、我々を導きます。

フェーズ 1

- ー評価されるコンポーネントリストの確認
- ークイック評価基準と各々のコンポーネントの比較
- ーユーザーからの必要条件を満たさないコンポーネントのリストからの削除

フェーズ 2

- カテゴリーの加重化
- ー重要性による 1 2 カテゴリーの順位付け（1 が高、1 2 が低）
- ーそれらのコンポーネントのためにトップ 7 のカテゴリーを選び、各々の重要性に従い%を振り分ける（トータルを 1 0 0 %とする）
- メトリックスの加重化
- ーカテゴリー内での各々のメトリックスに対して、ビジネスのレディネスの重要性に従い、メトリックスの順位付け
- ーカテゴリー内での各々のメトリックスに対して、各々の重要性に従い%を振り分ける（トータルを 1 0 0 %とする）

フェーズ 3

- データの収集と処理
- ー各々のカテゴリーレーティングにおいて使われたメトリックスのデータを集め、各々のメトリックスに当てはまる加重化の計算

フェーズ 4

- データの移行
- ーカテゴリーレーティングと、**Business Readiness Rating** を計算するための機能志向の加重要因の活用
- ーソフトウェアの **Business Readiness Rating** スコアの公表

5. 結論

Business Readiness Rating モデルを提案することでの我々のゴールは、高信頼でオープンなフレームワークを、ソフトウェア評価に対して提供することです。我々のモデルは、組織的アプローチでソフトウェア評価プロセスを速めて、IT マネージャーの間で情報の交換を容易にして、より良い決定となり、高品質のオープンソースソフトウェアに対する信頼を増す事を目指しています。

Business Readiness Rating モデルは、オープンでカスタマイズ可能です。そのため、それはどんなビジネス状況（それは、評価基準の一部を適応させることによって、標準化された方向でプロプライエトリーなソフトウェアを評価するのに用いられさえすることができます）にでも適用されることができます。それは、異なる会社や組織からの IT スタッフが、彼ら自身の評価基準を設定して、彼ら自身の完全な評価をするのに十分完全です。モデルをオープンにすることによって、我々はその漸進な進歩と改善を保証して、その広範囲にわたる採用を増やすことを望みます。他のオープンソースシステムのように、そのユーザーベースが拡大して、**Business Readiness Rating** モデルは進歩します。

我々のゴールは、**Business Readiness Rating** モデルの採用と発展を促進することを助けるための、オープンソースソフトウェアパッケージに関する定量化可能なデータのベンダー中立な情報センターを提供することによって、オープンソースコミュニティのためになることです。<http://www.openbrr.org>で、我々はオープンソースソフトウェアを検討する人の使用のために、**Business Readiness Rating** モデルのチュートリアル、討議フォーラム、サンプル、標準のテンプレートとワークシートを提供します。

Appendix1. Business Readiness Rating (詳細)

ソフトウェアコンポーネントの Business Readiness Rating のスコアは、1 から 5 に分けられます（「受け入れがたい」 1 から「素晴らしい」 5）。そして、「標準」のスコアは3、または「受け入れられる」とします。

I. 迅速な評価のためのベストプラクティス

クイック評価プロセスのゴールは、速くかつ明らかに不適當である製品を除去することです。そのため、検討する人はエネルギーを最も有望な製品を評価することに集中させることができます。我々はそうするためのベストプラクティスを確認します。そして、それは以下です：

1. アプリケーションの目的となる使用方法の決定（ミッションクリティカルなシステムか、レギュラーなシステムか、開発か、または実験か）

どのようにアプリケーションを使うかにより、コンポーネントは現実的なオプションであることもあり、そうではないかもしれません。半成熟しているアプリケーションは、実験的使用のためには現実的であるかもしれませんが、そのようなアプリケーションはミッションクリティカルなシステムや、レギュラーなシステムでは活用できません。したがって、他のステップに移る前に、検討する人が、アプリケーショングループの意図された使用法を評価することは必要です。

