模擬試験

2018 年 11 月版

目次

はじめに

これは EXIN DevOps Master™ (DEVOPSM.JP) です。この試験は EXIN 試験の規則および規定を適用します。

この試験では 50 問が多肢選択形式で出題されます。各問題には、選択肢が複数ありますが、そのうち正解は 1 つのみです。

この試験の最高点は 50 点です。正解 1 つにつき 1 点とします。33 点以上獲得すると合格となります。

試験時間は 120 分です。

ご健闘を祈ります。

模擬試験

1 / 50
DevOpsを組織に導入する理由として、妥当なものはどれでしょうか？

A) DevOpsでは開発スピードが速まるため、新規サービスのフィードバック・サイクルがより頻繁になる。
B) DevOpsではプロセスが最適化されて価値を高める活動のみが実施されるため、事業継続性とアジリティが向上する。
C) DevOpsではソフトウエアがより頻繁にリリースされるため、新しいサービスをジャストインタイムで提供できる。
D) DevOpsでは開発と運用が一体化されているため、両者の協業におけるムダが減る。

2 / 50
CTOは、DevOpsを導入する際にある種のリーンの概念を適用すると最も効果的であると考えています。

次のリーンの概念または手法うち、DevOpsを導入する時に **最も**効果的であるのはどれでしょうか？

A) **カイゼンと５S。**アジャイルやDevOpsはリーンの主要な概念に基づいている。そしてカイゼンと５Sはリーンの基本である。それらがDevOps導入に際して最も効果的である。
B) **先行カイゼン 。**DevOpsでは、運用から開発へのフィードバックが要求される。先行カイゼンは、上流へのフィードバック・ループを作る。DevOpsにこの原則を適用するとフィードバック・ループを支援する。
C) **大部屋システム 。**DevOpsは、異なった管理形式のプロセスを統合する。大部屋システムは、全体プロセスの見える化を支援し、DevOps導入を成功させる。
D) **一個流しと自工程完結。**DevOpsは上流工程の構築と単一のバリューストリームフローから効果が出る。一個流しはこれを可能にし、自工程完結は効率化とそのフローを作るのに役立つ。

3 / 50
従来のプロジェクトと比較して、DevOpsでプロジェクトを成功させるためには何を変更する必要がありますか？

A) プル型システムと一個流しを使用する、ITサービスのサプライチェーンを構築する必要がある。
B) サービスの素早い保守に向けて、開発者が運用チームに加わる必要がある。
C) 運用は開発チームのために働く必要がある。だからこそ、DevOpsと呼ばれている。
D) 運用チームのメンバーは、開発チームに参加する必要がある。

4 / 50
DevOpsを採用しているチームはすべて、共同体と呼ぶことができます。

共同体を機能させるうえで、**最もよく**当てはまる原則はどれでしょうか？

A) 動的な協業
B) 継続的コミュニケーション
C) サイクルの短縮
D) 説明責任の共有

5 / 50
DevOpsの導入のために利用できる、数多くの知識、標準、および手法が存在します。

DevOps導入の成功のために**最も重要**と見なされるものはどれでしょうか？

A) CMMI Level 3
B) 規律あるアジャイル
C) ISO 20001
D) PMI PMBok

6 / 50
協業は、効果的なDevOpsの4本柱の1つです。

協業はなぜそれほど重要なのでしょうか？

A) 協業はDevOpsにおいて一個流しを実現する唯一の方法であるから。
B) DevOpsのビジネス価値は、チーム間の協業を通じて実現するから。
C) DevOps組織は小規模であり、チームは効果的に協業する必要があるから。
D) 協業は変更を促進し、容易に成功をもたらすから。また、開発コストを削減できるから。

7 / 50
あなたは小規模なDevOps企業のオーナです。同僚は5名で、障害のある子ども向けのモバイル・アプリを構築しています。チームが最も誇りを持っているアプリの1つは、自閉症の子ども向けのアプリで、これは自分でスケジュールを組めるようにするものです。

このアプリが大成功を収めたため、チームはスケジュールを自分で組むことでメリットを得られる他の人々のために、このアプリの機能を拡張してほしいと頼まれています。このリクエストによってコードが複雑化し、技術的な課題が生じることは確実で、チームで取り組む必要があります。

この仕事は非常に報酬が高そうであったため、あなたは仕事を引き受けました。しかし数週間後、チームは言い争いをしていました。あなたも腹を立て、チーム・メンバー全員の業務内容に注意を払うようになりました。あなたは常にメンバーとともに働き、コード・エラーを見つけて素早く修正できるようにしました。

十分な作業が完了したにも関わらず、チームはまだ腹を立てており、あなたはDevOpsの考え方が部分的に欠落していることに気づきました。

この問題を解決するための**最適な**戦略はどれでしょうか？

A) 技術分野以外の補完的なスタッフを採用し、チームにメンバーを追加して多様性を高める
B) 社外のDevOpsエキスパートに、チーム・メンバーの支援と指導を依頼する
C) 共通の目的を見つけ、その目標に向かって共に努力し、協業を強化する
D) 数か月かけてまずチーム・ビルディングに着手し、互いについて学ぶ

8 / 50
あなたは、自分の開発チームが本当に良いチームだと感じています。

彼らが単にグループ（人の集まり）では無くて、チームになっていると認める徴候は何ですか？

A) チーム会議で合意したルールに従っている。
B) 自分たち自らが主導する効果的な会議を行っている。
C) チームは、共通の目的に向かって、安定した作業ペースを保っている。
D) チームは、責任あるチームメンバーに問い合わせることで問題を解決している。

9 / 50
AppAtoZ社は、iPhoneやアンドロイドプラットフォーム向けのモバイル・アプリケーションの開発とデプロイにおいて驚異的な速さで成長し続けています。

この新興企業の開発チームは、現行のモバイル・アプリケーションの機能拡張を強気なスケジュールで迅速にデプロイするために大変なプレッシャーを経験しています。チームメンバーは過去６ヶ月間にわたり、平均して週60時間働き続けています。上層部は人員を増やすことに消極的で、それよりも運用と開発のコストを削減して収益を増やすことに関心を持っています。

ここ数ヶ月で、社員の欠勤や病欠が増加し、退職する社員もいたことから、現在働いている社員の仕事量が増加しました。再雇用や新入社員の育成を行っても、開発チームの仕事のプレッシャーはすぐに軽減できません。

社員の疲弊やストレスに対処するには、どの**長期**戦略を考慮する必要がありますか？

A)
- 作業負荷をより良く管理するためにチームに正社員と契約社員の両方を雇う。
- 上層部と開発部門が疲弊の一因となる作業環境の全ての要因を列挙する。
- それらの課題に対処する計画を立てる。

B)
- 能力がないため上層部を解雇。
- 新たに、持続可能で現実的なワークライフバランスや文化をどの様に作るか、ノウハウを持ったより能力のある上層部チームを採用。
- 開発チームに自身のワークライフバランスを考える時間を与える。

C)
- 開発チームに休息を取らせる。
- 年間の作業ピーク時に増加する開発作業負荷を管理するための契約社員。
- 開発者が必要な時に相談できるメンタルヘルスの専門家を用意する。

D)
- 開発チームに休息を取らせる。
- 上層部と開発部門が疲弊の一因となる作業環境の全ての要因を列挙する。
- それらの課題に対処する計画を立てる。

10 / 50
あなたは開発者としてDevOpsチームで働いており、チーム・メンバー全員による協業を促進しようと試みています。チームには、先輩にあたる男性の開発者が2名、後輩にあたる女性のシステム管理者が1名います。

チーム・メンバーは前途多難なスタートを切っており、うまくやっていくことができませんでした。あなたはその時点で仲裁に入り、仲良くやっていけるよう手助けを試みました。この仲裁が功を奏し、彼らは口論をやめて、多くの仕事を成し遂げました。

あなたは今、後輩の管理者が先輩の開発者らにひたすら同意する傾向があることに気づきました。

この傾向には、どのように対応したらよいでしょうか？

A) チーム・メンバーが口論をしていない限り、そのまま放置しておく。彼らは何とかうまくやっていくであろうし、あなたが邪魔をするべきではない。
B) 運用がビルドについての懸念を表明することは重要であるため、管理者に言いなりになることをやめさせ、自分の意見を明確に示すよう指導する。
C) 先輩の開発者らのほうが責任が大きいため、彼らにもっと思いやりを持つよう促し、後輩が気づかない場合は運用の懸念について考えるよう依頼する。
D) あなた自身は開発者であるが、当面の間、運用チームに加わり、後輩の管理者に手本を示す。

11 / 50
アジャイルとスクラムによって、ソフトウエア開発のスピードと予測可能性が高まる理由はどれでしょうか？

A) 設計前に、より効果的で完璧な要件収集／処理が可能になるから
B) 小規模かつ自律的な自主編成／計画チームが可能になるから
C) プロダクトオーナが日常的なスタンドアップ・ミーティングに参加できるから
D) プロジェクト管理者が必要に応じて素早く優先順位を変更できるから

12 / 50
軽量化したITSMとは何でしょうか？

A) 事業継続性に焦点を絞ったITSM
B) 標準として提案された新しいITILバージョン
C) ITILプロセスの不完全な導入
D) リリース管理指向のITSM

13 / 50
あなたはリーンやアジャイル手法を導入している企業で働いています。CEOは、DevOpsという新しい手法をさらに導入する価値について、納得していません。

あなたはDevOpsのエキスパートです。あなたは、ソフトウエアを開発している企業なら例外なく、DevOpsからメリットを得られると考えています。

DevOpsによって、あなたが働いている企業に追加されるものは何でしょうか？

A) より優れたユーザー・ストーリーと機能要件を作成できる
B) 顧客が定期的なアップデートで苦労しないよう、リリース頻度を削減できる
C) 開発と運用の双方を担えるようにプロフェッショナルをトレーニングできる
D) ビジネス成果をサポートするコードを作成し、より迅速にリリースできる

14 / 50
上級経営陣は、ビジネス目標に対するIT部門からのビジネス支援と協力を強化するよう求めています。さまざまな対策が考えられる中、あなたはCIOとして、運用における管理負荷を軽減することに決めました。

既存のサービスレベルマネジメント（SLM）を改善するために、DevOpsはどのような方法で**最も**役立ちますか？

A) ITSMは重厚長大なアプローチであるため、ITSMのベストプラクティスを廃止
B) 開発と運用の間で、より優れたオペレーショナルレベルアグリーメント（OLA）を設計
C) ITILバージョン3に基づいて新しいITSMプロセスを導入
D) ITSMを再調整して必要最低限の情報（MRI）を使用

15 / 50
業務系システム（SoR）アプローチを採用している企業に**最も**適している、DevOpsの導入はどれですか？

A) 協業
B) 継続的デリバリー
C) トヨタ方式

16 / 50
大部屋システムを使用する**主な**メリットは何でしょうか？

A) 顧客に不満を表明するよう促し、チームが継続的な改善に向けて十分なフィードバックを得られるよう保証する
B) チーム内のストレスに対処し、チーム・メンバーが持続可能なペースを維持できるようにする
C) 日常的なバグのレポートを改善して、やり直しを少なくし、バグが他のワークステーションに広がることを減らす
D) 迅速な情報収集／共有により、現状に基づいて素早く意思決定を下す

17 / 50
あなたは、DevOps を採用し始めたばかりのソフトウエア企業で働いており、組織文化だけではなく手法とツールも変える必要があると気付いています。あなたの会社ではすでに、問題がないか確認するためソフトウエア・プロジェクトを監視しています。

あなたは、次のような対策を追加するよう提案しています：
- 変更管理プロセスを自動化
- 誰かが承認を得ずに変更を行うことを防ぐため、アクセス・コントロールを導入

これらはなぜ必要なのでしょうか？

A) 自動化は、信頼性を維持しつつ、より迅速に変更を実施できるようにする。アクセス・コントロールは、問題解決の経験則を防ぎ、計画外のサービス停止を防ぐために必要である。
B) 自動化は、変更の数を抑えるために必要である。アクセス・コントロールは、こちらが気づかないうちに勝手に、顧客が自らソフトウエアを変更することを防ぐ。
C) 自動化は、運用を不要にするために役立つ。アクセス・コントロールは、DevOps プロジェクトがクラウド環境で行われており、高いリスクを伴うため、導入する必要がある。

18 / 50
仮想化とクラウド・コンピューティングは、DevOps手法を支え、促進できる技術です。

これらはどのようにDevOpsに役立ちますか？

A) クラウド・コンピューティングはリモート・アクセスを可能にし、顧客による管理を向上させる。
B) 仮想環境は標準化が容易であり、ハードウエアの使用効率が向上する。
C) 仮想インフラストラクチャは分かりやすく、保守が不要である。

19 / 50
サービスレベルアグリーメント（SLA）は、顧客との合意内容を具体的に定めるもので、あらゆるプロジェクトにおいて重要です。しかしDevOpsでは、SLAは他の重要な目的にも役立ちます。

この目的とは何でしょうか？

A) 顧客はDevOpsチームのためにSLAを作成する責任を負う。したがって、運用の責任としての正式なタスクに取って代わる。
B) SLAでは顧客は、すべての非機能要件を指定できるため、開発はこれらに焦点を絞ることができる。
C) SLAは受け入れ可能なサービスレベルを指定する。開発はSLAを理解し、運用がそれを維持できるよう支援する必要がある。

20 / 50
新しいプロダクトのためにあなたのチームはデプロイメント・パイプラインを作る必要があります。継続的インテグレーションの一部として、あなたはパイプラインの中でコミット・ステージを定義する必要があります。あなたはあなたのチームの仲間とこのステージについて議論します。

プロセスマスタ（管理者）が言いました。「完了の定義（DoD）は、コミットステージの前または最中に定義すべきだ。コミットした時にコードが完了していない場合、作業を停止すべきである。」

これは正しいでしょうか？

A) はい、もし仕事が完了していなければ、プロセスマスタは職務を果たしていません。この問題は早急に解決すべきです。
B) はい、顧客に付加価値を与えていないため、完了していない仕事は、コミットすべきではありません。
C) いいえ、完了の定義とは、顧客との会議でのみ定義できます。完了の定義を待つことはあまりにも仕事を遅らせます。
D) いいえ、デプロイメント・パイプラインでの仕事は、常に継続されるべきものです。もしコードが完了していなければ、それはただ停止する必要があります。

21 / 50
より幅広い経歴や文化を持つチームの多様性を高めることの**最大の**メリットは何でしょうか？

A) 多くの経験や視点をもたらす。
B) チーム内での摩擦を削減する。
C) 独創性と新たな洞察を生む能力を制限する。
D) 決定に至るまでにかかる時間が長くなる。

22 / 50
あなたはDevOpsチームを1つ擁する小規模企業で働いています。DevOpsチームは、複数のコンポーネントで構成されるアプリケーションに取り組んでいます。新しいコンポーネントもあれば、更新が必要なコンポーネントもあります。

