GitHub

開発ツール社内導入攻略ガイド

－ツール選定と技術検証－

はじめに

IoT、AI、コネクテッド・カー、フィンテックなどのキーワードに代表されるよう、昨今、「ソフトウェア化」はすべての産業において必須事項となりました。そのような背景から、ソフトウェア開発環境の整備・改善は企業にとって重要な課題であるにもかかわらず、多くの企業ではまだ十分に進んでいません。

その要因の１つとして、開発環境が開発者たちに及ぼす影響が軽視され、結果として組織が開発環境を整備するシステム管理者たちに十分にコストを割いていないことが挙げられます。開発環境の整備･改善は、単純なサーバ管理ではありません。技術が進化するように、組織が変化するように、最適な開発環境も日々変化します。そのような中で、開発環境の整備・改善を目指し、最適な開発ツールの選定と導入にチャレンジすることは容易ではありません。技術に対する感度、またその技術に対応するための多種多様な技術スキルを要します。それに加えて、組織にとって最適なツールを見極めるための冷静な判断力と迅速かつ戦略的な決断力が必要です。また改善を推し進めるための推進力も必要でしょう。

本稿は、そうした開発環境の整備・改善を行う管理者を支援するために、開発ツール管理者であった私の実体験を基に「開発ツール社内導入攻略ガイド」としてまとめました。本ガイドの前半では、開発ツールを導入する際にそのツールの選定と技術検証で確認すべきポイントについて、後半では、導入した開発ツールを浸透させることの重要性と浸透させるために管理者が心がけることについて紹介します。

開発環境の整備により開発フローを効率化することで、開発者から無用な足かせを外し、素晴らしいプロダクトが世の中に沢山生まれることを願います。

GitHub
ビジネスサポートエンジニア
鈴木 順子

GitHub

目次

GitHub

第 1 章

開発ツールを導入する際に考えるべきポイント

まず組織に新しく開発ツールを導入する際の大事な2つのポイントを紹介します。

1つ目は「開発ツールの選定と技術検証」。開発ツールは世の中に数多く存在しますが、自社に最適なものがどれかを判断することはとても難しいことです。特に近年はインターネットで簡単に情報が入手できますが、逆にどの開発ツールを導入すべきかの判断が迷走してしまいがちです。「本当に自分の組織に導入すべき開発ツールはどういうものか」をしっかり見極めて選定することが大切です。選定したツールを技術検証する際は、確認すべきポイントを事前に用意しておくと、より効率的良く、効果的に検証できます。

2つ目は「組織へのツールの浸透」。どんなに素晴らしい開発ツールを導入しても使われなければ意味がありません。開発ツールを利用するユーザの中にはエンジニアだけでなくプランナーやデザイナーなどスキルセットが異なる人たちがいます。そうした人たちに日常的に使ってもらえるように、どう工夫するかが重要な鍵となります。「組織へのツールの浸透」については、攻略ガイドパート2「ツールを社内に浸透させるためのベスト・プラクティス」をご覧ください。

第 2 章

開発ツールの選定と技術検証

理想の開発環境をイメージすること

開発ツールを選定する前にまず最初にやるべきことは、可能な限り具体的に「理想の開発環境をイメージすること」です。

新しい開発ツールを組織に導入する際の動機は、必ずしも「組織におけるソフトウェア開発環境を改善する」こととは限りません。「最近流行っているツールを使ってみたい」という理由だけで導入してしまう場合があります。技術に対する感度の高いエンジニアほど、話題のツールがあるのなら、どんな機能があるのか知りたい、検証してみたい、他社が使っているならば自分の組織でも使ってみたいという気持ちが強く働きます。話題のツールは、それだけの良い部分が沢山あります。しかし、必ずしも組織のニーズにあっているとは限らないということに注意が必要です。また、それがどんなに素晴らしいツールであっても、組織にどのような利益をもたらすかを具体的に説明できないと、必要なコストに対して決裁者から承認を得られません。