2. ターゲットとなる使用方法に基づいた実行可能な指標の選択

SpikeSource 社は、採用するソフトウェアパッケージの実行可能性を示すいくつかの迅速で使いやすい指標を確認しました。我々のリストは、包括的ではありません。これらの実行力指標の一部は、ユーザーに固有ですので、完全なリストを作成することは、実を結ばないこともあります。その上、我々のリストの項目は、全てのターゲットとなる使用方法に関係するというわけではありません。検討する人は、意図しているターゲットとなる使用方法のために意味をなす実行可能な指標だけを選ばなければなりません。これらの指標は、以下を含みます：

ライセンス/法的事項

全てのオープンなライセンスが、同じというわけではありません。検討する人のターゲットとなる使用方法に従い、いくつかのライセンスは、他より非常に制限があります。2つのオープンライセンスが、オープンソースソフトウェアを考えている採用者にとっては重要です：

1) Approved License

もしオープンソースソフトウェアが、Open Source Initiative (OSI)

(<http://www.opensource.org>.) によって認められるならば、オープンなライセンスは”Approved License”と呼ばれます。これらの許可は OSI によって完全に調べられて、

オープンで安全であると思われます。

2) Copyleft License

Copyleft License は、新製品がオープンである限り、コードベースの修正と修正されたバージョンの再分配を許可するオープンソフトウェアライセンスです。Copyleft License は、オープンソースソフトウェアを商品に組み込ませることに興味がある独立系ソフトウェア企業に対して、規制するかもしれません。Copyleft License の詳細については <http://www.fsf.org/licensing/licenses/> を見てください。

オープンソースライセンスに関する一般的な知識は、以下の通りです。

- ① ターゲットとなる使用に関係なく、製品の許可は、Open Source Initiative (OSI) (<http://www.opensource.org>) によって認められる許可のうちの 1 つでなければなりません。
- ② あなたが独立系ソフトウェア企業であり、そして、オープンソースアプリケーションを商用ソフト製品に統合し、包装し直すなら、Copyleft License を使用するコンポーネントに注意して下さい。各々の Copyleft License は、異なる規制をします。Copyleft されたコンポーネントを採用する前に、あなたは規制について知らなくてはなりません。

標準的なコンプライアンス (遵守)

いくつかのソフトウェアカテゴリーにとって、標準の遵守は重要ですが、一方で他のカテゴリーにとっては標準が存在しないかもしれません。業界標準のコンプライアンスが評価されているソフトウェアカテゴリーにとって重要であるならば、我々は検討する人が完全な BRR 評価に移る前に、考慮中のソフトウェアのコンプライアンスを確かめるように提案します。

参照の採用

一部のユーザーは、新技術/製品の初期の採用について熱心です。他は、採用が遅いほうが快適であると感じます。あなたが第 2 のグループに属するならば、他の採用者の参照を探し、彼らの調査結果に対するあなたの満足度を定量化しようとしてください。

サポートする組織または安定している組織の活用

あなたの組織内部にサポートを提供するための資源がないならば、我々はあなたが専門のサポートが存在するアプリケーションだけを考慮するように提案します。専門のサポートの必要性は、サポートするコミュニティの存在によって改善されるかもしれません。

構築言語

しばしばオープンソースソフトウェアを採用するには、若干のカスタマイズ作業と社内のサポートを必要とします。社内で利用できるコーディングの専門性によってアプリケーションを選ぶことは、重要です。

第三者のレビュー

採用者は、考慮しているソフトウェアに関して、第三者のチェックを調査したいかもしれません。

本

ソフトウェアに関する本の入手は、ソフトウェアの成熟度と採用度のレベルの高い指標です。

業界アナリスト（例えばガートナーまたは IDC）

オープンソース採用のためのもう一つの指標は、ソフトウェアについてのガートナーや IDC などの市場調査会社のアナリストによる研究レポートの入手です。

3) リストへの更なら実行可能な指標の追加

特定の状況に従い、採用される新しいソフトウェアパッケージは、いくつかの重大な必要条件を満たす必要があるかもしれません。これらの必要条件は、迅速な評価のために実行可能な指標のリストに含まれなければなりません。