現時点では、各コンポーネントには独自のデプロイメント・パイプラインがあります。チームは、大半のデプロイメント・パイプラインで行っている継続的デリバリーに誇りを持っています。チームの生産量は大きく、高品質です。

どのような対応が必要でしょうか？

A) さまざまなパイプラインを維持し、チームに継続的デリバリーを拡張するよう促す
B) 継続的デリバリーを行っているパイプラインのみを維持し、他の継続的デリバリーを行っていないパイプラインを融合させる
C) 開発用と保守用に1つずつ、2種類のパイプラインのみを維持する
D) 複数のパイプラインを有するリスクを説明し、チームとともに単一のパイプラインに向けて取り組む

23 / 50
あなたは、２年前からDevOpsに取り組んでいる、中規模から大規模な組織であるCompany Builders社を査定しています。

同社は現在の成熟レベルを確認するためにあなたを雇いました。査定が完了し、あなたは改善のために提案をしなければなりません。同社は、次の成熟度レベルの *レベル２-定量的管理が行われている*に達するために、どの分野を重視すべきか知りたがっています。

あなたは、ほとんどの分野は *レベル１ ー一貫性がある*に達していることを知りました。ただし、次の２つの分野は例外です。

A. **環境とデプロイメント。**この分野は調整されたデプロイメントを管理し、リリースとロールバックプロセスをテストしました。
B. **ビルド管理と継続的インテグレーション。**この分野では、常に自動化されたビルドとテストを行い、どのビルドも自動化されたプロセスを使ってソース管理から再作成できます。

以上の情報に基づいて２つの分野の成熟度を判定してください。さらに、改善の重視点を推奨してください。

Company Builders 社がレベル２に進むには、２つの分野のうちどちらに重点を置くべきでしょうか？

A) 環境、デプロイメントとビルド管理、継続的インテグレーションはどちらもレベル０．仕事はどちらも同時に完了しなければならない。
B) 継続的インテグレーションはどちらもレベル１以上。仕事を進めるために他の分野の作業を完了させなければならない。
C) 環境、デプロイメントはレベル０。　ビルド管理、継続的インテグレーションはレベル１。環境、デプロイメントに最初に注目すべき。
D) 環境、デプロイメントはレベル２。ビルド管理、継続的インテグレーションはレベル０。ビルド管理、継続的インテグレーションのみに注目すべき。

24 / 50
あなたの会社は、顧客にオンライン・バックアップ・サービスを販売しています。現在、ある顧客がサービスに新機能を組み込むよう求めてきました。1週間以内に新機能を提供できなければ、取引先を変えられてしまう可能性があります。

あなたは、この新機能が重要であると考えており、開発チームがそれを素早く構築できることを知っています。しかしあなたは、次のような問題に直面しています：

- テスト担当がバグをクローズするのに長時間かかる。
- 開発者がかなり前に修正したバグを、テスト担当が見つけている。
- アプリケーションの動作をめったにデモンストレーションできない。
- 展示の機会がほとんどない

あなたの問題は何でしょうか？どのように解決すべきでしょうか？

A) バグの多いコードを導入している。デプロイメント・プロセスの理解を深め、協業を強化し、より規律ある方法で働くことで、この問題を解決すべきである。
B) 構成管理が不十分である。開発と運用の協業を強化し、監視および記録を拡大し、仮想化を導入することで、この問題を解決すべきである。
C) 継続的インテグレーション・プロセスが適切に管理されていない。自動テストとコミット・ステージを加速させ、継続的インテグレーション・プロセスの理解を深めることで、この問題を解決すべきである。
D) テスト戦略が効果的ではない。テストを自動化してテスト担当と他のチーム・メンバーの協業を強化することで、この問題を解決すべきである。

25 / 50
効果的なDevOpsにおける継続的インテグレーションのメリットはどれでしょうか？

A) 機能リリース前のテスト・サイクルが長い
B) 次の機能リリースまでの期間が長い
C) 機能リリースがより頻繁でタイムリーである

26 / 50
多国籍の組織では、ダラスにある中央リポジトリーに分散された場所からのコミットをマージするにあたりに多くの難問を抱えています。分散拠点とはメキシコシティ、パリ、サン・ディエゴや英国などにあります。各地域がいつコミットを実行するかには一貫性がないため、いくつかのテストが失敗したかどうかがわからないことがあります。

ここに4つの可能性のある手法があります。

1. 壊れたビルドにチェックインしない。
2. コミット前に全てのコミットテストを常に各地域で実行する。または、自分の継続的インテグレーション・サーバにコミットテストを実施させる。
3. コミットテストが通ってから次に進む。
4. 失敗したテストはコメントアウトしない。

各拠点がそれぞれ現在抱えている問題点に対処できるようにするには、これらのうちどれが **最も** 適用可能でしょうか？

A) １と２
B) １と２と３
C) ２と３
D) ２と３と４

27 / 50
基本的なデプロイメント・パイプラインの構造を考えてください。

どのステージが機能的および非機能的レベルでシステムが正常に動くと断言できるでしょうか？

A) 自動化された受け入れテスト
B) ビルドと単体テスト
C) 手動の受け入れテスト
D) バージョン管理

28 / 50
アプリケーションが稼働するそれぞれの環境にデプロイするまで同じプロセスを使用する事がDevOpsのベスト・プラクティスです。これによりビルドが効果的にテストされることを確実にします。あなたはビルドとデプロイメントプロセスでスクリプトを使用しています。

これを行うのにどの方法が**最善**でしょうか？

A) それぞれの環境毎に一つのスクリプトを使用する。そしてバージョン管理システムでそれらを保守する。
B) 環境間の差異を明確にしたそれぞれの環境に特別な一つのスクリプトを使用する。
C) 特定の構成にはマニュアル・パラメーターを使用してそれぞれの環境で同じスクリプトを使用する。
D) それぞれの環境にデプロイのための同じスクリプトを使用し、別々に構成情報を管理する。

29 / 50
新規ITサービスのリリース後に、運用のある業務が不意に終了してしまいました。

この原因として妥当では**ない**ものはどれでしょうか？

A) ゲートキーパーが作業項目とSACの相互関係を考慮に入れなかった。
B) ユーザー・ストーリーが不完全であったため、非機能要件が明確ではなかった。
C) サービスマスターが、リリースされるサービスのエンド・オブ・ライフ（EOL）についてユーザーと合意していなかった。
D) 開発プロセスにおいて、サービス受け入れ基準（SAC）でサポートされている通りに作業が行われなかった。

30 / 50
AppBC企業ではDevOpsを使用しています。この企業は高度に自動化された受け入れテストで、継続的デプロイメントと信頼できるデプロイメント・パイプラインを導入しています。そして毎日新しいソフトウエアのデリバリーを稼働まで行っています。

AppBC社は大規模なデータベースと多数のユーザーを抱えています。彼らは、広範で信頼できるテスト戦略を導入しています。彼らの環境が極めて大きくまた複雑であるため、それぞれの新しいバージョンでバグが稼働中に発生します。

これらのバグを予防する**最適**な戦略はどれでしょうか？

A) カナリア・リリースを採用
B) 自動化されたキャパシティーテスト
C) デリバリー回数を減らす
D) ブルー・グリーン・デプロイメントを導入

31 / 50
DevOpsは、トヨタ生産方式に由来するアジャイルから非常に重要な概念を取り入れています。

DevOpsの採用に、一個流しが重要である理由は何でしょうか？

A) チームが予測可能なベロシティで、持続可能なペースで働けるようにする。
B) 最大のビジネス価値をもたらし得る機能を追加する作業に、チームが注力できるようになる。
C) さまざまなチーム・メンバー間での作業の共有責任が増える。
D) 同時に行うタスクの数を制限することで、ボトルネックが減る。

32 / 50
中規模の自動車部品サプライヤのS社は大手のT自動車に部品を供給しています。S社がT自動車へ供給している部品は、S社の総売上のほぼ60%を占めます。

新たなパートナーシップを議論するための役員会が開催されました。T自動車は、S社にジャストインタイムでの供給方法に変更を要求しており、変更できないならばS社とのビジネスを停止しようとしています。S社はこのビジネスを失えば生き残れないため、ジャストインタイムに変更しなければならない危機感を持っています。この変更は６ヶ月以内に実施しなければならないため、最大でも5ヶ月しか準備期間はありません。

このために導入すべき対策の一つは、無線自動識別装置（RFID）を使用した部品追跡です。部品追跡は生産プロセスの透明性を保つのに役立つはずです。現在のプロセスをRFIDを用いたプロセスに変更することを進めるため、現在のプロセスを急いで見直す必要があります。

CIOは、変更プロセスを管理するように依頼されました。CIOは、DevOpsの手法で最小限のリリースを作成すればプロセス変更が可能であると信じています。理想としては、最初にRFIDを使用した生産方法の概念を作成しなければなりません。そして最後の段階で、RFIDデータを用いた生産管理システムを実装するべきです。しかし、これらのステップを順番通りに実行する充分な時間がありません。そのため、これらの作業を同時に行う必要があります。

CIOはこのプロジェクトのスクラムマスターであるEmさんを任命しました。開発チームはデプロイメント・パイプラインを構築する準備をします。

Emさんは、開発チームは熱意があり、懸命に働くとみていますが、もっと規律が必要だと考えています。さらに、リリースの頻度を上げる必要があります。

Emさんは **最初**に何に注目すべきでしょうか？

A) Emさんは、DevOpsでは最も大事な事柄である、コミュニケーションに注目すべきです。Emさんはチーム内のアイスブレークから始め、コミュニケーションのためのルールを設定すべきです。
B) 整流化されたプロセスと流れは大変重要ですので、Emさんはバリュー・ストリーム・マップと一個流しの仕組みをチームと議論することから始めるべきです。
C) DevOpsではツールや手法が正しく使用されて最も効果を発揮するので、Emさんはインフラストラクチャとチームメンバーの労働環境をチームと議論することから始めるべきです。
D) 組織文化の変更がDevOpsには必要ですから、Emさんは、関係者を集めて彼らにDevOpsを教育し、素晴らしい組織文化に変える支援を頼むことから始めるべきです。

33 / 50
あなたのDevOpsチームは、持続可能なペースでうまく協力し合っています。チームはプロセスに十分な緩みを持たせることで、ビルドの入念なチェック／テストのための時間と集中力を確保しています。現在チームは、手動でテストと導入を行っています。ペースは速く、定期的にビジネスに高い価値をもたらしています。

CEOは、チーム内の自動化に関してあなたのアドバイスを求めてきました。

あなたが提供すべきアドバイスはどれですか？

A) チームがより多くの機能を追加し、早期にビジネス価値を実証できるよう、可能な限り自動化する
B) 手動プロセスのほうが安全であるため、受け入れテストは自動化するが、デプロイメントは自動化しない
C) サイクルタイムを改善するためデプロイメントを自動化するが、テストは自動化せず、バグから学べるようにする
D) チームが現在実行している方式は、素晴らしい成果を上げているため、このチームの方式に自動化は追加しない

34 / 50
CIOがスクラムマスターで最も信頼できる社員のマイケルにプロジェクトを任命しました。開発チームはデプロイメント・パイプラインの構築を準備しています。

マイケルは、開発チームの優れた自発性や心構えに自信を持っています。しかし彼らにもっと自律してほしいと望んでいます。加えて、リリース頻度をもっと高めるべきだと望んでいます。マイケルは開発チームにもっとリリース頻度を上げることを望みます。

あるチーム・メンバーが言いました。「この新しいデプロイメント・パイプラインが自動的に働くという事について、最も大事なことはまず自動化されたデプロイメント・パイプラインにすべきだ。」

この発言は正しいでしょうか？

A) はい、正しいです。自動で働くデプロイメント・パイプラインは、効率を上げるために最も重要な要素です。
B) はい、正しいです。自動化されているデプロイメント・パイプラインを作る際に注力することで、後で直面する潜在的な問題を克服します。
C) いいえ、正しくありません。一個流しを実施することと確かなデプロイメント・プロセスが最優先されるべきです。プロセスの自動化は後でできます。
D) いいえ、正しくありません。デプロイメント・パイプラインの自動化の代わりに、テストプロセスを最初に自動化すべきです。

35 / 50
あなたの会社は、DevOpsでの仕事に着手するため、方法を変更中です。あなたのチームはこの変更に加わっています。あなたはコードのコミットステージでのベスト・プラクティスを議論しています。

あなたの同僚であるサンが言いました。「ビルドが壊れてだれも責任を取らない時、我々は誰がやったのかを見つけ出すべきであり、ビルドを修復できるようみんなを招集すべきだ。」

この考えは良いでしょうか？

A) はい、ビルドを壊した人のみが修復できます。たとえみんなの気分を害することでも、あなたは誰が原因かを特定すべきです。
B) はい、あなたは常に、ビルドを壊した責任を取るべきです。あなたの同僚はたぶんこのルールを強要するでしょう。
C) いいえ、DevOpsでは、責任を問わない環境です。もし同僚が責任を取らなくても、彼らに強制しません。
D) いいえ、あなたはビルドの修復をまずやるべきです。そのうえで責任を取るべき人物を特定するために時間をかけ、その人を罰します。

36 / 50
XAppGo社の開発チームは、現在のテスト作業で多くの課題に直面しています。現在彼らは、手動での受け入れテストプロセスを使用しています。開発チームは、彼らが作成した単体テストセットがレグレッションを防ぐのに十分であると信じています。

開発チームはリリースごとの手動での受け入れテストに100万ドルを費やしています。上層部は開発チームに、自動化された受け入れテストを導入して、テストの全体費用を削減し、また、稼働環境に与えるレグレッションやコードの欠陥の件数を最小化することを指示しました。

自動化を考慮してアプリケーションの受け入れ基準を定義するときに、従わなければならない原則はどれでしょうか？

A) アジャイルの原則
B) ATAMの原則
C) INVESTの原則

37 / 50
自動でデータを移行するための**最も**効果的な手段はどれでしょうか？

A) データベースのバージョニング・スキーマを作成し、バージョン管理下に置く
B) 移行が容易になるよう、小規模なデータセットを作成して管理する
C) データの移行前に、すべてのスクリプトが適切にテストされているか確認する
D) 移行の失敗に備えて、ロールバック手順を準備しておく

38 / 50
X-AppGo社は、ロールバック・プロセスに問題を抱えています。このため、ロールバック・スクリプトの実行時に、本番アプリケーション・データベースにおいて重要なデータの損失が頻繁に発生しています。

重要なデータを失わずにロールバック・スクリプトを実行することが**できない**のは、どのような場合ですか？

A) ロールバック・スクリプトが、新しいバージョンで使用されているデータのみを削除する。
B) ロールバック・スクリプトに、テーブル間のカラム移動が含まれている。
C) ロールバック・スクリプトが、一時テーブルからのデータを追加する。

39 / 50
ACMECONST社は、ルータとスイッチにアプリケーション・ソフトウエア・アップグレードおよびハードウエア更新を実施した後、多くのアプリケーション／ハードウエア障害に遭遇しています。

