システム開発の基礎スキル

大分類	中分類	小分類	番号	問題文	選択肢1	選択肢2	選択肢3	選択肢4	正解	解説
開発プロセスとチーム開発の基礎	アジャイル開発の基本と実践	アジャイル開発の特徴とウォーターフォールとの違い	1	アジャイル開発が重視する価値観にはどのようなものがありますか。	厳密な文書化と進捗報告の徹底	プロセスとツールによる制御	顧客との対話と変化への対応	要件の完全な事前定義	3	アジャイル開発は「プロセスやツールよりも個人と対話を」「動くソフトウェア」「顧客との協調」「変化への対応」など、柔軟かつ実用的な価値観を重視します。文書や計画よりも実際の動作や対話を大切にします。
			2	ウォーターフォール開発と比較した場合の、アジャイル開発の特徴は何ですか。	各工程を順に一度だけ実施する	顧客とのやり取りは開発前に限定される	要件を最初に完全に確定する	短いサイクルで開発と改善を繰り返す	4	アジャイル開発では「イテレーション（反復）」という短い開発サイクルを回しながら、段階的に改善・機能追加を行います。ウォーターフォールとは異なり、後からの変更に柔軟に対応できます。ウォーターフォールは階段を降りるように戻れないイメージ、アジャイルは小さな円を何度も描いて進むイメージです。
			3	要件変更が発生した場合、アジャイル開発ではどのように対応しますか。	原則として変更を受け付けない	プロジェクト終了後に反映させる	スプリント終了時のレビューで対応を検討する	上長の承認後にのみ変更可能とする	3	アジャイル開発では、変更が自然に起こるものとして受け入れ、次のスプリントで反映できるようにバックログを調整します。柔軟性が特徴です。
			4	アジャイル開発が向いているプロジェクトの特徴を挙げてください。	仕様が明確で変更の少ない大規模プロジェクト	要件が流動的で変更が予想されるプロジェクト	納期と内容が完全に固定されたプロジェクト	官公庁向けの厳格な文書管理が必要なプロジェクト	2	アジャイルは、要件が変化することが前提の環境に適しています。素早く対応しながら、段階的に成果を出すのに向いています。
			5	アジャイル開発とウォーターフォール開発を使い分ける際の判断基準は何ですか。	要件の明確さと変更の可能性	使用するプログラミング言語	開発メンバーの人数	サーバーの運用期間	1	要件が初めから明確で変更が少ない場合はウォーターフォールが向き、変更が予想される場合はアジャイルが適しています。
			6	短い開発サイクルを回すことにはどのようなメリットがありますか。	成果がまとめて得られ、品質も保証される	テストを省略してリリースできる	ドキュメント作成の手間を省ける	顧客からのフィードバックを早期に得られる	4	アジャイルでは「短いサイクル＝イテレーション」を繰り返すことで、顧客が途中経過を確認でき、修正や改善のフィードバックが得やすくなります。
		スクラム開発の基本要素	7	スクラム開発における「スプリント」の役割はどれですか。	開発チームが休息を取る期間	チームメンバーの評価を行う期間	一定期間で開発とレビューを繰り返す単位	要件定義を行う期間	3	スプリントは通常1〜4週間程度の短い期間で、設計・実装・テスト・レビューまでを繰り返す開発単位です。スクラムの中核となるサイクルです。
			8	スクラムにおける「プロダクトオーナー」の主な役割はどれですか。	チームのスケジュールを管理する	プロジェクトのテストを実施する	開発タスクの技術的な指示を出す	製品の価値を最大化するためにバックログを管理する	4	プロダクトオーナーは、プロダクトバックログの管理と優先順位付けを通じて、プロダクトの価値を最大化する責任を負います。
			9	スクラムにおける「デイリースクラム」とは何を行うミーティングですか。	進捗を確認し、障害を共有する短時間の定例ミーティング	顧客との成果物レビューを行う会議	全社的な開発方針を議論する会議	過去のプロジェクトを振り返る反省会	1	デイリースクラムは毎日行う15分程度の定例会議で、チームメンバーが進捗を共有し、障害を明らかにします。