4) クライテリアを満たす方針の作成

迅速な評価のために実行可能な指標のリストを作成した後に、採用者はどんな評価結果が許容できるかという方針をつくる必要があります。我々が選んだ大部分の実行可能基準は肯定であるか否定的な答えを提供します。そして、赤または緑、そしておそらく中間の黄色のカテゴリーに分けられます。許容できる方針は、「全て緑」または「黄色が2つ以下」です。

5) 実行可能な指標のリストに対する各々のソフトウェアコンポーネント評価

適用できる実行可能なフィルターと合格する方針のリストを作成して、検討する人はソフトウェア候補のために迅速な評価を実行し始めることができます。迅速な評価の結果は、候補ソフトウェアが完全なビジネスレディネス評価プロセスに移らなければならないかどうか決定することができます。

II. ターゲットとなる使用方法評価のためのベストプラクティス

モデルは12の評価カテゴリーを提供しますが、それらの全てにおいてソフトウェアを評価することは必ずしも賢明ではありません。ソフトウェアの機能により、評価カテゴリーの関連性は、データ収集の努力のために、時間的に効率的であることは十分ではないかもしれません。さらにまた、最も重要な評価カテゴリーは、より重要でないカテゴリーによって薄められるかもしれません。この問題を避けるために、我々は、検討する人が7つの評価カテゴリーにしか集中しないように提案します。範囲の縮小は、また、各々の領域からカテゴリー評価加重貢献を決定する際に、検討する人の役に立つかもしれません。我々は、各々の機能重視の要因を決定するために、以下の仕事の流れを提案します：

1. 最も重要な評価カテゴリーの選択

我々のエクササイズの間、我々は考慮から評価カテゴリーを除くことが精神的に難

しいとわかりました。検討する人が、良い評価を全ての評価域に期待することは自然です、そして、それはどんなカテゴリでも除くことにためらいます。しかし、我々は全ての評価カテゴリを含むことが逆効果であると思っています。我々は、検討する人に以下を提案します。

－ソフトウェアの機能性に関して、1～12の評価カテゴリの重要性をランクしてください。

－ランクを付けられ分類された評価カテゴリリストで、分割点（切り離すポイント）を決めてください・・・1つのカテゴリと次のカテゴリの間の重要性の違いが大きい（例えば「むしろ重要な」と「持つのに良い」）点で。検討する人は、分割点より上にあるカテゴリに集中し、他のカテゴリは捨てなければならない／無視すべきです。我々は、検討する人が、7カテゴリまたはより少ない評価カテゴリに集中するように提案します。

2. 適切な加重要因の選択

検討する人が、集中する7またはそれ以下の評価カテゴリを決めたならば、次のステップは、各々のカテゴリがどのくらい最終結果に役立つかについて確定することです。これらの貢献レベルは加重要因です、そして、それらはパーセンテージで表されます。全ての加重要因は、合計100%にならなければなりません。加重要因の良い分布を決定するために、あなたには7つの評価カテゴリがあると仮定して、我々はその検討する人が以下を行うこと提案します：

－上のステップ1で引き出される重要性のそれらのランキングに基づいて使われる評価カテゴリを分類してください。

－加重平均要因を中央のランキング（分類されたリストの中央）を持つカテゴリに割り当ててください。

－残りのカテゴリに、加重を割り当てるために、「ジグザクのパターン」を活用ください。例えば、検討する人が7つのカテゴリに集中するならば、第4位にランクされるカテゴリは、最初の15%の加重を得るでしょう。その後、リストの上位にあるカテゴリ（重要性で1位から3位）が15%以上の加重を割り当てられ、そして、リストの下位にあるカテゴリ（重要性で5位から7位）は15%以上は割り当てられません。7つのカテゴリの良い加重分布のサンプルは、以下の通りです。