こうした障害はメンテナンス期間中に発生したため、元の状態への復旧は極めて難しくなっています。このため、リカバリ時間が延びて通常のメンテナンス期間を超過し、重要なアプリケーションのダウンタイムが拡大しています。

自動プロビジョニングと自律的インフラストラクチャはこの状況で役立ちますが、いくつかの検討事項が存在します。

本番環境への導入の際、停止のリスクを減らすために、入念な管理が必要なアイテムはどれでしょうか？

A) アプリケーション・アップグレードの不具合の問題解決に向けた詳細な監視ログ
B) 外部システムやサービスなどの、外部の統合ポイント
C) サーバー構成と、基盤となるユーザー・アカウント情報
D) 自動プロビジョニング・ツール一式と自律的アーキテクチャ

40 / 50
X-AppGo社は、コア・アプリケーションに問題を抱えています。このアプリケーションは、他の外部アプリケーションと適切に連携できません。こうした外部アプリケーションは、特定のコールを実行できるよう、個々のデータ変数を効果的に取得する必要があります。コア・アプリケーションはあるチームによって開発中であり、同社は相応なビジネス上の理由により、このアプリケーションを維持したいと考えています。

開発者の1人が、インターフェイスの問題に対応するため、X-AppGoのコードベースからあるコンポーネントを分離することを提案してきました。

このケースでコンポーネントを分離する理由として、妥当なものはどれでしょうか？

A) コードベース内のプラグイン一式をモノリシックなコードベースに変換する
B) 変更の影響を制限し、コードベースの変更を容易にする
C) X-AppGoコードベースは、別のチームによって分離／管理される必要がある
D) 妥当な理由はなく、コードを分離するにはコンパイルに多くの時間がかかる

41 / 50
最も小規模なアプリケーションであっても、他のコンポーネントやライブラリとの従属条件があります。したがって、従属条件を理解／管理することは、デプロイメント・パイプラインのフローを維持するために欠かせない、継続的デプロイメントの重要な活動です。

あなたは、2つのライブラリを使用するアプリケーションを構築しました。それぞれのライブラリは、基盤となる3つ目のライブラリに依存していますが、別々のバージョンを参照しています。これによって固有の従属条件が生じています。

この従属条件を解決または防止するための**最適な**方策はどれでしょうか？

A) すべてのライブラリを単一のライブラリにまとめ、直接そのライブラリを参照して問題を回避できるようにする
B) バージョン管理を用いることでライブラリを管理し、この種類の従属条件を作成したら直接表示できるようにする
C) 大きなボードに付箋を貼ってすべての従属条件の概要を視覚的に記し、フローを追跡できるようにする
D) ツールチェーンのごく一部のみをチェックインし、チェックイン時に問題が発生しても容易にデバッグできるようにする

42 / 50
継続的デプロイメント環境においては、エラーを素早く検出したり、必要に応じてロールバックしたりできるよう、すべてをバージョン管理することが重要です。

しかし、バイナリー出力をバージョン管理することはお勧め**できません**。

この例外が生じる理由は何でしょうか？

A) バイナリー出力は非常に大きなファイルになる傾向がある。このファイルはビルドごとに変化し、自動的に更新される。
B) 複数のチーム・メンバーがバイナリーファイルに取り組むため、これをバージョン管理することは実際的ではない。
C) バイナリー出力はコンパイラ向けのインプットであり、すでにバージョン管理されている。
D) 通常のビルド・プロセスの一部として再コンパイルが実行されているため、これを行う必要がない。

43 / 50
あなたは、ITインフラの全てを管理するために全体論的アプローチを採用したいと望んでいます。

2つの概念のうち、このアプローチはどちらを**最も**ベースにしているでしょう？

A)
1. インフラストラクチャの望ましい状態は、変更管理された構成を通して特定すべきである。
2. インフラストラクチャの実態の監視やイベント管理を通して常に知るべきである。

B)
1. インフラストラクチャの望ましい状態は、変更管理された構成を通して特定すべきである。
2. インフラストラクチャの実態を測定機器の使用やインシデント管理を通して常に知るべきである。

C)
1. インフラストラクチャの望ましい状態は、バージョン管理された構成を通して特定されるべきである。
2. インフラストラクチャの実態を、現在のインシデントやイベント管理を通して知るべきである。

D)
1. インフラストラクチャの望ましい状態は、バージョン管理された構成を通して特定されるべきである。
2. インフラストラクチャの実態を、測定機器の使用や監視を通して知るべきである。

44 / 50
チームは優れた協業手法を採用し、作業チケットを同期させています。CTOは「現地現物」で、運用チームがいかに機能しているか把握しています。リリース後は、運用チームは常に運用インフラストラクチャを再評価しています。

この手法を改善するための**最適な**アドバイスはどれでしょうか？

A) 何もしない。再評価が常に行われているため、これ以上改善できない。
B) 運用環境のインフラストラクチャとアクセス・コントロールをモデル化する方法を模索する。
C) 運用インフラストラクチャが自動プロセスになるよう見直す。
D) デプロイメント・プロセスに関する知識を開発チームと共有する。

45 / 50
運用の変更を開発へ通知するために運用側にとって良い時期はいつでしょうか？

A) 開発は通知を受ける必要がない。運用の変更は運用チームのみの為である。
B) 即座に。開発へは可能な限り通知すべきである。
C) 翌日のスクラム・オブ・スクラム・ミーテイングで。
D) 運用チームが受け入れテストを完了した時点で。

46 / 50
あなたはあなたのDevOps組織が成熟することを望んでいます。そのためにはいろいろな方法があります。

DevOps組織の成熟を支援 **しない**方法は何でしょうか？

A) もしも、日々の活動が価値あるものであるならば、あなたのチームメンバーの判断を助けるマイルストーンのような目標を明確に定義する。
B) プロセスの定義を明確にし、支援し、チームメンバーが日々プロセスを改善することを可能にする。
C) 全ての会議の記録を残す。それであなたのチームメンバーは全てのコミュニケーションへ容易に参加できる。
D) 日々の進捗の小さな部分を識別することを助け、彼らを称賛する為に日々の活動を監視し記録する。

47 / 50
あなたはITサービスプロバイダで働いており、事業継続計画の一部として、最低限の合意されたサービスレベルを常に満たしていることを保証したいと考えています。

あなたはITサービスの継続性を確実にしたいと望んでいます。

DevOpsはITサービス継続性管理においてどのように役立ちますか？

A) DevOpsの文化的価値である密接な関係や協業は、DevOpsチームメンバーによってサービスが高い価値をもたらすことを確実にします。
B) DevOpsはシステムに障害を意図的に取り込むことによって、チームのディザスタ・ルーティンと大部屋の実践を準備します。
C) 運用は開発と共に働くから、リスク削減手法とリカバリー・オプションは、コーディングされていることが望ましい。
D) サービスレベル管理はDevOpsではより重要になる。なぜならプロセスマスタの仕事はこれを監視することだから。

48 / 50
ACMECONST社は、世界中で社員とエンジニアリングチームの数を増やすことによって世界規模での存在感を積極的に拡大しています。また、同社は年間30%の飛躍的なペースで顧客基盤を増やしています。

かつてエンジニアリングチームが1つの部屋にあった時には決定をすることは容易でしたが、今は長い時間がかかり、組織中でフラストレーションが起きています。管理承認までにより多くの段階が設けられ、プロセスの範囲は拡大し、そのためエンジニアの多くが、意思決定プロセス全体に幻滅を感じています。

提示される様々な問題のオーナーシップについても混乱が拡大しており、意志決定において躊躇を引き起こします。また、エンジニアは、追加のプロセスや官僚主義によって自分たちの創造性が抑え込まれていると感じており、エンジニアたちの士気に影響が出始めました。

このシナリオに対処する**最良**の方法は何ですか?

A) 現在のプロセスを維持するが、明確化された役割、説明責任、各プロセスのオーナーシップを確立する。リスク対生産性に重点を置くために効果的な手法を確立する。徐々に変更を加えていき、新手法の安全領域を作る。
B) 物事が整流化でき、明確な役割、説明責任、各プロセスのオーナーシップを識別するためにプロセスを再検討する。リスク対生産性に重点を置くために効果的な手法を確立する。徐々に変更を加えていき、新手法の安全領域を作る。
C) 物事が整流化でき、明確な役割、説明責任、各プロセスのオーナーシップを識別するためにプロセスを再検討する。リスク対生産性に重点を置くために効果的な手法を確立し、徐々に変更を加え、不必要なアプリケーションの失敗を阻止する手法の試みを最小限にする。

49 / 50
X-AppGo社では、優先順位と目標が異なっていることが原因となり、コロンビアの運用チームとアイルランドの開発チームで衝突が起こっています。この対立で、ビジネスに影響する課題の解決にかかる時間と工数が増加しています。

開発と運用チーム間の衝突を減らし、協業を高めるため、X-AppGo社はどの主要な手法を考慮すべきでしょうか？

A)
1. もし双方が対立を回避したいと望むならば、開発と運用のチームが互いに別々に仕事をすることを許容する。
2. 開発と運用のチームを支援するうえで役員会が全てを握る。

B)
1. 共に働く重要性について DevOps チームと会話するために会社の役員会からスポンサーを得る。
2. 開発と運用のチームに DevOps のトレーニングを行う。そのうえで彼らが互いの仕事を学ぶ。

C)
1. 開発と運用のチームが DevOps が上手く働いている他の企業を訪問して確認させる。
2. 運用と開発のチームが直面している要求の増大をより良くサポートするために資金を増加させる。

D)
1. 調和した関係を築き、信用を生み、相互理解できるよう、開発チームと運用チーム間で、互いのサイト訪問を推奨する。
2. 開発と運用のチーム間で知識が広まり互いにより効率的に働ける。

50 / 50
開発チームはDevOpsに興味を持っています。彼らは特に継続的インテグレーション（CI）に興味を抱いてました。彼らは現在3つの主要なソリューションと4つの小さめのソリューションの開発と保守を担当しています。彼らはスクラム手法を用いています。それぞれのスプリント（イテレーション）は4週間で、平均的に1つのコミットされたリリースからテスト環境までにそれぞれ10～15日かかります。そして1つのリリースから稼働まで1か月かかります。彼らは、CIを作成する努力や彼らの投資を支える管理のために定性的なビジネス・ケースを作りたいと望んでいます。.

このビジネス・ケースを**最も**支援するCIの具体的な利益（メリット）はどれでしょうか？

A) 1日に一回デプロイからテスト環境まで実行できればビジネス的メリットの増加と大幅な開発コストの削減ができる。
B) チームの精神を支援する。すでに彼らはスクラムを用いており、CIはビジネスの目に見える利益を**生まない**。
C) より良い自動化されたテスティング、全体的なリリーススピードの向上でリリースの安定性と品質を高める。
D) 1日に一回のリリースから稼働までの実行は、ビジネスの利益（メリット）を増大させ、大幅な開発コストの削減ができる。

解答と解説

1 / 50
DevOpsを組織に導入する理由として、妥当なものはどれでしょうか？

A) DevOpsでは開発スピードが速まるため、新規サービスのフィードバック・サイクルがより頻繁になる。
B) DevOpsではプロセスが最適化されて価値を高める活動のみが実施されるため、事業継続性とアジリティが向上する。
C) DevOpsではソフトウエアがより頻繁にリリースされるため、新しいサービスをジャストインタイムで提供できる。
D) DevOpsでは開発と運用が一体化されているため、両者の協業におけるムダが減る。

A) 不正解。開発スピードが速まるとフィードバック・サイクルはより頻繁になりますが、これは通常はDevOpsによるものではなく、スクラムなどの他のアジャイル手法によるものです。
B) 正解。価値の付加とプロセスの最適化は、事業継続性と企業のアジリティを高めるカギとなります。ITサービスが常にビジネスを支援するべきことの意味や、DevOpsの価値と目的について考える必要があります。 *(参考文献：C、Chapter 2)*
C) 不正解。ジャストインタイムの実現は素晴らしいことですが、それ自体はDevOps導入の妥当な理由にはなりません。この目標のためには、リーン・プロセスを導入したほうが効果的です。
D) 不正解。2つのチームをまとめるだけでは、ムダを確実に減らすことはできません。ムダを減らすには手法を変える必要があります。

2 / 50
CTOは、DevOpsを導入する際にある種のリーンの概念を適用すると最も効果的であると考えています。

次のリーンの概念または手法うち、DevOpsを導入する時に **最も**効果的であるのはどれでしょうか？

A) **カイゼンと５S。**アジャイルやDevOpsはリーンの主要な概念に基づいている。そしてカイゼンと５Sはリーンの基本である。それらがDevOps導入に際して最も効果的である。
B) **先行カイゼン 。**DevOpsでは、運用から開発へのフィードバックが要求される。先行カイゼンは、上流へのフィードバック・ループを作る。DevOpsにこの原則を適用するとフィードバック・ループを支援する。
C) **大部屋システム 。**DevOpsは、異なった管理形式のプロセスを統合する。大部屋システムは、全体プロセスの見える化を支援し、DevOps導入を成功させる。
D) **一個流しと自工程完結。**DevOpsは上流工程の構築と単一のバリューストリームフローから効果が出る。一個流しはこれを可能にし、自工程完結は効率化とそのフローを作るのに役立つ。

A) 不正解。リーン、アジャイルとDevOpsが相互に関係しているにも関わらず、カイゼンと５SはDevOpsの開始の成功を支援する最適な組み合わせではありません。DevOps導入後、カイゼンは継続的な改善のために利用されます。そして５Sは、良い手順を維持するために使用されます。どちらもDevOps導入が成功した後になります。
B) 不正解。フィードバックはいつでも歓迎されます。しかしこれがDevOps導入時にリーンの最も効果的な運用を保証するものではありません。
C) 不正解。見える化は役立ちますが、DevOps導入時で最もインパクトの強いリーンの手法ではありません。
D) 正解。実行可能な、単一の、デプロイメント・パイプラインの構築はDevOps導入を成功させるための手助けとなります。DevOpsで最も重要なことは、開発から運用までの上流工程を構築することです。特に単一のデプロイメント・パイプラインのために必要です。自工程完結はこの整流化を実現する最も効果的な仕事の行動習慣です。*(参考文献: C, Chapter 4)*

3 / 50
従来のプロジェクトと比較して、DevOpsでプロジェクトを成功させるためには何を変更する必要がありますか？

A) プル型システムと一個流しを使用する、ITサービスのサプライチェーンを構築する必要がある。
B) サービスの素早い保守に向けて、開発者が運用チームに加わる必要がある。
C) 運用は開発チームのために働く必要がある。だからこそ、DevOpsと呼ばれている。
D) 運用チームのメンバーは、開発チームに参加する必要がある。

A) 正解。プロジェクトは、プロセスが自動化ベースのプル型システムを使用して有益なITサービス（または製品）を作成した場合に成功します。 *(参考文献：C、Chapter 4、および参考文献：B、Chapter 1)*
B) 不正解。DevOpsは、開発者が運用に加わることを意味するものではありません。
C) 不正解。協業だけでは、プロジェクトでDevOpsを成功させることはできません。
D) 不正解。DevOpsは、単に運用が開発に加わることを意味するものではありません。