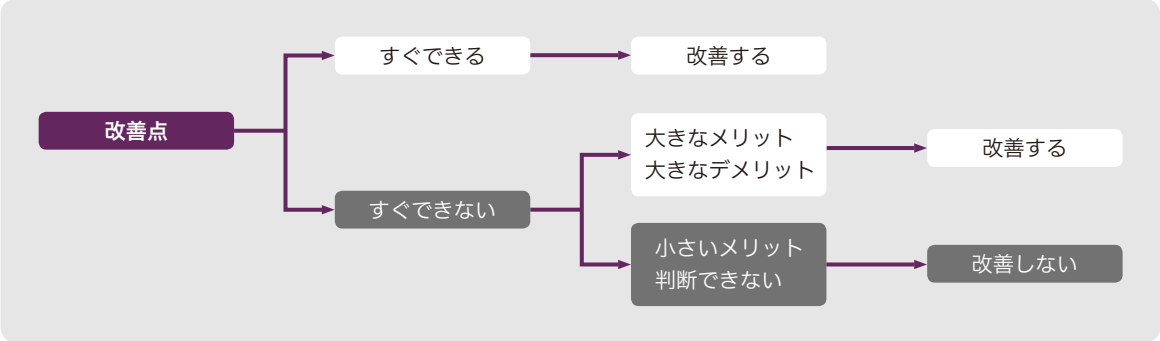
現状の非生産的な環境を改善するという観点からツールの導入を検討するケースもあります。この場合、「あれも改善すべき、これも改善すべき」と不便なことを漠然と沢山思い浮かべて、どれから手を付けていいのか分からなくなるという落とし穴がある点に注意してください。結果としてやるべきことの大きさにストレスを抱え、具体的なアクションを起こすまでモチベーションが維持できず、導入の提案にまで至らないといった結果に陥りかねません。

一番重要なことは、まず最初に、可能な限り具体的に、「理想の開発環境をイメージすること」です。その次に、

理想の開発環境と現在の開発環境とを比較します。そうすることで、具体的に何を改善すべきかが自ずと見えてきます。さらに、最終的に目指す開発環境を明確にイメージできます。

改善点が見えたら、それを「改善する」か「改善しない」かを決定します。ここでのポイントは、「すぐできる」か「すぐできない」かです。ここで「すぐにできる」とは、数日から数週間で完了する道筋が見えるものを指します。例えば、「数日でスクリプトを書いたらできるもの」、「簡単な調整で変えられること」などがあるでしょう。「すぐにできること」は、やらないという判断はありません。すぐに実施してください。次に、「すぐにできないこと」を改善するかしないかを判断します。「すぐにできないこと」は、時間・人・お金のようなコストが大きくかかるものを指します。本ガイドのトピックである「新しくツールを導入する」ことも「すぐにできないこと」に分類されます。

「すぐにできないこと」を「改善する」か「改善しない」かの判断ポイントは「大きなメリットを得られる」か、あるいは「大きなデメリットを取り除くことができる」かです。重要なのは、「大きな」という点です。このどちらも思いつかない場合は、やらないという選択でよいでしょう。小さなメリットしか得られない、または効果の判断がつけられないのであれば、やらないという選択をしても構いません。



こうして情報を整理整頓すると、自然にどのようなツールを入れるべきか、候補ツールのリストが見えてくるはずです。

3つの役割の懸念点からチェックリストを作成する

候補ツールをリストアップしたら、候補の中からどれを検証フェーズへ進めるかを判断するためのチェックリストを作りましょう。

チェックリストは、“ 3つの役割の人の懸念点を解消する ” という観点でリストを作成します。3つの役割とは、「ユーザ（開発ツールを使う人）」、「マネージャ（導入を承認する人、または決裁権を持つ人）」、そして「管理者（開発ツールを管理するシステム管理者）」です。

まずは、ユーザの懸念について見てみましょう。ユーザは、第一に“自分たちの現在のプロジェクトに支障が出ないか”を考えます。ユーザは既に何らかのプロジェクトを抱えており、日常的に開発を進めています。そのため、現在稼働中の業務への影響を最小限に抑えることはとても重要です。つまり、主に次の2つのコストを心配します。