最重要なカテゴリ：25%

2番目に最も重要なカテゴリ：20%

3番目に重要なカテゴリ：15%

4番目にランクを付けられた中央のカテゴリ：15%

5番目にランクを付けられたカテゴリ：10%

6番目にランクを付けられたカテゴリ：10%

7番目にランクを付けられたカテゴリー：5%

- －加重の合計は100%です。
- －必要なら、最終的な調整をしてください。

III. データの収集と処理のためのベストプラクティス

データの収集と処理のフェーズは、**Business Readiness Rating** モデルにおける最も時間のかかるフェーズです。十分な忍耐は、評価のためにより良いデータを得ることができるので、非常に価値があることです。我々は、データが、成熟したソフトウェアパッケージのためにより簡単に入手できることを知っています。したがって、実行はまた、ソフトウェア品質の最初の指標でありえます。

*メトリックス測定の標準化

定性的であるか定量的であるかどうかに関係なく、カテゴリーの範囲内の全ての測定は、測定が意味があると考えられる標準化されたスケールと比較する必要があります。

例えば、ソフトウェアコンポーネントが1ヵ月につき2000のダウンロードされるなら、「月のダウンロード数」メトリックスを見ることは、よいことなのでしょうか？それとも悪いことなのでしょうか？どのように、それはスケールの範囲内で評価されますか？

- 1 – 0 to 499 downloads/month – 受け入れがたい
- 2 – 500 to 999 downloads/month – プアー
- 3 – 1000 to 1999 downloads/month – 許容できる
- 4 – 2000 to 2999 downloads/month – とてもよい
- 5 – 5000 or more downloads/month – すばらしい

もし我々が「出版される全体の本」を見て、可能なスケールとスコアは以下です：

- 1 – 0 books – 受け入れがたい
- 2 – 1 to 2 books – プアー
- 3 – 3 to 5 books – 許容できる
- 4 – 6 to 15 – とてもよい
- 5 – 15 or more books – すばらしい

メトリックスは現在標準化されて、カテゴリー評価を計算するのに用いられることができます。

大部分のメトリックスの標準化を単純化するために、可能な時はいつでも、我々は1つの一般的なスケールに対して正確であるよう努力します：5つのスケールを **Unacceptable**、**Poor**、**Acceptable**、**Very Good** と **Excellent** に分けます。

定量的メトリックスのために、生の数とスコアを1-5のスケールにマップすることは、あまり難しくはありません。定性的なメトリックスのために、我々は全てのメトリックスが範囲（幅）で測られることができるというわけでないこと理解しています、あるいは、も

し可能ならば、メトリックスの範囲は5のスケールに適合しないかもしれません。範囲(幅)で測られることができないメトリックスのために、我々は以下を提案します：

I. 二者択一式(イエスまたはノー)メトリックス

メトリックスに対する否定的な答えは、1(受け入れがたい)となります。そして、肯定のものは、3(許容できる)または5(素晴らしい)となります。肯定の反応のためのスコアは、それが否定的なものと比較してどれくらいポジティブか(どのくらいよりよいか?)について判断されなければなりません。

例1：セキュリティ：セキュリティサイト/Wikiは利用できますか？

いいえ=1 はい=5。 CERTによってモニターされるソフトウェアパッケージは、そうしないものより非常に高いセキュリティ品質があります。

例2：セキュリティ：セキュリティサイト/Wikiは利用できますか？

いいえ=1 はい=3。 セキュリティサイトが、開発者がソフトウェアのセキュリティ品質に注意を置いていることを意味します。しかし、サイトだけで十分ではありません。したがって、肯定の答えは、否定的なものよりそれほどよくはありません。

II.

いくつかのメトリックスは範囲内で測定可能です、しかし、それらは5段階のスケールの上では測定されることができません。そのようなメトリックスのために、我々は、「ことば」のスコアを使うことを提案します、使っていない「スペアのスコアの数」を、残します。あるいは、2つ以上の答え/数を、1つのスコアナンバーに組み込みます。

例1：「中心的な開発者チームに入ることの難しさ」というメトリックス

- ・誰でも貢献できる・・・1(受け入れがたい)
- ・かなり難しい・・・2(プアー)
- ・非常に難しい・・・5(すばらしい)