4 / 50
DevOpsを採用しているチームはすべて、共同体と呼ぶことができます。

共同体を機能させるうえで、**最もよく**当てはまる原則はどれでしょうか？

A) 動的な協業
B) 継続的コミュニケーション
C) サイクルの短縮
D) 説明責任の共有

A) 不正解。動的な協業は共同体の原則ではありません。
B) 正解。DevOps共同体の原則は、継続的コミュニケーション、明確に定義された共有の目標、および理解に対する動的な調整と修正にあります。 *(参考文献：A、Chapter 2)*
C) 不正解。サイクルの短縮は共同体の原則ではありません。
D) 不正解。説明責任の共有は共同体の原則ではありません。

5 / 50
DevOpsの導入のために利用できる、数多くの知識、標準、および手法が存在します。

DevOps導入の成功のために**最も重要**と見なされるものはどれでしょうか？

A) CMMI Level 3
B) 規律あるアジャイル
C) ISO 20001
D) PMI PMBok

A) 不正解。CMMI認証は役に立つ可能性もありますが、DevOpsの導入に最も重要というわけではなく、まして具体的なCMMIレベルは関係ありません。
B) 正解。規律あるアジャイルは、DevOps導入を成功させるうえで最も重要な要件です。 *(参考文献：C、Chapter 4i)*
C) 不正解。ISO 20001認証は役に立つ可能性もありますが、DevOpsの導入に最も重要というわけではありません。
D) 不正解。PMI PMBokはプロジェクト管理に関する参考文献であり、DevOpsの導入に最も重要というわけではありません。

6 / 50
協業は、効果的なDevOpsの4本柱の1つです。

協業はなぜそれほど重要なのでしょうか？

A) 協業はDevOpsにおいて一個流しを実現する唯一の方法であるから。
B) DevOpsのビジネス価値は、チーム間の協業を通じて実現するから。
C) DevOps組織は小規模であり、チームは効果的に協業する必要があるから。
D) 協業は変更を促進し、容易に成功をもたらすから。また、開発コストを削減できるから。

A) 不正解。協業しなくても一個流しは実現できます。両者はメカニズムが異なります。
B) 正解。関係するすべてのチーム（開発と運用を含む）間での協業は、コミュニケーション、自動化、およびソフトウエア品質を向上させるため、優れたビジネス価値の実現に極めて重要です。 *（参考文献：A、Chapter 6および7）*
C) 不正解。DevOps組織は非常に大規模になる場合もあります。チームが十分なビジネス価値を付加するには、協業が必要です。
D) 不正解。変更を促進し、容易に成功をもたらすものはツールです。協業は容易ではありません。協業によって開発コストは削減できますが、これは主要目標ではありません。主な目標は、品質を向上することにあります。

7 / 50
あなたは小規模なDevOps企業のオーナです。同僚は5名で、障害のある子ども向けのモバイル・アプリを構築しています。チームが最も誇りを持っているアプリの1つは、自閉症の子ども向けのアプリで、これは自分でスケジュールを組めるようにするものです。

このアプリが大成功を収めたため、チームはスケジュールを自分で組むことでメリットを得られる他の人々のために、このアプリの機能を拡張してほしいと頼まれています。このリクエストによってコードが複雑化し、技術的な課題が生じることは確実で、チームで取り組む必要があります。

この仕事は非常に報酬が高そうであったため、あなたは仕事を引き受けました。しかし数週間後、チームは言い争いをしていました。あなたも腹を立て、チーム・メンバー全員の業務内容に注意を払うようになりました。あなたは常にメンバーとともに働き、コード・エラーを見つけて素早く修正できるようにしました。

十分な作業が完了したにも関わらず、チームはまだ腹を立てており、あなたはDevOpsの考え方が部分的に欠落していることに気づきました。

この問題を解決するための**最適な**戦略はどれでしょうか？

A) 技術分野以外の補完的なスタッフを採用し、チームにメンバーを追加して多様性を高める
B) 社外のDevOpsエキスパートに、チーム・メンバーの支援と指導を依頼する
C) 共通の目的を見つけ、その目標に向かって共に努力し、協業を強化する
D) 数か月かけてまずチーム・ビルディングに着手し、互いについて学ぶ

A) 不正解。チームの拡張は、必要な活動を実施できるエキスパートがいない場合は賢明かもしれません。しかし、チームの効率性を保つために技術分野以外の人材を雇うことは、賢明でも望ましくもありません。このケースでは十分な作業が完了しているので、既存の問題をさらに悪化させるだけです。信頼と協業が欠けているからです。
B) 不正解。素晴らしいアイディアですが、DevOpsの考え方のうち、あなたの会社で欠けている協業と共感の問題は解決できません。協業は、目標を共有し、ともに成功することで強化されます。
C) 正解。現在の問題を解決する適切な方法です。問題解決に多大な時間を費やしたり、場合によっては仕事を失ったりすることなく、言い争いを減らすことができます。目標の共有は、信頼、共感、および協業を培います。 *(参考文献：A、Chapter 7)*
D) 不正解。場合によってはうまくいき、あなたの目標である、互いに信頼し合って協業するチームを構築できるかもしれません。しかし、問題の解決に時間がかかりすぎて仕事を失うリスクがあるため、最適な方法とは言えません。

8 / 50
あなたは、自分の開発チームが本当に良いチームだと感じています。

彼らが単にグループ（人の集まり）では無くて、チームになっていると認める徴候は何ですか？

A) チーム会議で合意したルールに従っている。
B) 自分たち自らが主導する効果的な会議を行っている。
C) チームは、共通の目的に向かって、安定した作業ペースを保っている。
D) チームは、責任あるチームメンバーに問い合わせることで問題を解決している。

A) 不正解。グループの人々は、ルールに従うことができます。これはチームを作ることに必要不可欠ではありません。
B) 不正解。グループの人々は、効果的な会議を開くことができます。これはチームの徴候に必要不可欠ではありません。
C) 正解。真のチームとは、安定した作業ペースを確実にします。そして共通の目標に向かって働き続けます。*(参考文献: A, Chapter 9)*
D) 不正解。チームは問題を共に解決します。更にチームメンバーに質問することから始めません。DevOpsでは、責任を問わない文化です。

9 / 50
AppAtoZ社は、iPhoneやアンドロイドプラットフォーム向けのモバイル・アプリケーションの開発とデプロイにおいて驚異的な速さで成長し続けています。

この新興企業の開発チームは、現行のモバイル・アプリケーションの機能拡張を強気なスケジュールで迅速にデプロイするために大変なプレッシャーを経験しています。チームメンバーは過去６ヶ月間にわたり、平均して週60時間働き続けています。上層部は人員を増やすことに消極的で、それよりも運用と開発のコストを削減して収益を増やすことに関心を持っています。

ここ数ヶ月で、社員の欠勤や病欠が増加し、退職する社員もいたことから、現在働いている社員の仕事量が増加しました。再雇用や新入社員の育成を行っても、開発チームの仕事のプレッシャーはすぐに軽減できません。

社員の疲弊やストレスに対処するには、どの**長期**戦略を考慮する必要がありますか？

A)
- 作業負荷をより良く管理するためにチームに正社員と契約社員の両方を雇う。
- 上層部と開発部門が疲弊の一因となる作業環境の全ての要因を列挙する。
- それらの課題に対処する計画を立てる。

B)
- 能力がないため上層部を解雇。
- 新たに、持続可能で現実的なワークライフバランスや文化をどの様に作るか、ノウハウを持ったより能力のある上層部チームを採用。
- 開発チームに自身のワークライフバランスを考える時間を与える。

C)
- 開発チームに休息を取らせる。
- 年間の作業ピーク時に増加する開発作業負荷を管理するための契約社員。
- 開発者が必要な時に相談できるメンタルヘルスの専門家を用意する。

D)
- 開発チームに休息を取らせる。
- 上層部と開発部門が疲弊の一因となる作業環境の全ての要因を列挙する。
- それらの課題に対処する計画を立てる。

A) 正解。この解答の全ての選択肢が長期的です。他の解答は少なくとも一つは短期の選択肢が含まれます。*(参考文献: A, Chapter 8)*

B) 不正解。上層チームを解雇しても長期的には今ある課題を解決できません。これは短期的対処になるでしょう。バランスを取るために時間を設けることは賢い考えですが、計画がないのであれば、自然と効果は表れないでしょう。

C) 不正解。小休止を取るのは良い考えですが、短期でのみ効果があります。メンタルヘルスの支援もよいでしょう。しかし根本的な変化はなく、短期での解決策です。契約開発者の雇用もまた効果があり、良い考えです。

D) 不正解。計画を立てることと、作業環境の要因を特定することは、良い考えです。しかし、この解答よりももっと良い解答があります。なぜなら、小休止を取ることは短期的にのみ有効だからです。

10 / 50
あなたは開発者としてDevOpsチームで働いており、チーム・メンバー全員による協業を促進しようと試みています。チームには、先輩にあたる男性の開発者が2名、後輩にあたる女性のシステム管理者が1名います。

チーム・メンバーは前途多難なスタートを切っており、うまくやっていくことができませんでした。あなたはその時点で仲裁に入り、仲良くやっていけるよう手助けを試みました。この仲裁が功を奏し、彼らは口論をやめて、多くの仕事を成し遂げました。

あなたは今、後輩の管理者が先輩の開発者らにひたすら同意する傾向があることに気づきました。

この傾向には、どのように対応したらよいでしょうか？

A) チーム・メンバーが口論をしていない限り、そのまま放置しておく。彼らは何とかうまくやっていくであろうし、あなたが邪魔をするべきではない。
B) 運用がビルドについての懸念を表明することは重要であるため、管理者に言いなりになることをやめさせ、自分の意見を明確に示すよう指導する。
C) 先輩の開発者らのほうが責任が大きいため、彼らにもっと思いやりを持つよう促し、後輩が気づかない場合は運用の懸念について考えるよう依頼する。
D) あなた自身は開発者であるが、当面の間、運用チームに加わり、後輩の管理者に手本を示す。

A) 不正解。この状況では対立を解決する必要があります。後輩メンバーは、対立解決の手法として順応を使用していますが、これは生産的な手法ではありません。ビルドにおいてスケジュール通りに懸念を解決できないというリスクを避けるため、それぞれの役割で責任を果たす必要があります。
B) 正解。これが状況に適した解決策です。この状況では、対立の解決が必要です。後輩メンバーは、対立解決の手法として順応を使用していますが、これは生産的な手法ではありません。さらにこの手法では、運用がビルドに関する懸念を表明しないというリスクが生じます。 *(参考文献：A、Chapter 7および14)*
C) 不正解。誰もが同等の責任を担っており、後輩・先輩の別や性別は関係ありません。また開発者は、ビルドに関する問題を運用担当と同じようにとらえることはできません。だからこそDevOpsでは、異なる分野の相互作用が必要です。
D) 不正解。手本を示すことが最適な指導方法だったとしても、チームを移ることはできません。開発者は、ビルドに関する問題を運用担当と同じようにとらえることはできないため、これによってビルドの品質が損なわれる可能性があります。

11 / 50
アジャイルとスクラムによって、ソフトウエア開発のスピードと予測可能性が高まる理由はどれでしょうか？

A) 設計前に、より効果的で完璧な要件収集／処理が可能になるから
B) 小規模かつ自律的な自主編成／計画チームが可能になるから
C) プロダクトオーナが日常的なスタンドアップ・ミーティングに参加できるから
D) プロジェクト管理者が必要に応じて素早く優先順位を変更できるから

A) 不正解。これはむしろウォーターフォール・モデルのアプローチです。
B) 正解。これは、スピードと成果を向上させるために、スクラムとアジャイルの原則で提案される方法です。 *(参考文献：A、Chapter 4)*
C) 不正解。プロダクトオーナが参加できたとしても、ソフトウエア開発のスピードと予測可能性を向上させるという約束を果たすこととは関係ありません。
D) 不正解。プロジェクト管理者は優先順位を変更すべきではありません。これはプロダクトオーナの役割です。

12 / 50
軽量化したITSMとは何でしょうか？

A) 事業継続性に焦点を絞ったITSM
B) 標準として提案された新しいITILバージョン
C) ITILプロセスの不完全な導入
D) リリース管理指向のITSM

A) 正解。ITILは重厚長大で、DevOpsの迅速なプロセスには向かないように思われます。軽量化したITSMはDevOpsに合わせて再調整されたもので、必要最低限の情報を用いて事業継続性に焦点を絞っています。 *(参考文献：C、Chapter 4iii)*
B) 不正解。このようなITILバージョンはまだ提案されていません。
C) 不正解。軽量化したITSMは不完全な導入というわけではなく、事業継続性と管理負荷の軽減に焦点を絞って要点のみをまとめたバージョンです。
D) 不正解。ITSMはリリース管理ではなく、サービスマネジメント指向です。ITSMの概念においては、リリースとはサービスを支えるプロセスを指します。

13 / 50
あなたはリーンやアジャイル手法を導入している企業で働いています。CEOは、DevOpsという新しい手法をさらに導入する価値について、納得していません。

あなたはDevOpsのエキスパートです。あなたは、ソフトウエアを開発している企業なら例外なく、DevOpsからメリットを得られると考えています。

DevOpsによって、あなたが働いている企業に追加されるものは何でしょうか？

A) より優れたユーザー・ストーリーと機能要件を作成できる
B) 顧客が定期的なアップデートで苦労しないよう、リリース頻度を削減できる
C) 開発と運用の双方を担えるようにプロフェッショナルをトレーニングできる
D) ビジネス成果をサポートするコードを作成し、より迅速にリリースできる

A) 不正解。あなたはアジャイル手法ですでに、優れたユーザー・ストーリーと機能要件を作成しているはずです。DevOpsは基盤としてアジャイル手法に準拠します。運用が早期に関与するようになったとしても、アジャイルに従って作成されたユーザー・ストーリーや機能要件は、DevOpsでもそれほど変わりません。
B) 不正解。DevOpsはより迅速で連続的なリリースを通じて、ビジネスに価値を素早くもたらします。これは、リーンの原則でもあります。新機能に実質的な付加価値があるのであれば、顧客が頻繁にアップデートすることは問題にはなりません。そのうえ、頻繁にリリースを行いつつ、エンドユーザーには予定時刻にのみアップデートを実施させることも可能です。
C) 不正解。DevOpsの目標は、運用と開発が連携することであり、それぞれが両方の役割を果たすことではありません。非常に小規模な企業では、理論的には可能ですが、これはDevOpsそのものではありません。
D) 正解。リーンやアジャイル手法ではすでに、ビジネス価値と機能要求の変化にフォーカスしています。DevOpsは、運用の早期からの関与と可能な限りの自動化を通じた継続的デプロイメント・パイプラインの作成に焦点を絞ることで、本番リリースの頻度を増やし、ビジネス成果を直接サポートします。 *(参考文献：C、Chapter 1)*