1つは学習コストです。どんなに便利なツールを導入しても、使い方を学ばなければなりません。それだけではなく、他のプロジェクトメンバーに教えてあげる必要があるかもしれません。通常の開発業務に並行して新しいツールを習得する負担は大きな懸念点となるでしょう。学習コストの負担を軽減するために、ユーザ向けの公式ドキュメントが充実しているか、そのツールを学ぶためのワークショップやイベントが開催されているか、などを確認しましょう。

もう1つは移行コストです。現状使っているツールから新しいツールに移行する際に発生する業務負担は小さくありません。CIやCDなどの連携ツールの変更やデータ移行が必要になるケースが考えられます。移行のタイミングも調整が必要でしょう。可能な限り、ユーザの移行コストを軽減する方法を検討しましょう。

さらに、開発ツールの性質によりますが、クライアントが必要なものに関しては、WindowsやMac両方のクライアントが存在するかを確認しましょう。特に利用ユーザにエンジニア以外の職種が含まれる場合は重要になることがあります。

以上のことから、ユーザの懸念点を解消するためのチェック項目は以下です。

- ☐ ユーザ用公式ドキュメントが充実しているか？
- ☐ ツールを学ぶためのワークショップなどのイベントが開催されているか？
- ☐ 新しいツールへの移行が簡単か？
- ☐ 連携ツールのサポートが充実しているか？
- ☐ データ移行のための機能が提供されているか？
- ☐ WindowsやMacの両方にクライアントは対応しているか（ツールの性質に寄る）？

それでは、マネージャの懸念点について考えましょう。マネージャが第一に考えるのは、開発者たち（ユーザ）の負担です。新しいツール導入の学習コストが高く、ツールを使いこなせないのでは、プロジェクトが滞り業務に支障が出るからです。この部分のチェックポイントに関しては前述しました。

次に、導入コストに対してどれだけ効果があるかです。まずは、類似ツールの導入コストの相場観を抑えておきましょう。開発ツールは、金銭的コストのインパクトが大きいものが多くあります。類似ツールの相場を確認し

GitHub

ておくと、その金額が妥当なのかを判断しやすくなります。

また、その開発ツールを導入することによる具体的な効果を、決済権のあるマネージャがしっかりとイメージできることが大切です。新しい開発ツールの導入による効果として、まずは不要な業務を取り除いたことによるお金や人的リソースといったコストの削減があるでしょう。これについては、最初に理想の開発環境と現在の開発環境とを比較した段階でまとまっているはずです。さらに、必要な時にしっかりとマネージャに共有できるように、開発ツールに関する開発者からの評判もしっかりと抑えておきましょう。開発者のモチベーションがあがれば、より生産性の向上につながります。

業務の効率化だけが、その開発ツールから受ける恩恵ではありません。開発ツールを導入することによる広告効果についても念頭に置いておきましょう。導入するツールによっては、社外への技術アピールになったり、優秀なエンジニアを採用するための広告になり得ます。

さらに、セキュリティがしっかりと担保されていることは大事なポイントです。開発ツールのセキュリティが担保されていないと、開発したプロダクトを脅かす事態になりかねません。また、ユーザの管理機能も重要です。不要なユーザが残ったままになると、セキュリティの問題だけでなく、ライセンス料などの導入コストも余分にかかってしまいます。ネットワークのアクセス制御やアカウントの管理方法など、開発ツールへアクセスするユーザの制御方法について確認しましょう。

以上のことから、マネージャの懸念点を解消するためのチェック項目は次のようになります。

- ☐ 導入コストとその効果
  - ● 類似ツールの相場
  - ● 社内業務の効率化
- ☐ 社内インパクト
  - ● ユーザのモチベーションアップ
- ☐ 社外インパクト
  - ● 外部アピールできるか
- ☐ ネットワークなどアクセス制御ができるか
- ☐ アカウント管理のしやすさ

最後に、管理者の懸念について考えていきましょう。開発ツールの検証・導入が決まった後、それを日々運用するのは管理者です。開発ツールの管理は、開発ツールの仕様把握やシステム管理まで、多種多様な知識が必要です。さらに開発ツールを利用するエンジニアやデザイナーといったユーザたちの様々な利用ケースに応じて、ユーザに寄り添いながら柔軟に対応をしていかなければなりません。しかし、開発ツールを管理する組織には十分なリソースが割かれるケースは少ないのが現状です。そのため開発ツールの管理者は多くの負担を抱える傾向にあります。管理者への負担が大きくなりすぎると、結果的に、検証に支障が出たり、せっかく良い開発ツールを導入しても組織に浸透しないままになってしまいます。