我々は、上記2つのスケールリング方法が、全てのメトリックスのために機能するというわけではないかもしれないと理解します。実際、我々は機能評価に貢献するメトリックスを測るために、急進的に異なる方法を提案します。しかし、我々は論評家が自由に新しいものをつくる前に、これらの2つのスケールリング方法を使用してみる努力をすることをお願いします。新しいスケールリング方法が考案される必要があるならば、同じカテゴリー評価に影響を及ぼす他のメトリックスと比較して新しいスケールの一貫性を考慮してください。

これらのメトリックスの一部が、不正確である点に注意してください。例えば、ダウンロードメトリックスは、もしソフトウェアが他のソフトウェアパッケージの一部であるなら、妥協されています・・・多くのオープンソースコンポーネントがLinuxに含まれています。評価のスコアは、そのような状況に対応するために調整される必要があるかもしれません。

*機能マトリックスの処理

機能性は、他のカテゴリと違って、計算することができる評価カテゴリです。各々の種類のソフトウェアアプリケーションは、ソフトウェアパッケージによって満たされる必要がある独特の特徴を持ちています。その上、若干のソフトウェアパッケージは、「プラスの点」と考えられることができる特徴を提供するかもしれません。これらの違いのため、我々は、機能的なレートを得るため機能カテゴリとなる処理とマトリックスとなる独特な方法を考案しています。

機能性評価は、最初に、評価するコンポーネントの特徴と、平均的使用のために必要な標準の特徴を比較することによって得られます。この標準の特徴は、外部資源（ソース）から構成されなければならないか、借りられなければならない。たとえば、CMS マトリックス (<http://www.cmsmatrix.org>) によって定義されて特徴は、コンテンツ・マネージメント・システムの基礎です。一旦標準の特徴が可能になるなら、検討する人は以下のステップを活用して、特徴評価を開始することができます：

- 重要度のスコアを特徴リストの全てのアイテムに割り当ててください。（あまり重要でない 1～とても重要 3）
- コンポーネントの特徴リストを、標準の特徴リストと比較してください。各々の特徴が満たされれば、重要性スコアを累積合計として加えてください。満たされないならば、重要性スコアを合計から差し引いてください。
- もしソフトウェアが標準の特徴リストにない余分の特徴を持つなら、それらの特徴のための重要性スコアを割り当て、スコアを合計に加えてください。
- 累積合計を、標準の特徴だけによって得られることができる最大の得点で、割算してください。この比率は、特徴スコアと呼ばれています。このシナリオを使って、100%超または 0%未満の特徴スコアを得ることは可能です。これは意図的に、余分の特徴に「リワード」を与え、欠けている標準の特徴のために「罰」を与えています。
- このスキームを使い、1-5 のスケールに、特徴スコアを標準化してください。
 - 65%以下のスコア = 1 (受け入れがたい)
 - 65% - 80%のスコア = 2 (悪い)
 - 80% - 90%のスコア = 3 (許容できる)
 - 90% - 96%のスコア = 4 (とてもよい)
 - 96%超のスコア = 5 (すばらしい)

*加重要因のカテゴリレート

各々のカテゴリ内での各々のマトリックスは、特定のカテゴリの範囲内でマトリックスの重要性の差別化のために、加重要因を持たなければなりません。単純性のため

に、カテゴリーの範囲内のメトリックスのための加重要因の分布は、機能加重要因と同じ方法に従います（この Appendix で「適切な加重要因の選択」にあります）。通常、カテゴリー評価に寄与しているメトリックスが7つ以下の場合がしばしばあるので、メトリックスのための加重分布は、カテゴリーよりも簡単に決められることができます。

IV. 典型的なメトリックスとスコア

これらのメトリックスとスコアは決定的なセットではありません。しかし、単に BRR モデルを例示するための典型だけではありません。我々は、各々のカテゴリーでビジネスレディネスを測るためのメトリックスのより良い候補があると理解します。それらのために、データは集めるのが難しいか、とても利用できないです。我々はモデルの使用法を確認するために、最初のセットを提供しました、そして、我々は Business Readiness Rating モデルの将来のバージョンでメトリックスの改善をし続けます。