14 / 50
上級経営陣は、ビジネス目標に対するIT部門からのビジネス支援と協力を強化するよう求めています。さまざまな対策が考えられる中、あなたはCIOとして、運用における管理負荷を軽減することに決めました。

既存のサービスレベルマネジメント（SLM）を改善するために、DevOpsはどのような方法で**最も**役立ちますか？

A) ITSMは重厚長大なアプローチであるため、ITSMのベストプラクティスを廃止
B) 開発と運用の間で、より優れたオペレーショナルレベルアグリーメント（OLA）を設計
C) ITILバージョン3に基づいて新しいITSMプロセスを導入
D) ITSMを再調整して必要最低限の情報（MRI）を使用

A) 不正解。ITSMのベスト・プラクティスを廃止しても、SLMは改善されません。
B) 不正解。開発と運用の間でより優れたOLAを設計しても負荷の軽減には役立たず、官僚主義が強まるだけです。もちろんこれはDevOpsの主眼ではありません。
C) 不正解。DevOpsにそぐわない、重厚長大なプロセスが増えるだけです。
D) 正解。これはDevOps導入に欠かせない条件で、軽量化したITSMを実現します。 *（参考文献：C、Chapter 4iii）*

15 / 50
業務系システム（SoR）アプローチを採用している企業に**最も**適している、DevOpsの導入はどれですか？

A) 協業
B) 継続的デリバリー
C) トヨタ方式

A) 正解。これは、迅速で頻繁なITサービスと信頼できる運用を提供することに焦点を絞っており、SoEおよびSoRに最も適しています。 *（参考文献：C、Chapter 8）*
B) 不正解。デジタル製品ベンダーに最も適しています。
C) 不正解。ITサービス・プロバイダに最も適しています。

16 / 50
大部屋システムを使用する**主な**メリットは何でしょうか？

A) 顧客に不満を表明するよう促し、チームが継続的な改善に向けて十分なフィードバックを得られるよう保証する
B) チーム内のストレスに対処し、チーム・メンバーが持続可能なペースを維持できるようにする
C) 日常的なバグのレポートを改善して、やり直しを少なくし、バグが他のワークステーションに広がることを減らす
D) 迅速な情報収集／共有により、現状に基づいて素早く意思決定を下す

A) 不正解。大部屋は顧客に不満を表明するよう促すものではありません。
B) 不正解。ストレスの多い状況への対処に役立つ場合もありますが、主なメリットではありません。
C) 不正解。大部屋は、バグ・レポートの改善には役立ちません。
D) 正解。大部屋または作戦室は、トヨタ生産システム/リーンのツールであり、プロジェクト・チームがあらゆる関連情報を把握できるよう支援するとともに、小規模チーム内での迅速なやり取りと情報共有を促進し、意思決定に向けた情報収集を加速させます。 *(参考文献：C、Chapter 7iii)*

17 / 50
あなたは、DevOpsを採用し始めたばかりのソフトウエア企業で働いており、組織文化だけではなく手法とツールも変える必要があると気付いています。あなたの会社ではすでに、問題がないか確認するためソフトウエア・プロジェクトを監視しています。

あなたは、次のような対策を追加するよう提案しています：
- 変更管理プロセスを自動化
- 誰かが承認を得ずに変更を行うことを防ぐため、アクセス・コントロールを導入

これらはなぜ必要なのでしょうか？

A) 自動化は、信頼性を維持しつつ、より迅速に変更を実施できるようにする。アクセス・コントロールは、問題解決の経験則を防ぎ、計画外のサービス停止を防ぐために必要である。
B) 自動化は、変更の数を抑えるために必要である。アクセス・コントロールは、こちらが気づかないうちに勝手に、顧客が自らソフトウエアを変更することを防ぐ。
C) 自動化は、運用を不要にするために役立つ。アクセス・コントロールは、DevOps プロジェクトがクラウド環境で行われており、高いリスクを伴うため、導入する必要がある。

A) 正解。引用：「一般的に私たちは、対象をロックダウンして承認プロセスを確立することを好まないが、本番インフラストラクチャに関してはこれは必須である。当然の結果として、私たちは本番環境と同様にテスト環境を扱うべきだと信じていることから、同じプロセスが双方に適用される。組織外の人々だけではなく、組織内の人々（運用スタッフも含む）からの不正アクセスを防ぐために、本番環境をロックダウンすることは不可欠である。さもなければ何か問題が発生したときに、疑わしい環境にログインし、問題を解決するためにつつき回したくなるものだ（礼儀正しく言えば「問題解決の経験則」である）。これは次の2つの理由から、恐ろしいアイディアと言ってよい。まず、サービス停止を招くことが多い（人々は手当たり次第にリブートやサービスパックの適用を試みる傾向がある）。次に、後にまた問題が発生した場合に、誰がいつ何を実施したのかという記録が存在しないため、後に発生した問題の原因を解明することが不可能になってしまう。この状況では、環境を既知の状態に戻すため、再びゼロから構築し直したほうがよい」 *（参考文献：B、Chapter 11）*
B) 不正解。自動化は、プロセスへの信頼性を損なうことなく、実施可能な変更の数を増やすために役立ちます。アクセス・コントロールによって顧客を締め出すことはできますが、これは主要目標ではありません。
C) 不正解。運用は決して不要にはなりません。ただし、困惑や負担が減ったと感じる可能性はあります。DevOpsプロジェクトは、クラウド環境内で行われる場合も、そうではない場合もあります。アクセス・コントロールは、クラウド環境におけるソフトウエアのセキュリティ確保に役立ちますが、これは主要目標ではありません。

18 / 50
仮想化とクラウド・コンピューティングは、DevOps手法を支え、促進できる技術です。

これらはどのようにDevOpsに役立ちますか？

A) クラウド・コンピューティングはリモート・アクセスを可能にし、顧客による管理を向上させる。
B) 仮想環境は標準化が容易であり、ハードウエアの使用効率が向上する。
C) 仮想インフラストラクチャは分かりやすく、保守が不要である。

A) 不正解。確かにクラウド・コンピューティングによってリモート・アクセスは容易になりますが、それだけで顧客の管理が向上するというわけではありません。いずれの点もDevOpsにはまったく役立ちません。
B) 正解。仮想化によってCIとテスト・インフラストラクチャの統合が容易になり、デリバリー・チームへのサービスとして提供できます。また、ハードウエアの使用効率も高まります。仮想化により、物理環境に関して単一のハードウエア構成で標準化しつつ、さまざまな異機種環境／プラットフォームを仮想的に実行できます。 *（参考文献：B、Chapter 11）*
C) 不正解。仮想インフラストラクチャは本質的に理解が容易というわけではありません。さらに、保守と管理も必要です（ただし、オンプレミスのインフラストラクチャと同様ではありません）。したがって、これは妥当な理由とは言えません。

19 / 50
サービスレベルアグリーメント（SLA）は、顧客との合意内容を具体的に定めるもので、あらゆるプロジェクトにおいて重要です。しかしDevOpsでは、SLAは他の重要な目的にも役立ちます。

この目的とは何でしょうか？

A) 顧客はDevOpsチームのためにSLAを作成する責任を負う。したがって、運用の責任としての正式なタスクに取って代わる。
B) SLAでは顧客は、すべての非機能要件を指定できるため、開発はこれらに焦点を絞ることができる。
C) SLAは受け入れ可能なサービスレベルを指定する。開発はSLAを理解し、運用がそれを維持できるよう支援する必要がある。

A) 不正解。SLAは常に、顧客とサービス提供部門との契約であり、双方がその内容に関わります。
B) 不正解。SLAに由来する非機能／機能要件もありますが、大半はSLA経由ではなく直接DevOpsチームに提供されます。また、非機能要件の提供は、SLA自体の目的ではありません。
C) 正解。通常、SLAに記載された条件は、運用に最も関係があります。開発は、可能な限り運用の業務を容易にすることで、運用を支援すべきです。この点が、DevOpsと通常の開発との違いです。 *（参考文献：B、Chapter 12）*

20 / 50
新しいプロダクトのためにあなたのチームはデプロイメント・パイプラインを作る必要があります。継続的インテグレーションの一部として、あなたはパイプラインの中でコミット・ステージを定義する必要があります。あなたはあなたのチームの仲間とこのステージについて議論します。

プロセスマスタ（管理者）が言いました。「完了の定義（DoD）は、コミットステージの前または最中に定義すべきだ。コミットした時にコードが完了していない場合、作業を停止すべきである。」

これは正しいでしょうか？

A) はい、もし仕事が完了していなければ、プロセスマスタは職務を果たしていません。この問題は早急に解決すべきです。
B) はい、顧客に付加価値を与えていないため、完了していない仕事は、コミットすべきではありません。
C) いいえ、完了の定義とは、顧客との会議でのみ定義できます。完了の定義を待つことはあまりにも仕事を遅らせます。
D) いいえ、デプロイメント・パイプラインでの仕事は、常に継続されるべきものです。もしコードが完了していなければ、それはただ停止する必要があります。

A) 不正解。プロセスマスタは、完了の定義と完了していないコードがコミットされたとき、作業を停止すべきであるという事を確実にするという職務があります。しかし、コードが完了していないコミットされた場合でもプロセスマスタが必ずしも職務を果たしていないということにはなりません。
B) 正解。仕事が完了していないとき、デプロイメント・パイプラインの仕事を開始することは、顧客に十分な価値をもたらしません。一個流しを考えれば、これはより価値のある仕事の流れを遅らせるでしょう。 *（参考文献：B, Chapter 3）*
C) 不正解。完了の定義とは、プロジェクトにおいて合意された重要な事の一つです。それは、顧客との会議中に定義されません。コーディングを開始した時、常に完了の定義を意識すべきです。さもなければ何時コーディングを止めるかが解らないでしょう。
D) 不正解。コードに何か問題がある、または価値を付加できないというのは、デプロイメント・パイプラインを停止させ修復する、あるいは1個流しのパイプラインでもっと価値のあるものを得るための十分な理由です。

21 / 50
より幅広い経歴や文化を持つチームの多様性を高めることの**最大の**メリットは何でしょうか？

A) 多くの経験や視点をもたらす。
B) チーム内での摩擦を削減する。
C) 独創性と新たな洞察を生む能力を制限する。
D) 決定に至るまでにかかる時間が長くなる。

A) 正解。多様性には、人種、性別、性的特質、地位、教育レベル、更に種々の勤務経験といった側面を含む広範囲な背景が包まれます。これら全ての独特の様相が組織に多くの経験と視点をもたらします。 *(参考文献: A, Chapter 7)*
B) 不正解。多様性の増加が圧力と摩擦を増加させる可能性はあります。何故なら異なった文化的価値は共に働かなくてはなりません。
C) 不正解。より多様性があるとは、より異なった視点があるという事です。通常この要素はより独創性を引き出します。
D) 不正解。一般的には、これはデメリットであるとみなされます。（たとえより遅い意思決定が得策であるにせよ）さらなる多様性はコンセンサスを得るまでに時間が掛かるかもしれません。

22 / 50
あなたはDevOpsチームを1つ擁する小規模企業で働いています。DevOpsチームは、複数のコンポーネントで構成されるアプリケーションに取り組んでいます。新しいコンポーネントもあれば、更新が必要なコンポーネントもあります。

現時点では、各コンポーネントには独自のデプロイメント・パイプラインがあります。チームは、大半のデプロイメント・パイプラインで行っている継続的デリバリーに誇りを持っています。チームの生産量は大きく、高品質です。

どのような対応が必要でしょうか？

A) さまざまなパイプラインを維持し、チームに継続的デリバリーを拡張するよう促す
B) 継続的デリバリーを行っているパイプラインのみを維持し、他の継続的デリバリーを行っていないパイプラインを融合させる
C) 開発用と保守用に1つずつ、2種類のパイプラインのみを維持する
D) 複数のパイプラインを有するリスクを説明し、チームとともに単一のパイプラインに向けて取り組む

A) 不正解。1チームあたり複数のデプロイメント・パイプラインを有することには、リスクが伴います。さまざまなパイプラインの優先順位の設定は難しく、あるパイプラインで何かが優先されると、別のパイプラインの作業を中断しなければなりません。これにより、DevOpsが回避しようと試みている、一定の無秩序やマルチタスクが復活してしまいます。単一のデプロイメント・パイプラインを用いることで、効率性とビジネス価値が増大します。
B) 不正解。あらゆるデプロイメント・パイプラインは、継続的デリバリーから多大なメリットを得ますが、継続的デリバリー手法が存在することで、複数のデプロイメント・パイプラインのリスクが減るわけではありません。
C) 不正解。所有しているデプロイメント・パイプラインが2つのみで、機能的に分離していたとしても、同じリスクが生じます。
D) 正解。これが最善の方法です。できればチームは、継続的デリバリーによる手法から十分に学び、これをパイプライン全体に組み込みます。 *(参考文献：B、Chapter 13)*

23 / 50
あなたは、２年前からDevOpsに取り組んでいる、中規模から大規模な組織であるCompany Builders社を査定しています。

同社は現在の成熟レベルを確認するためにあなたを雇いました。査定が完了し、あなたは改善のために提案をしなければなりません。同社は、次の成熟度レベルの *レベル２-定量的管理が行われているに達するために、どの分野を重視すべきか知りたがっています。*

あなたは、ほとんどの分野は *レベル１ ー一貫性がある*に達していることを知りました。ただし、次の２つの分野は例外です。

1. **環境とデプロイメント。**この分野は調整されたデプロイメントを管理し、リリースとロールバックプロセスをテストしました。
2. **ビルド管理と継続的インテグレーション。**この分野では、常に自動化されたビルドとテストを行い、どのビルドも自動化されたプロセスを使ってソース管理から再作成できます。

以上の情報に基づいて２つの分野の成熟度を判定してください。さらに、改善の重視点を推奨してください。

Company Builders 社がレベル２に進むには、２つの分野のうちどちらに重点を置くべきでしょうか？

A) 環境、デプロイメントとビルド管理、継続的インテグレーションはどちらもレベル０．仕事はどちらも同時に完了しなければならない。
B) 継続的インテグレーションはどちらもレベル１以上。仕事を進めるために他の分野の作業を完了させなければならない。
C) 環境、デプロイメントはレベル０。　ビルド管理、継続的インテグレーションはレベル１。環境、デプロイメントに最初に注目すべき。
D) 環境、デプロイメントはレベル２。ビルド管理、継続的インテグレーションはレベル０。ビルド管理、継続的インテグレーションのみに注目すべき。

A) 不正解。分野１はレベル２そして分野２はレベル０
B) 不正解。分野１はレベル２そして分野２はレベル０
C) 不正解。環境、デプロイメントはすでにレベル２であり追加の作業の必要がありません。
D) 正解。分野２はレベル０でレベル２を目指している組織ではまずレベル１に達すべきです。*(参考文献: B, Chapter 15)*