開発ツールの検証を始める前に、可能な限り事前に管理者が抱えるであろう負担を解消しておきましょう。

まず第一にユーザとマネージャの懸念点をしっかり解消できる状態にあることが前提になります。この部分がしっかりできていないと、ユーザやマネージャの不満が溜まり、その不満は管理者に向くことになります。そうなると管理者に負担が大きくのしかかります。前述したユーザとマネージャの懸念点を解消するポイントはしっかりと抑えておきましょう。

次に、無駄な運用コストを可能な限り削減するようにしましょう。無駄な運用コストに時間を割いてしまっては、日々の改善に使える時間が減ってしまいます。たとえば、特に大きな組織においては、手動でアカウント作成・削

GitHub

除といった作業に無駄な時間は割かないようにしましょう。ツール上にアカウントを作成するような認証方法だけではなく、LDAPやSAMLなどの外部の認証プラットフォームの利用がサポートされているかや、その他の管理に関しても自動化するためのAPIやコマンドが提供されているかを確認しましょう。

さらに、そのツール自体のシステム管理コストについても考えておきましょう。管理コストが多くかかってしまっては、いつまでも古いバージョンを利用したり、融通の効かないルールだらけの開発環境ができあがる原因になります。

システム管理で抑えておくことは、まずはシステムのアップグレードのしやすさです。必要なバグ修正や、新機能などをすぐに取り入れられるように、アップグレード方法を確認しておきましょう。また、開発ツールの性質によっては、ソースコードやデザインデータなど重要な資産を保存している場合もあるため、データロストが発生しないためにバックアップなどの仕組みも必要です。

また、システム障害発生時にすぐ検知できる仕組みや、復旧方法についても必ず確認しましょう。開発ツールに障害が発生したら、直接開発に影響が出てしまうため、いち早く障害の検知、そして復旧が必要です。公式サポート窓口や管理者用ドキュメントの有無についても確認しましょう。開発ツールの特有の問題を管理者自身で解決するのは困難です。自身で試行錯誤して対処すると、二次災害を招いてしまう可能性があります。

以上を踏まえて、管理者の懸念点を解消するためのチェック項目は次のようになります。

- ☐ アカウント管理のしやすさ
  - ● 認証プラットフォーム
- ☐ システムの管理のしやすさ
  - ● 容易なアップグレード
  - ● バックアップ方法
  - ● システム監視の仕組み
  - ● 復旧方法
- ☐ 運用を自動化できる仕組み
  - ● API、コマンドなど
- ☐ サポート体制
  - ● 窓口の有無
  - ● 管理者用ドキュメントの有無

以上で、すべてのチェック項目をリストアップできました。このチェックリストを使って、候補ツールの中からどれを検証フェーズへ進めるかを判断していきましょう。

作成したチェックリストの中で、コスト（お金）パフォーマンス、社内インパクトや社外インパクトについてですが、コストパフォーマンスをどれだけシビアに考えるかは、その組織がどれだけ開発者の環境を重視しているかに依存します。厳しければ厳しいほど、周りを巻き込んでいくことが大切です。

- ☐ 導入コストとその効果
  - ● 類似ツールの相場
  - ● 社内業務の効率化
- ☐ 社内インパクト
  - ● ユーザのモチベーションアップ
- ☐ 社外インパクト
  - ● 外部アピールできるか

業務効率化は自分自身で可能ですが、社内インパクトは社内の声（特にキーパーソン）が、社外インパクトは社外の声が必要です。周りの人達にヒアリングをして、フィードバックを記録しておきましょう。

次に、新しいツールへの移行についてのチェック項目ですが、これは、新しく開発ツールを導入する場合は、無視して問題ありません。

- [ ] 新しいツールへの移行が簡単か？
  - 連携ツールのサポートが充実しているか？
  - データ移行のための機能が提供されているか？

ただし、すでに利用している開発ツールから、新しいツールへの移行を検討している場合は、必須要件となります。既存ツールからの移行が難しいとなれば、ユーザへの負担がとても大きくなってしまいます。