			10	スクラムにおける「スプリントレビュー」の目的はどれですか。	チームメンバーの評価を行う	スプリントの失敗要因を分析する	成果物を関係者に見せ、フィードバックを得る	次回スプリントのバックログを決定する	3	スプリントレビューでは、スプリントで完成した成果物を関係者に提示し、フィードバックを受けて次の改善に活かします。
			11	スクラムマスターの役割として適切なものはどれですか。	スクラムの実践を促し、チームの障害を取り除く	開発チームに直接的な指示を出す	チームの進捗を評価し、給与を決定する	すべての仕様書を作成する責任を持つ	1	スクラムマスターは、チームが自律的にスクラムを運用できるよう支援する役割で、障害の除去やファシリテーションが主な任務です。
			12	スクラムにおける「スプリントレトロスペクティブ」の目的として最も適切なものはどれでしょう。	スプリントでの成果物の品質をユーザーが評価する	チームの働きぶりを評価し、メンバーを選別する	チーム全体でプロセスを振り返り、改善点を見つける	次回スプリントのストーリーポイントを算出する	3	スプリントレトロスペクティブは、スプリントの最後に行われる振り返りの場であり、チーム内の働き方やプロセスを見直し、改善点を洗い出すことを目的としています。
		カンバン方式と進捗管理の工夫	13	カンバン方式の基本的な考え方を説明してください。	タスクの優先度に関係なく一斉に処理を進める方法	時間ごとにすべての作業を見直す方式	タスクの状態を可視化し、フローを最適化する方式	チームリーダーがすべてのタスクを手配する方式	3	カンバン方式は、タスクの進行状況を「見える化」し、作業の流れ（フロー）を最適化して、作業の停滞を減らすことを目的としています。
			14	カンバンボードを使うことで得られる効果は何ですか。	タスクごとの作業時間が自動で測定される	作業の進捗状況をリアルタイムで共有できる	作業者のミスを自動的に修正できる	スプリントの予定を正確に決定できる	2	カンバンボードにより、タスクの「To Do」「Doing」「Done」などの状態が可視化され、チーム全員が進捗を把握しやすくなります。
			15	WIP（仕掛かり作業）制限とは何ですか。	作業ごとの予算を制限すること	一度に担当できる作業数に制限を設けること	優先度の低い作業を除外すること	作業の期限を厳密に固定すること	2	WIP制限は、同時に進行するタスク数を制限することで、作業の集中と流れの改善を図る手法です。多すぎる作業の同時進行を防ぐ効果があります。
			16	タスクの「見える化」がチームに与える影響を説明してください。	作業の属人化が進みやすくなる	チーム間の競争を煽る効果がある	各人の作業状況が共有され、助け合いや調整がしやすくなる	個人の評価が明確になり人事考課に有利となる	3	タスクを「見える化」することで、誰がどの作業を行っているかが明確になり、作業の分担や支援がしやすくなります。
			17	カンバン方式が向いている開発現場の特徴は何ですか。	タスク内容がほぼ変わらず、繰り返し作業が多い現場	全体の作業を長期計画で一括管理する現場	ウォーターフォール型で一斉納品を行うプロジェクト	成果物が目に見えない研究開発プロジェクト	1	カンバン方式は、ヘルプデスクや保守業務など、定常的で流動的なタスクが多い現場に適しており、作業の流れの最適化に役立ちます。
			18	進捗管理において「滞留中のタスク」を把握することが重要な理由は何ですか。	タスクの見積精度を上げるため	責任者の交代時に便利だから	チームのモチベーションを維持するため	作業の流れが滞っている部分を発見しやすくなるから	4	滞留しているタスクを見つけることで、ボトルネック（作業の詰まり）を把握でき、改善すべきポイントが明確になります。
			19	カンバン方式とスクラムを併用する場合の注意点は何ですか。	タスク数を2倍に増やす必要がある	タイムボックスを無視して進めることが望ましい	スプリントを廃止して、フローを完全に自由にすることが重要である	両方の原則のバランスを取り、目的に応じて使い分ける必要がある	4	スクラムとカンバンを併用する場合、それぞれの特性（時間ベース vs フローベース）を理解し、プロジェクトに応じて適切に使い分けることが重要です。