24 / 50
あなたの会社は、顧客にオンライン・バックアップ・サービスを販売しています。現在、ある顧客がサービスに新機能を組み込むよう求めてきました。1週間以内に新機能を提供できなければ、取引先を変えられてしまう可能性があります。

あなたは、この新機能が重要であると考えており、開発チームがそれを素早く構築できることを知っています。しかしあなたは、次のような問題に直面しています：

- テスト担当がバグをクローズするのに長時間かかる。
- 開発者がかなり前に修正したバグを、テスト担当が見つけている。
- アプリケーションの動作をめったにデモンストレーションできない。
- 展示の機会がほとんどない

あなたの問題は何でしょうか？どのように解決すべきでしょうか？

A) バグの多いコードを導入している。デプロイメント・プロセスの理解を深め、協業を強化し、より規律ある方法で働くことで、この問題を解決すべきである。
B) 構成管理が不十分である。開発と運用の協業を強化し、監視および記録を拡大し、仮想化を導入することで、この問題を解決すべきである。
C) 継続的インテグレーション・プロセスが適切に管理されていない。自動テストとコミット・ステージを加速させ、継続的インテグレーション・プロセスの理解を深めることで、この問題を解決すべきである。
D) テスト戦略が効果的ではない。テストを自動化してテスト担当と他のチーム・メンバーの協業を強化することで、この問題を解決すべきである。

A) 正解。バグの多いコードの導入や不十分な導入が招く状況としては、導入に長時間かかる、ベロシティが低い、リリース日に関して疑念が生じる、継続的インテグレーション環境への信頼が失われる、バグの修正に長時間かかる、開発者がかなり前に修正したバグが見つかる、デモンストレーションや展示がほとんど行われない、などがあります。この選択肢の回答は、こうした問題を解決します。
（参考文献：B、Chapter 15）
B) 不正解。不十分な構成管理が招く具体的な状況としては、本番環境の障害が説明されない、デプロイメント・イベントを管理できない、環境の構成に長時間かかる、障害発生時の復旧に時間がかかる、などがあります。この選択肢の回答は、こうした問題を解決します。
C) 不正解。継続的インテグレーション・プロセスを適切に管理しないことが招く具体的な状況としては、デプロイメントが1日1回よりも少ない、コミット・ステージに損傷がある、リリース間のインテグレーション段階が長い、などがあります。この選択肢の回答は、こうした問題を解決します。
D) 不正解。非効果的なテスト戦略が招く状況としては、バグが再発する、バグ修正に長時間かかる、顧客からの苦情が多い、製品の品質が低い、開発者がストレスを感じる、などがあります。この選択肢の回答は、こうした問題を解決します。

25 / 50
効果的なDevOpsにおける継続的インテグレーションのメリットはどれでしょうか？

A) 機能リリース前のテスト・サイクルが長い
B) 次の機能リリースまでの期間が長い
C) 機能リリースがより頻繁でタイムリーである

A) 不正解。適切なテスト手法は必要ですが、長いサイクルである必要はありません。加えて、これはメリットではありません。
B) 不正解。実際はこの反対です。
C) 正解。継続的インテグレーションでは統合が自動化されるため、より迅速かつ頻繁なリリースの実現に役立ちます。 *(参考文献：B、Chapter 3)*

26 / 50
多国籍の組織では、ダラスにある中央リポジトリーに分散された場所からのコミットをマージするにあたりに多くの難問を抱えています。分散拠点とはメキシコシティ、パリ、サン・ディエゴや英国などにあります。各地域がいつコミットを実行するかには一貫性がないため、いくつかのテストが失敗したかどうかがわからないことがあります。

ここに4つの可能性のある手法があります。

1. 壊れたビルドにチェックインしない。
2. コミット前に全てのコミットテストを常に各地域で実行する。または、自分の継続的インテグレーション・サーバにコミットテストを実施させる。
3. コミットテストが通ってから次に進む。
4. 失敗したテストはコメントアウトしない。

各拠点がそれぞれ現在抱えている問題点に対処できるようにするには、これらのうちどれが **最も** 適用可能でしょうか？

A) １と２
B) １と２と３
C) ２と３
D) ２と３と４

A) 不正解。１は、ここでは適用不能。３と４は重要です。
B) 不正解。１は重要ではありません。
C) 不正解。4は重要です。
D) 正解。これら3つの手法はこのシナリオでは最も適用可能です。壊れたビルドにチェックインするという証拠はありません。それでこれは適用できるという事ではありません。*(参考文献: B, Chapter 3)*

27 / 50
基本的なデプロイメント・パイプラインの構造を考えてください。

どのステージが機能的および非機能的レベルでシステムが正常に動くと断言できるでしょうか？

A) 自動化された受け入れテスト
B) ビルドと単体テスト
C) 手動の受け入れテスト
D) バージョン管理

A) 正解。自動受入れテスト段階では、システムは機能的および非機能的レベルで動作し、ユーザーのニーズおよび顧客の仕様を行動的に満たしていると断言します。 *(参考文献: B, Chapter 8)*
B) 不正解。ビルドテストや単体テストは新しく書かれたコードが正常であることを確認します。すでにあるビルドへの統合を確認するのでありません。
C) 不正解。この解答はたぶん正解になるでしょう。しかしながらデプロイメント・パイプラインの機能では、受け入れテストを自動化することが期待されています。
D) 不正解。バージョン管理は壊れたビルドや問題、課題を修復するために用いられます。機能的にも非機能的レベルでシステムが正常に動くという事を表すために用いられません。

28 / 50
アプリケーションが稼働するそれぞれの環境にデプロイするまで同じプロセスを使用する事がDevOpsのベスト・プラクティスです。これによりビルドが効果的にテストされることを確実にします。あなたはビルドとデプロイメントプロセスでスクリプトを使用しています。

これを行うのにどの方法が**最善**でしょうか？

A) それぞれの環境毎に一つのスクリプトを使用する。そしてバージョン管理システムでそれらを保守する。
B) 環境間の差異を明確にしたそれぞれの環境に特別な一つのスクリプトを使用する。
C) 特定の構成にはマニュアル・パラメーターを使用してそれぞれの環境で同じスクリプトを使用する。
D) それぞれの環境にデプロイのための同じスクリプトを使用し、別々に構成情報を管理する。

A) 不正解。 この保守にかかる労力と、複雑さを加えることでもたらされる潜在的なエラーを考えると、この解答は正しい選択ではありません。
B) 不正解。修正した異なるスクリプトは問題を生み、その上、プロセスに問題を起こします。それらは追跡したり解決することが容易ではありません。
C) 不正解。ビルドやデプロイでマニュアル（手動）との相互操作はすべきではありません。敏捷性のためにも、エラーが起きないようにするためにも、このプロセスに限り自動化すべきです。
D) 正解。ビルドとデリバリー・プロセスが効果的にテストできることを確実にするためにスクリプトは同じであるべきです。URI、IPなどの環境にの違いは、構成管理プロセスの中で管理すべきです。*(参考文献: B, Chapter 6)*

29 / 50
新規ITサービスのリリース後に、運用のある業務が不意に終了してしまいました。

この原因として妥当では**ない**ものはどれでしょうか？

A) ゲートキーパーが作業項目とSACの相互関係を考慮に入れなかった。
B) ユーザー・ストーリーが不完全であったため、非機能要件が明確ではなかった。
C) サービスマスターが、リリースされるサービスのエンド・オブ・ライフ（EOL）についてユーザーと合意していなかった。
D) 開発プロセスにおいて、サービス受け入れ基準（SAC）でサポートされている通りに作業が行われなかった。

A) 不正解。これは原因として考えられます。
B) 不正解。これは原因として考えられます。
C) 正解。これは事前に判断できず、常にサービスが稼働してから判断する必要があるため、原因として考えられません。 *（参考文献：C、Chapter 7）*
D) 不正解。これは原因として考えられます。

30 / 50
AppBC企業ではDevOpsを使用しています。この企業は高度に自動化された受け入れテストで、継続的デプロイメントと信頼できるデプロイメント・パイプラインを導入しています。そして毎日新しいソフトウエアのデリバリーを稼働まで行っています。

AppBC社は大規模なデータベースと多数のユーザーを抱えています。彼らは、広範で信頼できるテスト戦略を導入しています。彼らの環境が極めて大きくまた複雑であるため、それぞれの新しいバージョンでバグが稼働中に発生します。

これらのバグを予防する**最適**な戦略はどれでしょうか？

A) カナリア・リリースを採用
B) 自動化されたキャパシティーテスト
C) デリバリー回数を減らす
D) ブルー・グリーン・デプロイメントを導入

A) 正解。カナリア・リリースは、素早いフィードバックを得るために一部の稼働環境のサーバーのために、アプリケーションの新しいバージョンを公開することが含まれます。レスポンスタイムや他のパフォーマンス指標について負荷を徐々に上げることで多くのユーザーに影響を与えず、新しいバージョンの問題を素早く発見します。そしてより早くバグを見つけて修復することができ、新しいバージョンのリリースのリスクを軽減します。 *（参考文献：B, Chapter 10）*
B) 不正解。この内容でキャパシティーテストはすでに自動化されています。しかしこのようなテストの自動化ではこのシナリオでバグを検知する手助けにはなりません。
C) 不正解。これはDevOpsに対抗する手法です。
D) 不正解。ブルー・グリーンはこのシナリオでは大変高額になることが予想される非常に多くのリソースを必要とします。また大きなDBでこの戦略を用いると、稼働停止やロールバックが起これば読み取り専用状態を引き起こします。またより良いキャパシティーテストを支援することはないでしょう。

31 / 50
DevOpsは、トヨタ生産方式に由来するアジャイルから非常に重要な概念を取り入れています。

DevOpsの採用に、一個流しが重要である理由は何でしょうか？

A) チームが予測可能なベロシティで、持続可能なペースで働けるようにする。
B) 最大のビジネス価値をもたらし得る機能を追加する作業に、チームが注力できるようになる。
C) さまざまなチーム・メンバー間での作業の共有責任が増える。
D) 同時に行うタスクの数を制限することで、ボトルネックが減る。

A) 不正解。これはリズムのメリットです。持続可能なリズムを設定することで、ペースを予測可能な状態に維持し、チームが燃え尽きることなく、従業員が適切なワークライフ・バランスを保てるよう保証できます。
B) 正解。これは一個流しの成果です。一個流しによって、最大の価値をもたらす機能を選択または更新し、パイプラインの次の段階に進めることができます。これにより、アジリティを維持できます。単一の機能に取り組むことで、仕掛（WIP）を制限できる傾向があるため、機能を実際に完成させることができます。 *（参考文献：C、Chapter 7）*
C) 不正解。これもDevOpsにとって重要ではありますが、仕掛（WIP）、一個流し、リズム、自工程完結とは直接関係ありません。
D) 不正解。これは仕掛（WIP）の説明です。チームが同時に取り組めるタスクの数を制限することで、実際に割り当てられたタスクを首尾よく完了できます。完了した作業を常に誰かが待っているよう、仕掛の制限を調整することで、ボトルネックを回避します。

32 / 50
中規模の自動車部品サプライヤのS社は大手のT自動車に部品を供給しています。S社がT自動車へ供給している部品は、S社の総売上のほぼ60%を占めます。

新たなパートナーシップを議論するための役員会が開催されました。T自動車は、S社にジャストインタイムでの供給方法に変更を要求しており、変更できないならばS社とのビジネスを停止しようとしています。S社はこのビジネスを失えば生き残れないため、ジャストインタイムに変更しなければならない危機感を持っています。この変更は６ヶ月以内に実施しなければならないため、最大でも5ヶ月しか準備期間はありません。

このために導入すべき対策の一つは、無線自動識別装置（RFID）を使用した部品追跡です。部品追跡は生産プロセスの透明性を保つのに役立つはずです。現在のプロセスをRFIDを用いたプロセスに変更することを進めるため、現在のプロセスを急いで見直す必要があります。

CIOは、変更プロセスを管理するように依頼されました。CIOは、DevOpsの手法で最小限のリリースを作成すればプロセス変更が可能であると信じています。理想としては、最初にRFIDを使用した生産方法の概念を作成しなければなりません。そして最後の段階で、RFIDデータを用いた生産管理システムを実装するべきです。しかし、これらのステップを順番通りに実行する充分な時間がありません。そのため、これらの作業を同時に行う必要があります。

CIOはこのプロジェクトのスクラムマスターであるEmさんを任命しました。開発チームはデプロイメント・パイプラインを構築する準備をします。

Emさんは、開発チームは熱意があり、懸命に働くとみていますが、もっと規律が必要だと考えています。さらに、リリースの頻度を上げる必要があります。

Emさんは **最初**に何に注目すべきでしょうか？

A) Emさんは、DevOpsでは最も大事な事柄である、コミュニケーションに注目すべきです。Emさんはチーム内のアイスブレークから始め、コミュニケーションのためのルールを設定すべきです。
B) 整流化されたプロセスと流れは大変重要ですので、Emさんはバリュー・ストリーム・マップと一個流しの仕組みをチームと議論することから始めるべきです。
C) DevOpsではツールや手法が正しく使用されて最も効果を発揮するので、Emさんはインフラストラクチャとチームメンバーの労働環境をチームと議論することから始めるべきです。
D) 組織文化の変更がDevOpsには必要ですから、Emさんは、関係者を集めて彼らにDevOpsを教育し、素晴らしい組織文化に変える支援を頼むことから始めるべきです。

A) 不正解。DevOpsチームを機能させるにはコミュニケーションが重要ではありますが、彼らは以前から一緒に働いていました。これは簡単に達成できる目標です。より重要なのは、顧客への価値の観点で考えるように思考を転換しなければならないことです。一度この考え方が定着すれば、Emさんはコミュニケーションスキルの微調整に取り掛かることができます。

B) 正解。プロセスは効率化が必要であり、バリュー・ストリーム・マップを作成しなければなりません。そうすればチームは可能な限り少ない努力でより多くの価値を付加し始めることができます。この後、変更できるもの、変更すべきもの、すでにうまく機能しているもの（ツール、コミュニケーション、組織文化など）を定義します。DevOpsはそれぞれの企業で同じように解釈される必要はありません。しかし、チームは顧客への価値を付加することを重視しなければなりません。*(参考文献: B, Chapter 1 and A, Chapter 1 and 2)*

C) 不正解。最初の段階で必要ではなくとも、ツールと自動化はDevOpsの重要な一部であり、忘れてはいけません。バリュー・ストリーム・マップを作成して、あなたが全ての生産やサービスプロセスをより簡便に、短く、安くさせる事が出来るのであれば、そしてチームとそのことについて議論できるのであればそれは良いことです。これにより、必然的に継続的な改善のサイクルが始まります。ほかの全てのステップは二次的に行われます。

D) 不正解。多くの企業では、組織文化の変更が必要です。しかし、必ずしも最初にやる必要があるわけではありません。組織文化の変更について関係者を集める必要もありません。顧客や関係者のための価値の大部分がどのように作られるかに従って組織文化を変更するべきです。異なる関係者にバリュー・ストリーム・マップを見てもらい、何か追加できるものはないか考えてもらうように依頼することは大変良い考えでしょう。

33 / 50
あなたのDevOpsチームは、持続可能なペースでうまく協力し合っています。チームはプロセスに十分な緩みを持たせることで、ビルドの入念なチェック／テストのための時間と集中力を確保しています。現在チームは、手動でテストと導入を行っています。ペースは速く、定期的にビジネスに高い価値をもたらしています。

CEOは、チーム内の自動化に関してあなたのアドバイスを求めてきました。

あなたが提供すべきアドバイスはどれですか？

A) チームがより多くの機能を追加し、早期にビジネス価値を実証できるよう、可能な限り自動化する
B) 手動プロセスのほうが安全であるため、受け入れテストは自動化するが、デプロイメントは自動化しない
C) サイクルタイムを改善するためデプロイメントを自動化するが、テストは自動化せず、バグから学べるようにする
D) チームが現在実行している方式は、素晴らしい成果を上げているため、このチームの方式に自動化は追加しない

A) 正解。プロセスの管理を向上させ、より多くのビジネス価値を早期に実証できるよう、可能なものはすべて自動化します。 *(参考文献：B、Chapter 1および8)*
B) 不正解。受け入れテストの自動化は素晴らしいアイディアですが、手動によるデプロイメントのほうが安全というわけではありません。
C) 不正解。リリースの自動化は素晴らしいアイディアですが、手動によるテストのほうが自動化されたテストよりもバグから学習できるわけではありません。
D) 不正解。チームは素晴らしい働きをしていますが、自動化できる活動を自動化していないことで、潜在能力を浪費しています。

34 / 50
CIOがスクラムマスターで最も信頼できる社員のマイケルにプロジェクトを任命しました。開発チームはデプロイメント・パイプラインの構築を準備しています。

マイケルは、開発チームの優れた自発性や心構えに自信を持っています。しかし彼らにもっと自律してほしいと望んでいます。加えて、リリース頻度をもっと高めるべきだと望んでいます。マイケルは開発チームにもっとリリース頻度を上げることを望みます。

あるチーム・メンバーが言いました。「この新しいデプロイメント・パイプラインが自動的に働くという事について、最も大事なことはまず自動化されたデプロイメント・パイプラインにすべきだ。」

この発言は正しいでしょうか?