カテゴリー	メトリックス	詳細	スコア				
			5- すばらしい	4- とてもよい	3- 許容	2- プアー	1- 受け入れ難い
ユーザビリティ							
	エンドユーザのユーザインターフェースの経験	ユーザインターフェースがエンドユーザによりどのように認識されているかを測ります。 (直観的なインターフェースやナビゲーション/コントロールスキーム)	単純性と直観性 情報は整理されており、マニュアルを必要としない		学ぶために多少時間が必要、情報はある程度整理されている、たまにマニュアルが必要		複雑、とてもたくさんの情報、明らかに整理されていない、マニュアルなしでは使えない
	OSSをインストールするために事前に必要とされるセットアップのための時間	すべての事前要件を満たし、システムをセットアップする時間 (OSは含まない)	10分以下	10~30分	30分~1時間	1~4時間	4時間以上
	バニライnstレーション/コンフィギュレーションのための時間	即座の満足を得る時間 (プロジェクトが、ユーザーがUpし、運用することについて考えているかどうかに関係なく)	10分以下	10~30分	30分~1時間	1~4時間	4時間以上
品質							

カテゴリー	メトリックス	詳細	スコア				
			5- すばらしい	4- とてもよい	3- 許容	2- プアー	1- 受け入れ難い
	過去12ヶ月でのマイナーリリースの回数	計画されたアップデートやバグフィックスを測定する。商用製品のコマーシャルパック	2回		1~3回		0または3回以上
	過去12ヶ月でのパッチリリースの回数	パッチ等が、メモリーやセキュリティの弱点、デッドロックのような P1 クラスのバグを修理する	3~4回		1~2回または5~6回		0回または6回以上
	過去6ヶ月間のオープンバグの数	製品使用の品質を測る	50以下	50~100	100~500	500~1000	1000以上
	過去6ヶ月間のバグフィックスの数（オープンバグとの比較）	どのくらい早くバグがフィックスされるかを測る	75%以上	60~75%	45~60%	25~45%	25%以下
	P1/クリティカルなバグがオープンである数	見つかった品質の問題の深刻さを測る	0	1~5	5~10	10~20	20以上
	過去6ヶ月間でのP1バグの平均エイジ	クリティカルな問題に対する責任感	1週間以内	1~2週間	2~3週間	3~4週間	4週間以上
セキュリティ							
	過去6ヶ月の間の、中位または非常にクリティカルなセキュリティの弱点の数	セキュリティの弱点に関する品質を測る。ソフトウェアがセキュリティの弱点にどの位影響されるか	0個	1~2個	3~4個	5~6個	6個以上
	セキュリティの弱点が未だにオープンである数（パッチが当てられていない）	すべてのセキュリティの問題を解決できるプロジェクトの能力を測る	0個	1個	2個	3~5個	5個以上
	セキュリティに関する専属の情報（Web Page とか Wiki 等）はあるか？	セキュリティの問題をどの位気にして、真剣であるかを測る	はい、よく維持されている		はい		いいえ

カテゴリー	メトリックス	詳細	スコア				
			5- すばらしい	4- とてもよい	3- 許容	2- プアー	1- 受け入れ難い
パフォーマンス							
	パフォーマンステストとベンチマークレポートが入手可能	パフォーマンス・テストが実施され、ベンチマークが公表されているかどうかを測る（他の同様なソリューションとの比較）	はい、よい内容です		はい		いいえ
	パフォーマンス・チューニングとコンフィギュレーション	パフォーマンスのためにコンポーネントをよくチューニングするためのツールやドキュメントがあるかどうかを測る（CPU, DiskやNetworkの情報）	はい、広範囲にわたります		はい、いくつかは		いいえ
拡張性							
	構築のリファレンス	ソフトウェアの拡張性と、現実の世界での現実のユーザがテストしたかを測る	はい、ユーザのサイズも含んでいます		はい		いいえ
	拡張性を持ってデザインされている	コンポーネントが拡張性を持ってデザインされているかを測る (それは脅威からは安全ですか？それはクラスター環境でも動くか？H/Wはパフォーマンスの問題を解決できるか？)	はい、広範囲にわたって		はい、いくつかは		いいえ
アーキテクチャー							