	プロジェクト管理とチームでの役割	開発工程とドキュメントの流れ	20	システム開発における一般的な工程の流れを説明してください。	実装 → 運用 → 設計 → テスト	要件定義 → 設計 → 実装 → テスト → 運用	設計 → 要件定義 → 実装 → テスト → 展開	実装 → 設計 → テスト → 検収 → 運用	2	一般的なシステム開発は「要件定義 → 設計 → 実装 → テスト → 運用」の順で進みます。前工程の成果物が次工程の基礎となります。
			21	要件定義書にはどのような内容が記載されますか。	プログラムの命令文と変数の定義	テストケースと実行結果	システムに求められる機能や制約、利用者のニーズ	画面のレイアウトと配色設計のみ	3	要件定義書は、システムに求める機能・性能・制約条件などを明記し、顧客と合意を形成するための文書です。
			22	設計書にはどのような役割がありますか。	実装やテストの土台となる仕様を明文化する	システム開発の費用を試算する文書である	システムの利用者に向けたマニュアルとして使う	開発後に捨てても問題ない一時的な文書である	1	設計書には、システムの構成、画面遷移、データ構造、処理ロジックなど、開発やテストの基盤となる情報が記載されます。
			23	上流工程の品質が後工程に与える影響は何ですか。	特に影響はない	上流工程の品質が悪くても後で修正できる	下流工程での工数削減につながる	品質が低いと下流での手戻りや修正コストが大きくなる	4	上流工程（要件定義・設計）のミスは後工程での大きな手戻りにつながり、修正コストが高くなるため注意が必要です。
			24	ドキュメントを残すことのメリットを挙げてください。	すぐに削除できるから	プロジェクトのメンバー変更時に知識を引き継げる	書いた人しか理解できない情報を残せる	毎回ゼロから開発ができるようになる	2	ドキュメントがあると、メンバー交代時にも業務や開発内容をスムーズに引き継げるなど、ナレッジの共有が可能になります。
			25	開発工程ごとに成果物を確認することが重要な理由は何ですか。	テスト工程の数を減らすため	作業の担当者を増やすため	各工程の問題点を早期に発見できるから	納期を後ろ倒しにするため	3	工程ごとに成果物をレビューすることで、後工程に影響する問題を早期に発見・修正できます。
		チーム開発におけるコミュニケーションスキル	26	チーム開発において報連相が重要とされる理由は何ですか。	上司への忠誠心を示すため	エンジニアの評価基準になるから	他人に仕事を任せないようにするため	情報共有と意思疎通を円滑にするため	4	報告・連絡・相談は、チーム内の状況共有とスムーズな意思決定に不可欠なコミュニケーションの基本です。
			27	朝会で共有すべき情報にはどのようなものがありますか。	仕事以外の個人的な話題	昨日やったこと、今日やること、困っていること	昼食の予定と雑談	チームの成績と売上情報	2	朝会では、各メンバーが進捗状況を簡潔に共有することで、問題の早期発見や作業の調整が可能になります。
			28	チケット管理ツールを使う目的は何ですか。	作業状況の透明性を高めること	メンバーの人事評価に使うため	作業量を記録して罰則を与えるため	顧客との契約書を管理するため	1	チケット管理ツールを使うことで、各タスクの状態、担当者、期日などが可視化され、チームの作業が管理しやすくなります。
			29	認識のズレが発生しやすい場面と、その対策を説明してください。	メールの宛先を間違えたとき／後で謝る	仕様の共有時／言葉だけでなく図や例を使って説明する	開発環境の設定時／相手の作業を止める	雑談中／話題を変える	2	仕様や要件を言葉だけで共有すると誤解が生じやすく、図・プロトタイプ・具体例を使った説明が有効です。
			30	チーム内での情報共有が不足するとどのような問題が起こりますか。	作業時間が短縮される	意図しない重複作業や手戻りが発生する	コミュニケーションコストが下がる	チームの結束力が強まる	2	情報共有が不十分だと、重複作業、仕様の誤解、納期遅れなどが発生するリスクが高まります。
			31	非エンジニアとのコミュニケーションで意識すべき点は何ですか。	専門用語を多く使ってプロフェッショナルさを示す	相手の業務内容には立ち入らない	相手の立場や知識に配慮し、分かりやすく説明する	できるだけ自動応答に頼る	3	非エンジニアとのやり取りでは、専門用語を避け、図や比喩を使いながら相手に合わせて説明することが大切です。