その他の項目については、多くの開発ツールにおいて共通して確認すべきことです。これらの項目を確認して、検証フェーズに進めるかどうかを検討しましょう。公式ドキュメントなどを確認して、リストの中でなるべく多くサポートされているものを検証フェーズに進めましょう。中にはサポートされていなくて自身で対応する必要のあるものがあります。そういった場合は、初期コストがかかっても運用コストがかからないことをポイントにおいて、検討しましょう。ここで初期コストとは、スクリプト作成など一度自身で対応すれば、その後の運用は自動化できるものを指します。運用コストは、ユーザが増えるたびにアカウントを手動で追加するなど定常的に対応が必要なものを指します。

ユーザ目線と管理者目線による候補ツールの検証を実施

検証フェーズでは、「管理者目線」と「ユーザ目線」の2つの目線に分けて開発ツールの検証を行いましょう。

管理者目線の検証は、開発ツールを導入・管理する上での必要な機能の動作検証が主です。前の章で作成したチェックリストの多くが検証項目に該当します。

一方、ユーザ目線の検証については、チェックリストの項目はほとんど該当しません。これは、ユーザ向けの機能が開発ツールの特性に大きく依存するため、開発ツールごとに検証項目が変わるためです。しかし、ユーザ目線の検証で最低限確認すべきことは、すでに分かっているはずです。これまでに、「何を改善したいか」という目的を明確にし、候補となる開発ツールの選定を行い、チェックリストから検証フェーズに進めるものを決定しました。まずは、「何を改善したいか」の部分がユーザ目線の必須の検証項目となるでしょう。また必須項目以外については、その開発ツールを導入することによって得られる新しい利点を公式ドキュメントなどから確認して、検証項目にいれましょう。

管理者目線の検証と比べて、ユーザ目線の検証はとても難しくなります。公式ドキュメントに記載されている機能の検証だけでなく、その開発ツールを使って、ユーザが具体的にどのような事ができるようになり、どのようなことに困るか、しっかりとイメージしながら検証する必要があります。

可能であれば、有志を募って実際に利用する予定のユーザに検証を手伝ってもらうことも一つの手段です。有志は、新しい開発ツールに感度が高い人、アウトプットが得意な人や組織のキーパーソンが関わっているプロジェクトにお願いすると良いでしょう。管理者目線では気づきにくい質問や意見が得られ、より効果的に検証できます。ただし、基本的には「自身の検証にプラスアルファの知見」が得られる可能性があるかもしれないくらいの温度感でいましょう。ユーザはあくまで有志であり、彼ら自身のプロジェクトが別で存在します。必ずしも検証に十分な時間が割けるとは限らず、良い検証ができるとも限りません。

GitHub

それでは実際に検証する項目をおさらいします。

（管理者目線の検証項目）

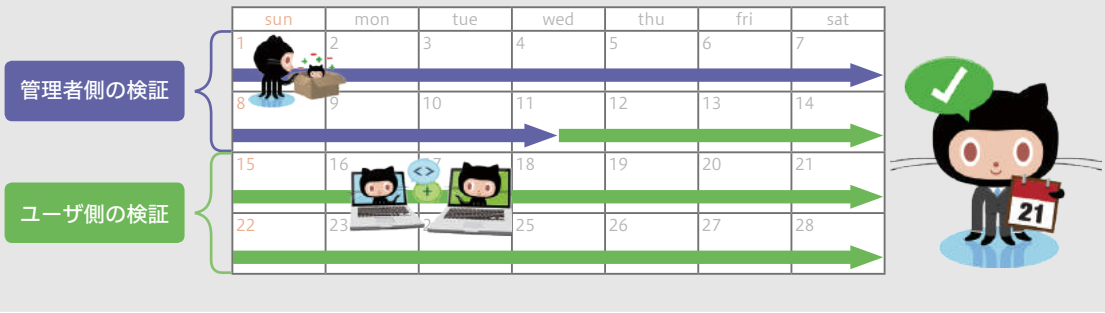
- ☐ ネットワークなどアクセス制御ができるか
- ☐ アカウント管理のしやすさ
  - ● 認証プラットフォーム
- ☐ 運用を自動化できる仕組み
  - ● API、コマンドなど
- ☐ システムの管理のしやすさ
  - ● 容易なアップグレード
  - ● バックアップ方法
  - ● システム監視の仕組み
  - ● 復旧方法
- ☐ サポート体制
  - ● 窓口の有無
  - ● 管理者用ドキュメントの有無