カテゴリー	メトリックス	詳細	スコア				
			5- すばらしい	4- とてもよい	3- 許容	2- プアー	1- 受け入れ難い
	プラグ・イン可能な第三者のソフトはあるか?	プラグ・イン可能な第三者のソフトを通して、伸張性を測る	10以上	6-10	2-5	1	0
	パブリック API/内部サービス	パブリック API を通して伸張性を確認 また、カスタマイゼーションのためのデザインを示す	はい、広範囲にわたって		はい		いいえ
サポート							
	過去6ヶ月間の一般的なメールの平均数	この一般的なメーリングリストは、人々がヘルプを得る最初の場所です	720メッセージ以上/月	300~720メッセージ/月	150~300メッセージ/月	30~150メッセージ/月	30メッセージ以下/月
	プロフェッショナルサポートの品質	トラブルシューティングがいつも期待通りで、ローカルでの構築のためのファインチューニングのヘルプとなるプロフェッショナルサポート	インストラクション/トラブルシューティング/インテグレーション/カスタマイズのサポートがある		インストラクションサポートのみ		プロフェッショナルサポートはない
ドキュメント							

カテゴリー	メトリックス	詳細	スコア				
			5- すばらしい	4- とてもよい	3- 許容	2- プアー	1- 受け入れ難い
	色々な種類のドキュメントの存在	よいドキュメントは、色々なフォーマットで、色々なユーザーグループのためのドキュメントを含む	インストール／構築／ユーザー／アドミニ／最適化／アップグレード／開発のドキュメントが、色々な種類のフォーマット（pdf、single htm、multifile html 等）で入手可能	インストール／構築／ユーザー／アドミニ／アップグレードのガイドが色々な種類のフォーマットで入手可能	インストール／構築／ユーザーのガイドが入手可能	テキストベースのインストールレーションドキュメントのみ存在	適切なドキュメントがない README File がない
	ユーザのコントリビューション（貢献）のフレームワーク	最高のガイドはしばしばユーザからのインプットやサンプルから成り立ちます これは、製品を使った人からのフィードバックです	人々は貢献する事が許されており、貢献物は、専門家により編集されフィルターされている		人々は貢献する事を許されている		ユーザは貢献する事ができない
採用度							
	アマゾンで本のタイトルがどのくらいあるか 表題：コンピューター タイトル：コンポーネント名	本があることは非常によいことです	15タイトル以上	6-15タイトル	3-6タイトル	1-3タイトル	0タイトル
	構築のリファレンス	ソフトウェアの拡張性と、現実の世界での現実のユーザがテストしたかを測る	はい、ユーザのサイズも含まれています		はい		いいえ
コミュニティ							

カテゴリー	メトリックス	詳細	スコア				
			5- すばらしい	4- とてもよい	3- 許容	2- プアー	1- 受け入れ難い
	過去6ヶ月間の一般的なメールの平均数	この一般的なメーリングリストは、人々がヘルプを得る最初の場所です	720メッセージ以上/月	300~720メッセージ/月	150~300メッセージ/月	30~150メッセージ/月	30メッセージ以下/月
	過去6ヶ月間での独特のコード貢献者の数	コード貢献者はプロジェクトのコミュニティの構築を推進します コード貢献者の数が多いほど、コミュニティやサポートがよいことを意味します	50人以上	20~50人	10~20人	5~10人	5人以下

プロフェッショナリズム							
	プロジェクトドライバー	プロジェクトドライバーは、プロジェクトマネージメントやリソース（お金）を集めることを実行します	会社(ApacheやOSDLスタイル)によりサポートされている独立の基金（ファンデーション）	会社 (MySQLスタイル)		グループ	個人
	コアの開発チームに入るのが困難	ソフトウェアの品質を保つため、コミッター（作業権限を持つ人）を受け入れることにおいて、成熟したプロジェクトは選択できます 新しいプロジェクトは選択肢がありません	外部においてしばらくの間行動したあとのみ		かなり困難 何回か許容できるパッチに貢献しなければならない		誰でも入れる