A) はい、正しいです。自動で働くデプロイメント・パイプラインは、効率を上げるために最も重要な要素です。
B) はい、正しいです。自動化されているデプロイメント・パイプラインを作る際に注力することで、後で直面する潜在的な問題を克服します。
C) いいえ、正しくありません。一個流しを実施することと確かなデプロイメント・プロセスが最優先されるべきです。プロセスの自動化は後でできます。
D) いいえ、正しくありません。デプロイメント・パイプラインの自動化の代わりに、テストプロセスを最初に自動化すべきです。

A) 不正解。デプロイメント・パイプラインは常に一個流しのデプロイメント・パイプラインが最初にあるべきです。自動化されずとも、この仕組みが上手く働きます。一度実行されて確定したならば、可能なところからプロセスを自動化する機会があります。どんな状況でも自動化は、確固たるデプロイメント・パイプラインを構築するためには常に二次的であるべきです。
B) 不正解。デプロイメント・パイプラインは常に一個流しのデプロイメント・パイプラインが最初にあるべきです。自動化されずとも、この仕組みが上手く働きます。一度実行されて確定したならば、可能なところからプロセスを自動化する機会があります。どんな状況でも自動化は、確固たるデプロイメント・パイプラインを構築するためには常に二次的であるべきです。
C) 正解。デプロイメント・パイプラインは常に一個流しのデプロイメント・パイプラインが最初にあるべきです。自動化されずとも、この仕組みが上手く働きます。一度実行されて確定したならば、可能なところからプロセスを自動化する機会があります。どんな状況でも自動化は、確固たるデプロイメント・パイプラインを構築するためには常に二次的であるべきです。*(参考文献: B, Chapter 5)*
D) 不正解。自動化されたテストは主要な活動です。自動的に働くテストや確固たるデプロイメント・パイプラインを構築するための選択肢に直面しても、まず常に確固たるデプロイメント・パイプラインを構築することに最初に注力すべきです。一度実現できれば、テストの自動化をとおして効率を高める機会が生まれます。

35 / 50
あなたの会社は、DevOpsでの仕事に着手するため、方法を変更中です。あなたのチームはこの変更に加わっています。あなたはコードのコミットステージでのベスト・プラクティスを議論しています。

あなたの同僚であるサンが言いました。「ビルドが壊れてだれも責任を取らない時、我々は誰がやったのかを見つけ出すべきであり、ビルドを修復できるようみんなを招集すべきだ。」

この考えは良いでしょうか？

A) はい、ビルドを壊した人のみが修復できます。たとえみんなの気分を害することでも、あなたは誰が原因かを特定すべきです。
B) はい、あなたは常に、ビルドを壊した責任を取るべきです。あなたの同僚はたぶんこのルールを強要するでしょう。
C) いいえ、DevOpsでは、責任を問わない環境です。もし同僚が責任を取らなくても、彼らに強制しません。
D) いいえ、あなたはビルドの修復をまずやるべきです。そのうえで責任を取るべき人物を特定するために時間をかけ、その人を罰します。

A) 不正解。これは、たぶん問題を起こした人に問題を見つけさせる最も簡易な方法でが、必要ではありません。DevOpsでは責任を問わない環境です。もし同僚が責任を取らなくても、彼らに強制しません。誰かに何かをさせるという事は敬意を表していません。
B) 不正解。DevOpsとは責任を問わない環境です。もし同僚が責任を取らなくても、彼らに強制しません。誰かに何かをさせるという事は敬意を表していません。
C) 正解。誰かに何かをさせるという事は敬意を表していません。エラーを起こすのは問題ありません。協業の習慣で働くチーム・メンバーはいろいろなエラーや挑戦から互いに協力して働く方法が得られます。*(参考文献: B, Chapter 3 and A, Chapter 4)*
D) 不正解。ビルドは修復する必要はありません。あなたは前のバージョンに戻すことができます。付け加えればビルドを修復することは、悪い考えではないですが、ミスを犯した誰かを罰する事は悪い考えです。DevOpsでは責任を問わない環境です。もし仲間が責任を取らなくても、彼らに強制しません。誰かに何かをさせるという事は敬意を表していません。

36 / 50
XAppGo社の開発チームは、現在のテスト作業で多くの課題に直面しています。現在彼らは、手動での受け入れテストプロセスを使用しています。開発チームは、彼らが作成した単体テストセットがレグレッションを防ぐのに十分であると信じています。

開発チームはリリースごとの手動での受け入れテストに100万ドルを費やしています。上層部は開発チームに、自動化された受け入れテストを導入して、テストの全体費用を削減し、また、稼働環境に与えるレグレッションやコードの欠陥の件数を最小化することを指示しました。

自動化を考慮してアプリケーションの受け入れ基準を定義するときに、従わなければならない原則はどれでしょうか？

A) アジャイルの原則
B) ATAMの原則
C) INVESTの原則

A) 不正解。INVESTは、保守可能な受け入れテスト・スイートを作成中に適用する一連の推奨された原則です。何故ATAMやアジャイルが推奨していないか何も特定な情報はありません。アジャイルはテストの自動化について何も原則や特別なガイドを提示していません。
B) 不正解。INVESTは、保守可能な受け入れテスト・スイートを作成中に適用する一連の推奨された原則です。何故ATAMやアジャイルが推奨していないか何も特定な情報はありません。
C) 正解。受け入れテストは受け入れ基準に基づいており、あなたのアプリケーションのための受け入れ基準は自動化を意識して書かれていなければなりません。そして、独立している（independent)、交渉できる(negotiable)、価値がある(valuable)、見積もり可能(estimable)、小さい(small)、テスト可能(testable)の頭文字のINVESTの原則に従っていなければなりません。*(参考文献: B, Chapter 8)*

37 / 50
自動でデータを移行するための**最も**効果的な手段はどれでしょうか？

A) データベースのバージョニング・スキーマを作成し、バージョン管理下に置く
B) 移行が容易になるよう、小規模なデータセットを作成して管理する
C) データの移行前に、すべてのスクリプトが適切にテストされているか確認する
D) 移行の失敗に備えて、ロールバック手順を準備しておく

A) 正解。データベースのバージョニングは、自動でデータを移行するための最適な手段です。*(参考文献：B、Chapter 12)*
B) 不正解。これは、自動移行をサポートする最適なメカニズムではなく、効果的なデータベース管理の方法に重点を置いています。
C) 不正解。これは最適な回答ではありません。また、実際には移行ではなく、テスト作業に重点を置いています。
D) 不正解。これは、移行が失敗した場合に講じるべきリカバリ措置に重点を置いています。

38 / 50
X-AppGo社は、ロールバック・プロセスに問題を抱えています。このため、ロールバック・スクリプトの実行時に、本番アプリケーション・データベースにおいて重要なデータの損失が頻繁に発生しています。

重要なデータを失わずにロールバック・スクリプトを実行することが**できない**のは、どのような場合ですか？

A) ロールバック・スクリプトが、新しいバージョンで使用されているデータのみを削除する。
B) ロールバック・スクリプトに、テーブル間のカラム移動が含まれている。
C) ロールバック・スクリプトが、一時テーブルからのデータを追加する。

A) 不正解。ロールバック・スクリプトは、新しいバージョンで使用されているデータのみを削除し、ロールバック中に重大なデータの損失が発生することはありません。
B) 不正解。ロールバック・スクリプトは、データ損失を招かない方法でデータベース・スキーマを修正します。
C) 正解。このシナリオでは、ロールバック・スクリプトは不可能です。 *(参考文献：B、Chapter 12)*

39 / 50
ACMECONST社は、ルータとスイッチにアプリケーション・ソフトウエア・アップグレードおよびハードウエア更新を実施した後、多くのアプリケーション／ハードウエア障害に遭遇しています。

こうした障害はメンテナンス期間中に発生したため、元の状態への復旧は極めて難しくなっています。このため、リカバリ時間が延びて通常のメンテナンス期間を超過し、重要なアプリケーションのダウンタイムが拡大しています。

自動プロビジョニングと自律的インフラストラクチャはこの状況で役立ちますが、いくつかの検討事項が存在します。

本番環境への導入の際、停止のリスクを減らすために、入念な管理が必要なアイテムはどれでしょうか？

A) アプリケーション・アップグレードの不具合の問題解決に向けた詳細な監視ログ
B) 外部システムやサービスなどの、外部の統合ポイント
C) サーバー構成と、基盤となるユーザー・アカウント情報
D) 自動プロビジョニング・ツール一式と自律的アーキテクチャ

A) 不正解。問題解決はアップグレード後に行われるため、これは本番環境への導入リスクを減らす有効なアイテムではありません。
B) 正解。このアイテムは、本番運用に近いあらゆる環境への導入リスクを削減するために、入念に管理する必要があります。 *(参考文献：B、Chapter 11)*
C) 不正解。テスト環境と本番環境のいずれの場合も、これはオペレーティング・システムとその構成の情報でカバーされるため、個別に管理すべきアイテムではありません。
D) 不正解。リスクを削減するためにツール自体を管理することは重要ではありません。ツールは、自動プロビジョニングと自律的インフラストラクチャの実現に向けて、適切なプロセス実装とコンテキストを提供するものとして役立ちます。

40 / 50
X-AppGo社は、コア・アプリケーションに問題を抱えています。このアプリケーションは、他の外部アプリケーションと適切に連携できません。こうした外部アプリケーションは、特定のコールを実行できるよう、個々のデータ変数を効果的に取得する必要があります。コア・アプリケーションはあるチームによって開発中であり、同社は相応なビジネス上の理由により、このアプリケーションを維持したいと考えています。

開発者の1人が、インターフェイスの問題に対応するため、X-AppGoのコードベースからあるコンポーネントを分離することを提案してきました。

このケースでコンポーネントを分離する理由として、妥当なものはどれでしょうか？

A) コードベース内のプラグイン一式をモノリシックなコードベースに変換する
B) 変更の影響を制限し、コードベースの変更を容易にする
C) X-AppGoコードベースは、別のチームによって分離／管理される必要がある
D) 妥当な理由はなく、コードを分離するにはコンパイルに多くの時間がかかる

A) 不正解。コンポーネントの作成は、モノリシックなコードベースからモジュラー・コードベースへと、コンポーネントに基づいて進められるため、実際はこれは正反対です。
B) 正解。責任を明確に説明してソフトウエアを設計／維持することで、変更の影響を制限し、コードベースの理解と変更が容易になります。 *(参考文献：B、Chapter 13)*
C) 不正解。これはX-AppGoアプリケーションを独立したコンポーネントに分離することではないため、ビジネス上の理由を疑ったり、チームを分裂させたりする必要はありません。
D) 不正解。妥当な理由は存在し、その1つが選択肢2です。また、モジュラー・コードベースを作成すれば、コンパイルに必要な時間が減り、コードのリンクも少なく済みます。

41 / 50
最も小規模なアプリケーションであっても、他のコンポーネントやライブラリとの従属条件があります。したがって、従属条件を理解／管理することは、デプロイメント・パイプラインのフローを維持するために欠かせない、継続的デプロイメントの重要な活動です。

あなたは、2つのライブラリを使用するアプリケーションを構築しました。それぞれのライブラリは、基盤となる3つ目のライブラリに依存していますが、別々のバージョンを参照しています。これによって固有の従属条件が生じています。

この従属条件を解決または防止するための**最適な**方策はどれでしょうか？

A) すべてのライブラリを単一のライブラリにまとめ、直接そのライブラリを参照して問題を回避できるようにする
B) バージョン管理を用いることでライブラリを管理し、この種類の従属条件を作成したら直接表示できるようにする
C) 大きなボードに付箋を貼ってすべての従属条件の概要を視覚的に記し、フローを追跡できるようにする
D) ツールチェーンのごく一部のみをチェックインし、チェックイン時に問題が発生しても容易にデバッグできるようにする

A) 不正解。これは妥当なアイディアではありません。代わりに、バージョン管理を用いて最新のライブラリを参照するか、最新バージョンのライブラリの使用を自動化するための自動化ツールを使用します。
B) 正解。これは妥当な方策です。また、ライブラリのバージョン管理により、ユーザーが古いバージョンのソフトウエアを使用するという問題を回避できます。もう1つの妥当な方策として、最新バージョンのライブラリを使用するよう、自動化ツールを用いることもできます。 *（参考文献：B、Chapter 13）*
C) 不正解。これはフローの追跡には役立つかもしれませんが、従属条件に対応する方策ではありません。カンバン方式は、バージョン管理の導入の際にはあまり役に立ちません。
D) 不正解。代わりに、ツールチェーン全体をチェックインするべきです。これにより、相互の従属条件や非互換性を、より素早く確実に検出できます。

42 / 50
継続的デプロイメント環境においては、エラーを素早く検出したり、必要に応じてロールバックしたりできるよう、すべてをバージョン管理することが重要です。