（ユーザ目線の検証項目）

- ☐ ユーザ用公式ドキュメントが充実しているか？
- ☐ ツールを学ぶためのワークショップなどのイベントが開催されているか？
- ☐ 新しいツールへの移行が簡単か？
  - ● 連携ツールのサポートが充実しているか？
  - ● データ移行のための機能が提供されているか？
- ☐ WindowsとMacの両方にクライアントは対応しているか（ツールの性質に寄る）？
- ☐ そのほかのユーザ機能（開発ツールごとに機能が異なるため別途項目を洗い出しましょう）

検証スケジュールをたてる

開発ツールを検証する際、トライアル版を利用するケースが多く、検証できる期間が限られています。そのため、事前にリストアップした検証項目から、大まかなスケジュールを作成し、検証を開始しましょう。管理者目線の検証は、すでに説明したように検証項目が明確であり、スケジュールの見積もりが容易です。先に管理者目線の検証を実施しましょう。一方で、ユーザ目線の検証は項目が多く、また検証中に新しい検証項目が発覚することも多々あります。有志で参加する開発者から質問が多く寄せられるなど、予測不能な要素が多く、想定以上に時間がかかる可能性が高いです。ユーザ目線の検証は余裕をもって行えるようにスケジュールをたてましょう。



例えば、検証期間が1ヶ月とすると、最初の1週間半で検証項目が明確な管理者目線の検証を終わらせ、残った時間でユーザ目線の検証を行いましょう。

検証作業中にやるべき3つのこと

開発ツールの検証では、次の3つのことを行うように心がけて下さい。

1 検証ログをしっかり取ること

検証中に、画面のスクリーンショットや、実行したコマンドやその実行結果などを、可能な限り全てドキュメントとして残しておきましょう。検証中にコマンドや検証フローを再度確認できるほか、ユーザ機能の検証に関しては、後に社内用のドキュメントとして転用することもできます。手間や時間がかかっても、検証ログを残しておくことが、最終的に検証や導入を迅速かつスムーズに進めるための強力な武器となります。

2 分からないことは積極的にサポートに質問すること

多くの開発ツールはサポート窓口を提供しています。検証の段階で沢山問い合わせをして、必要な情報を得るとともにサポートの品質を確認しましょう。1日以内に返答があるか、1週間経っても返答がないのか、といったレスポンスの速度や、返答内容の的確さなど、導入後にサポート窓口が満足に機能するかを検証しましょう。

3 検証用環境と本番環境を別々に用意すること

検証を完了し正式に導入を決めた場合、検証環境をそのまま本番環境として利用するケースがありますが、可能であれば検証環境はそのまま残しておいて、新しく本番環境を構築しましょう。ユーザに影響のある動作検証や、後のアップグレードのテストをする際に、すぐに検証環境を使えるようにしておくことで、スムーズに動作検証ができます。また、予算とインフラの許す限り、検証環境は本番環境と同様のネットワーク構成やシステムスペックにすることを推奨します。たとえば、CI/CDツールなど外部のツールと連携する際、ネットワーク構成に起因した問題が発生することがあります。その際に、検証環境が本番環境と同じネットワーク構成になっていると、原因を見つけやすくなります。またデータに関しても、本番環境と同等のデータを入れておくと、アップグレードにかかる時間をより実際の時間に近く見積もることができます。

GitHub

第 3 章

GitHub Enterprise

GitHub Enterprise は世界中で広く使われている GitHub.com と同等の機能を利用できるオンプレミス版の開発プラットフォームです。この章では、GitHub Enterprise が、検証項目をどれくらい満たしているかについて表で紹介します。