Appendix 2. 他の情報

I. 成熟したオープンソースソフトウェアの特徴

以下は、成熟したオープンソースソフトウェアの典型的特徴です。それらは、大部分の状況にあてはまりますが、すべてではありません。

1. 開発の分離と、安定した枝分かれ
2. ソフトウェアは、基金、会社または強いコミュニティにバックアップされている
3. コミュニティは、各々が責任のあるグループに組織されている（保守者、ドキュメンテーショングループ、開発グループ、外部への発信グループ）
4. プロジェクト機能拡張は利用可能
5. プロジェクトは少なくとも1年は存在している
6. コアの開発チームに入るためには、よく定義されたプロセスがある
7. プロジェクトのライセンスは、**Open Source Initiative** によって認められている
8. ドキュメンテーションの分離：ユーザードキュメント、インストレーションドキュメント、アドミニストレーションドキュメントと、開発ドキュメント
9. ユーザードキュメントは広範囲で、多くのフォーマットで利用できる
10. メーリングリストの分離：ユーザーメーリングリスト、開発者メーリングリスト、セキュリティメーリングリスト、外部発信者メーリングリスト、その他
11. マイナーまたはメジャーなリリースをすることにおいて積極的でない。速いポイントリリースはいいことです。（メジャー、マイナー、ポイントのために、番号付けシステムを公表してください。）
12. 本は、すぐに利用できる
13. ソフトウェアの大規模な採用と使用は、組織や個人により存在
14. ソフトウェアのコンポーネントは、理にかなったユニット／機能テストがなされており、そしてこれらのテストのためのコード範囲は理にかなっている（30－80%の範囲）
15. コンポーネントは、含んでいる／含まれたコンポーネントとよく連携されている必要がある
16. コンポーネントのバグデータベースは、リビジョン番号や修理された各々のバグを示さなければならない
17. ソフトウェアは、インストールするのが簡単で、インストールインストラクションのドキュメントがある
18. ソフトウェアは、きれいなユーザインタフェースがある（GUI またはコマンドライン）
19. パフォーマンスメトリックスは利用できる

- 2 0. 構築ガイドは利用できる
- 2 1. 有名な大規模な構築（例えば mediawiki のための Wikipedia）
- 2 2. 使用のための直観性と複雑ではないデザイン（例えば MoinMoin 対 Tikiwiki）
- 2 3. 複数のプラットホーム（Linux、Windows、Solaris と Mac）に移植されている
- 2 4. 押し付けがましくない、例えば小さな Runtime
- 2 5. セキュリティパッチとバグフィックス配信の分離、そして新機能／エンハンス

II. 参照

Top Tips for Selecting Open Source Software

<http://www.oss-watch.ac.uk/resources/tips.xml>

How to Evaluate Open Source Software / Free Software (OSS/FS) Programs

David A. Wheeler

http://www.dwheeler.com/oss_fs_eval.html

Choosing Open Source, A Guide to Civil Society Organizations

Mark Surman and Jason Diceman Jan 06 2004

<http://www.commonsworld.org/articles/fulltext.shtml?x=335>

Free and Open Source Software Overview and Preliminary Guidelines for the Government of Canada

http://www.tbs-sct.gc.ca/fap-paf/oss-ll/foss-llo/foss-llo17_e.asp

CapGemini's OSMM

<http://www.seriouslyopen.org/nuke/html/index.php>

Golden's OSMM

<http://www.navicasoft.com/pages/osmmoverview.htm>

Johnson, K.; A descriptive Model of Open-Source Software Development

<http://sern.ucalgary.ca/students/theses/KimJohnson/toc.htm>

Dalle, M.J., Jullien, N.; "Open Source vs. Proprietary Software"

<http://opensource.mit.edu/papers/dalle2.pdf>

CMS Matrix

<http://www.cmsmatrix.org>

**Translated by
Kiyoshi Tanaka
Business Planning Group
CEC, Ltd.**