しかし、バイナリー出力をバージョン管理することはお勧め**できません**。

この例外が生じる理由は何でしょうか？

A) バイナリー出力は非常に大きなファイルになる傾向がある。このファイルはビルドごとに変化し、自動的に更新される。
B) 複数のチーム・メンバーがバイナリーファイルに取り組むため、これをバージョン管理することは実際的ではない。
C) バイナリー出力はコンパイラ向けのインプットであり、すでにバージョン管理されている。
D) 通常のビルド・プロセスの一部として再コンパイルが実行されているため、これを行う必要がない。

A) 正解。まず、出力は非常に大きくなり、コンパイルされて自動テストに合格したチェックインごとに作り直されます。また、ビルド・スクリプトを再実行することで、ソース・コードから作り直される場合もあります。 *(参考文献：B、Chapter 2)*
B) 不正解。これは理由にはなりません。
C) 不正解。バイナリー出力はコンパイラのアウトプットであり、インプットではありません。他の部分は妥当です。
D) 不正解。通常のビルド・プロセスの一部として再コンパイルを行うことは賢明ではありません。再コンパイルが新しいバイナリー出力をもたらすことは確かです。

43 / 50
あなたは、ITインフラの全てを管理するために全体論的アプローチを採用したいと望んでいます。

2つの概念のうち、このアプローチはどちらを**最も**ベースにしているでしょう？

A) 1. インフラストラクチャの望ましい状態は、変更管理された構成を通して特定すべきである。
2. インフラストラクチャの実態の監視やイベント管理を通して常に知るべきである。

B) 1. インフラストラクチャの望ましい状態は、変更管理された構成を通して特定すべきである。
2. インフラストラクチャの実態を測定機器の使用やインシデント管理を通して常に知るべきである。

C) 1. インフラストラクチャの望ましい状態は、バージョン管理された構成を通して特定されるべきである。
2. インフラストラクチャの実態を、現在のインシデントやイベント管理を通して知るべきである。

D) 1. インフラストラクチャの望ましい状態は、バージョン管理された構成を通して特定されるべきである。
2. インフラストラクチャの実態を、測定機器の使用や監視を通して知るべきである。

A) 不正解。インフラストラクチャの望ましい状態は、変更管理された構成を通してではなくバージョン管理された構成を通して特定されるべきである。2も不正解：イベント管理は正解ではなく測定機器の原則を欠いている。

B) 不正解。インフラストラクチャの望ましい状態は、変更管理された構成を通してではなくバージョン管理された構成を通して特定されるべきである。2も不正解：インシデント管理は正解ではなく監視の原則を欠いている。

C) 不正解。１は正解。２は不正解：全体論的アプローチが**最も**ベースとなった原則の一つではない。

D) 正解：これらは、全てのインフラストラクチャを管理するための全体論的アプローチが作用する**最も**ベースとなった二つの原則です。 *(参考文献: B, Chapter 11)*

44 / 50
チームは優れた協業手法を採用し、作業チケットを同期させています。CTOは「現地現物」で、運用チームがいかに機能しているか把握しています。リリース後は、運用チームは常に運用インフラストラクチャを再評価しています。

この手法を改善するための**最適な**アドバイスはどれでしょうか？

A) 何もしない。再評価が常に行われているため、これ以上改善できない。

B) 運用環境のインフラストラクチャとアクセス・コントロールをモデル化する方法を模索する。

C) 運用インフラストラクチャが自動プロセスになるよう見直す。

D) デプロイメント・プロセスに関する知識を開発チームと共有する。

A) 不正解。これは不要な作業であり、改善できます。

B) 不正解。一見素晴らしく思えますが、何度も繰り返されるため、ムダが生じます。

C) 不正解。開発を関与させずに、このプロセスの自動化に着手する明確な方法はありません。

D) 正解。これが最善の方法です。知識を共有し、その後、さらなる対策を講じます。 *(参考文献：A、Chapter 17)*

45 / 50
運用の変更を開発へ通知するために運用側にとって良い時期はいつでしょうか？

A) 開発は通知を受ける必要がない。運用の変更は運用チームのみの為である。
B) 即座に。開発へは可能な限り通知すべきである。
C) 翌日のスクラム・オブ・スクラム・ミーテイングで。
D) 運用チームが受け入れテストを完了した時点で。

A) 不正解。開発へは即座に通知すべきです。それでリスクや問題の可能性を予見できます。
B) 正解。開発へは即座に通知すべきです。それでリスクや問題の可能性を予見できます。 *(参考文献: C, Chapter 5 and 7)*
C) 不正解。開発へは即座に通知すべきです。それでリスクや問題の可能性を予見できます。
D) 不正解。開発へは即座に通知すべきです。それでリスクや問題の可能性を予見できます。

46 / 50
あなたはあなたのDevOps組織が成熟することを望んでいます。そのためにはいろいろな方法があります。

DevOps組織の成熟を支援 **しない**方法は何でしょうか？

A) もしも、日々の活動が価値あるものであるならば、あなたのチームメンバーの判断を助けるマイルストーンのような目標を明確に定義する。
B) プロセスの定義を明確にし、支援し、チームメンバーが日々プロセスを改善することを可能にする。
C) 全ての会議の記録を残す。それであなたのチームメンバーは全てのコミュニケーションへ容易に参加できる。
D) 日々の進捗の小さな部分を識別することを助け、彼らを称賛する為に日々の活動を監視し記録する。

A) 不正解。これはDevOps組織の成熟化を支援するために手助けになります。
B) 不正解。これはDevOps組織の成熟化を支援するために手助けになります。
C) 正解。これはDevOps組織の成熟化を支援しません。会議での全ての記録を書き出し、それを再検討するという必要はありません。合意内容を書き出す必要はあります。しかし、全ての会議ではありません。 *(参考文献: B, Chapter 15)*
D) 不正解。これはDevOps組織の成熟化を支援するために手助けになります。

47 / 50
あなたはITサービスプロバイダで働いており、事業継続計画の一部として、最低限の合意されたサービスレベルを常に満たしていることを保証したいと考えています。

あなたはITサービスの継続性を確実にしたいと望んでいます。

DevOpsはITサービス継続性管理においてどのように役立ちますか？

A) DevOpsの文化的価値である密接な関係や協業は、DevOpsチームメンバーによってサービスが高い価値をもたらすことを確実にします。
B) DevOpsはシステムに障害を意図的に取り込むことによって、チームのディザスタ・ルーティンと大部屋の実践を準備します。
C) 運用は開発と共に働くから、リスク削減手法とリカバリー・オプションは、コーディングされていることが望ましい。
D) サービスレベル管理はDevOpsではより重要になる。なぜならプロセスマスタの仕事はこれを監視することだから。

A) 不正解。文化的な価値は失敗から学び、人々を安定したペースで気分良く、仕事をやりやすくすることを手助けする。これ自体サービスレベル管理を支援しない。
B) 不正解。NetflixのChaos Monkey(サービス障害を起こし続けるツール）のような混乱の取り込みは、リスク削減手法やリカバリー・オプションのコーディングを開始する助けにはなるでしょう。しかし、それ自体はサービスレベル管理を支援しません。システムの混乱はコードで解決すべきであり、ウォールーム(プロジェクトルーム）形式の解決策によってではありません。
C) 正解。これは、インフラストラクチャとアプリケーションリスクを防止、予測、管理するための体系的なプロセスであり、それに対処しなければ、サービスを混乱させる可能性のある混乱や、インシデントにつながる可能性があります。 *(参考文献：B, Chapter 11 and C, Chapter 4)*
D) 不正解。プロセスマスタは、サービスレベル管理を維持するために優先されたタスクを持ちません。最低限これはどのDevOpsチームでも優先付けはしません。

48 / 50
ACMECONST社は、世界中で社員とエンジニアリングチームの数を増やすことによって世界規模での存在感を積極的に拡大しています。また、同社は年間30%の飛躍的なペースで顧客基盤を増やしています。

かつてエンジニアリングチームが1つの部屋にあった時には決定をすることは容易でしたが、今は長い時間がかかり、組織中でフラストレーションが起きています。管理承認までにより多くの段階が設けられ、プロセスの範囲は拡大し、そのためエンジニアの多くが、意思決定プロセス全体に幻滅を感じています。

提示される様々な問題のオーナーシップについても混乱が拡大しており、意志決定において躊躇を引き起こします。また、エンジニアは、追加のプロセスや官僚主義によって自分たちの創造性が抑え込まれていると感じており、エンジニアたちの士気に影響が出始めました。

このシナリオに対処する**最良**の方法は何ですか?

A) 現在のプロセスを維持するが、明確化された役割、説明責任、各プロセスのオーナーシップを確立する。リスク対生産性に重点を置くために効果的な手法を確立する。徐々に変更を加えていき、新手法の安全領域を作る。
B) 物事が整流化でき、明確な役割、説明責任、各プロセスのオーナーシップを識別するためにプロセスを再検討する。リスク対生産性に重点を置くために効果的な手法を確立する。徐々に変更を加えていき、新手法の安全領域を作る。
C) 物事が整流化でき、明確な役割、説明責任、各プロセスのオーナーシップを識別するためにプロセスを再検討する。リスク対生産性に重点を置くために効果的な手法を確立し、徐々に変更を加え、不必要なアプリケーションの失敗を阻止する手法の試みを最小限にする。

A) 不正解。現在のプロセスを維持する最高のアイデアではありません。組織は拡大しているため、現在のプロセスはもう機能していません。しかし、実験のための安全な場所の作成は良いアイデアです。
B) 正解。このような行動は、成長している会社で効果を発揮します。古いプロセスはもう機能してないため、プロセスの再検討が必要です。実験用の安全な場所も、継続的な改善を促進するために必要です。*(参考文献: A, Chapter 15)*
C) 不正解。プロセスの再検討が必要です。しかし、実験の回数を最小限に抑えることは悪い考えです。これは実験を許可しないことにより継続的な改善を阻害します。

49 / 50

X-AppGo社では、優先順位と目標が異なっていることが原因となり、コロンビアの運用チームとアイルランドの開発チームで衝突が起こっています。この対立で、ビジネスに影響する課題の解決にかかる時間と工数が増加しています。

開発と運用チーム間の衝突を減らし、協業を高めるため、X-AppGo社はどの主要な手法を考慮すべきでしょうか？

A)
1. もし双方が対立を回避したいと望むならば、開発と運用のチームが互いに別々に仕事をすることを許容する。
2. 開発と運用のチームを支援するうえで役員会が全てを握る。

B)
1. 共に働く重要性について DevOps チームと会話するために会社の役員会からスポンサーを得る。
2. 開発と運用のチームに DevOps のトレーニングを行う。そのうえで彼らが互いの仕事を学ぶ。

C)
1. 開発と運用のチームが DevOps が上手く働いている他の企業を訪問して確認させる。
2. 運用と開発のチームが直面している要求の増大をより良くサポートするために資金を増加させる。

D)
1. 調和した関係を築き、信用を生み、相互理解できるよう、開発チームと運用チーム間で、互いのサイト訪問を推奨する。
2. 開発と運用のチーム間で知識が広まり互いにより効率的に働ける

A) 不正解。チームが互いに励ましあって働くように仕向けるべき。別々に働くことを許容するべきではありません。彼らは互いに必要であり、互いから学ぶ必要があります。協業は、互いに顔を合わせなければ起こりません。役員会からの支援を得ることは、チームが互いに働く手助けにはなりません。

B) 不正解。役員会からのスポンサーによってより感謝を感じるかもしれませんが、互いに協業する助けにはなりません。今のところはトレーニングのみは有効です。チームが互いにより良く働く手助けのために、彼らは互いの仕事をよく知る必要があります。

C) 不正解。他の企業を訪問することで、やる気を起こさせることはできます。しかし、DevOpsは企業の中で非常に特有なものです。対立を解消し協業を直接的に支援するものではありません。チームは共に働くべきであり、知識を共有すべきです。資金を増やすことは、たくさんの仕事を少ない人数で行う際にはわずかな助けにはなります。しかし今のところ対立を減らし協業を増やすだけです。

D) 正解。運用と開発のチーム間での紛争対立を解消し減らし協業を促進する現在のシナリオを最高に支援するための最も好ましい手法である。 *(参考文献: A, Chapter 15)*

50 / 50
開発チームはDevOpsに興味を持っています。彼らは特に継続的インテグレーション（CI）に興味を抱いてました。彼らは現在3つの主要なソリューションと4つの小さめのソリューションの開発と保守を担当しています。彼らはスクラム手法を用いています。それぞれのスプリント（イテレーション）は4週間で、平均的に1つのコミットされたリリースからテスト環境までにそれぞれ10～15日かかります。そして1つのリリースから稼働まで1か月かかります。彼らは、CIを作成する努力や彼らの投資を支える管理のために定性的なビジネス・ケースを作りたいと望んでいます。.

このビジネス・ケースを**最も**支援するCIの具体的な利益（メリット）はどれでしょうか？

A) 1日に一回デプロイからテスト環境まで実行できればビジネス的メリットの増加と大幅な開発コストの削減ができる。
B) チームの精神を支援する。すでに彼らはスクラムを用いており、CIはビジネスの目に見える利益を**生まない**。
C) より良い自動化されたテスティング、全体的なリリーススピードの向上でリリースの安定性と品質を高める。
D) 1日に一回のリリースから稼働までの実行は、ビジネスの利益（メリット）を増大させ、大幅な開発コストの削減ができる。

A) 不正解。テスト環境へ素早くデプロイするのは良いし CI の結果であるが、ビジネス上のメリットは作成されません。
B) 不正解。スクラムを使用するかどうかは関係なく、CIは、少ないコストでバグを素早く見つけたり稼働までの早いデリバリーを支援することができます。
C) 正解。各々の変更は機能しメインコードに統合され、稼働への準備ができており、プロダクトが常に稼働する状態になるため、リリーススピードの向上は、継続的インテグレーションの結果となる。また自動化されたテスティングによってバグを速やかに見つけて修正しリリースの安定性や品質を高める。*（参考文献：B, Chapter 3）*
D) 不正解。より早いリリースからの稼働は継続的デリバリの主要な効果の一つだが、リリースが稼働できる状態で自動化されたテストの通過を確実にするため稼働環境に似せた環境へのデリバリを目的とした継続的インテグレーションの直接的なメリットではない。

評価

次の表に、本模擬試験問題の正解を示します。

番号	正解	番号	正解
1	B	26	D
2	D	27	A
3	A	28	D
4	B	29	C
5	B	30	A
6	B	31	B
7	C	32	B
8	C	33	A
9	A	34	C
10	B	35	C
11	B	36	C
12	A	37	A
13	D	38	C
14	D	39	B
15	A	40	B
16	D	41	B
17	A	42	A
18	B	43	D
19	C	44	D
20	B	45	B
21	A	46	C
22	D	47	C
23	D	48	B
24	A	49	D
25	C	50	C

EXIN の連絡先

www.exin.com

YOUR ICT COMPETENCE PARTNER