		管理者目線の検証項目
ネットワークなどのアクセス制御ができるか	●	社内ネットワークに GitHub Enterprise 用のインスタンスを配置できるため、アクセス制御が容易です。
アカウント管理のしやすさ	●	**認証プラットフォーム** Built-in、CAS、LDAP、SAML（AD FS、Azure AD、Okta、OneLogin、PingOne、Shibboleth）といった多くの認証方式をサポートしており、組織のポリシーに合った認証方法を選択できます。
運用を自動化できる仕組み	●	**API、コマンドなど** ユーザ向けの API だけでなく、GitHub Enterprise を管理するための API やコマンドを多くサポートしています。
システムの管理のしやすさ	●	**容易なアップグレード** パッケージダウンロード、コマンド実行などの簡単アップグレードに対応するほか、バグ修正などのパッチリリースは基本的にダウンタイムゼロで適用可能です。
	●	**システム監視の仕組み** SNMP サポート、システム状況のグラフ、ログ転送、監査ログに対応しています。
	●	**復旧方法** バックアップデータから簡単に復旧する仕組みや、ネットワークやハードウェア障害が発生した際の高可用性の仕組みを提供しています。
サポート体制	●	**窓口の有無** 英語でのテクニカルサポートを、平日 24 時間受け付けています。また平日午前 9 時から午後 5 時までの間は、日本語でもサポートを行っています。さらに、システムダウンなど緊急の問題が発生した場合は、英語限定ですが、24 時間 365 日サポートします。
	▲	**管理者用ドキュメントの有無** 公式ドキュメントは、現在日本語化を進めていますが、英語で記載されたものがほとんどです。

GitHub

ユーザ目線の検証項目		
ユーザ用公式ドキュメントが充実しているか？	▲	公式ドキュメントは、英語で記載されたものがほとんどです。また、公式ではありませんが、インターネット上で数多くの日本語の記事が公開されています。
ツールを学ぶためのワークショップなどのイベントが開催されているか？	●	ギットハブ・ジャパン合同会社主催のセミナーを継続的に開催しています。
新しいツールへの移行が簡単か？	●	**連携ツールのサポートが充実しているか？** 様々なサードパーティツールと連携可能です。
	●	**データ移行のための機能が提供されているか？** Subversionやその他類似ツールから、リポジトリやユーザ情報といったデータを移行するための手段を多くサポートしています。例えば、移行APIや専用の移行ツールを提供しています。
WindowsとMacの両方にクライアントは対応しているか（ツールの性質に寄る）？	●	WindowsとMacに対応したGitHub Desktopなどのクライアントを用いて、リポジトリのクローンやブランチの作成、履歴の参照、コードのコミットなどを快適に行えます。

GitHub

まとめ

ソフトウェア化するビジネス環境において、開発環境の整備・改善は競争力のある製品やサービスを開発するだけでなく、それを開発する優秀なエンジニアの採用活動を優位に進めるためにも重要です。開発環境の整備・改善において最も大切なことは、新しい開発ツールを導入することではなく、組織の要望にあったツールを選定して、それを組織に浸透させることです。開発ツールの管理者はただのシステム管理者ではありません。より良いサービスを世の中に生み出すための縁の下の力持ちであり、常に開発者を支える存在なのです。

本稿で述べたチェックリストを参考にすることで、ユーザ・マネージャ・管理者の３者すべてがメリットを実感する提案ができるよう願っています。

ギットハブ・ジャパンについて

米国カリフォルニア州サンフランシスコ市に本拠を置くGitHubは、2008年に設立され、Andreessen HorowitzやSequoia Capitalなどの主要投資家から総額3億5000万ドルを調達している、最も利用されているソフトウェア開発プラットフォームのひとつです。GitHubが運営するGitHub.comは、世界最大規模のオープンソースコミュニティであり、大きな影響力を持つテクノロジーが数多く開発・管理されています。ギットハブ・ジャパンは、2015年6月にGitHub, Inc.の日本支社として設立されました。日本におけるGitHubの普及と情報提供に努めるとともに、ソフトウェア開発にGitHubを導入する企業に向けて強力なサポートを展開しています。

GitHub

ギットハブ・ジャパン合同会社

〒105-0012 東京都港区芝大門 1-10-18 PMO 芝大門 7F

https://github.co.jp/