		プロジェクトの進行管理とリスク対応	32	プロジェクト管理における「進捗管理」とは何ですか。	プロジェクトの品質を評価すること	プロジェクトの関係者を選定すること	成果物をリリースする手順を定めること	作業の進み具合を把握し、計画との差を確認すること	4	進捗管理は、プロジェクトが予定通りに進んでいるかを把握し、遅れがあれば原因を分析し対策を講じるための重要な管理項目です。
			33	マイルストーンを設定する目的は何ですか。	チームメンバーの報酬を決めるため	作業の成果を分割して評価するため	プロジェクトを段階的に管理し、進行の節目を明確にするため	不要な工程をスキップするため	3	マイルストーンはプロジェクトの重要な節目であり、進捗確認や意思決定のタイミングとして活用されます。
			34	プロジェクトにおけるリスクとは何を指しますか。	プロジェクトの進行や成果に悪影響を与える可能性のある要因	発生すると確実に成果が上がる出来事	成功が確定している要素	プロジェクトに関係のない業務上の雑務	1	リスクとは、納期遅延や予算超過、人的リソース不足など、プロジェクトに悪影響を及ぼす可能性のある事象全般を指します。

			35	リスクを事前に洗い出すことの重要性を説明してください。	事前に対策を考えることで、被害を未然に防げるから	発生後にすぐ対処することで十分だから	計画の精度が下がっても問題ないから	プロジェクト終了後に対応する方が効率的だから	1	リスクを事前に把握することで、代替案の準備や優先順位の調整が可能となり、トラブル発生時の対応を迅速に行えます。
			36	想定外のトラブルが発生した場合、どのような対応が求められますか。	そのまま無視して計画通り進める	トラブルの責任者をすぐに特定して処分する	計画をすべて最初から作り直す	状況を冷静に把握し、関係者と連携して柔軟に対応する	4	トラブル時には、まず状況を正確に把握し、関係者と情報を共有した上で柔軟かつ迅速に対応することが求められます。
			37	スケジュール遅延を防ぐために有効な工夫は何ですか。	すべての作業を最後にまとめて行う	作業を個人の裁量に任せる	余裕を持ったスケジューリングと進捗の定期確認を行う	毎日計画を変えるようにする	3	余裕を持ったスケジュールと進捗確認は、リスクの早期発見と調整を可能にし、遅延の回避に効果的です。
			38	プロジェクトの振り返りを行うことの意義は何ですか。	関係者の責任を明確にするため	成果を否定的に評価するため	今後のプロジェクトに活かす学びを得るため	失敗したことをチーム内で共有しないようにするため	3	振り返りは、うまくいった点や改善点を整理し、次回のプロジェクトに生かすための重要なプロセスです。
開発実務に必要な技術スキル	バージョン管理（Git）の基本操作	バージョン管理の目的とGitの基本操作	39	バージョン管理を行う目的は何ですか。	ソースコードの変更履歴を記録・管理し、変更の追跡や復元を可能にするため	開発チームを縮小するため	ファイルの名前を統一するため	ソースコードを暗号化するため	1	バージョン管理の主な目的は、変更履歴の追跡・比較・復元を可能にし、共同作業の整合性を保つことにあります。
			40	Gitを使うことでどのような問題を防ぐことができますか。	コードの実行速度の低下	変更履歴の消失や他人の変更との衝突（コンフリクト）	データベースの肥大化	ネットワークの接続トラブル	2	Gitは変更履歴を記録し、複数人が同時に作業する際の衝突や誤操作によるデータの喪失を防ぎます。
			41	リポジトリとは何ですか。	ソースコードや履歴情報を含むプロジェクトの保管場所	プログラムの実行ファイルを保管する場所	データベースのログ記録ファイル	ネットワーク上のサーバー設定情報	1	Gitでは、リポジトリとはプロジェクトのソースコードやその変更履歴などを記録する保管場所を指します。
			42	commit操作の役割を説明してください。	コードを削除する操作	コードを自動的にテストする操作	ブランチを削除する操作	現在の変更内容を記録し、履歴として保存する操作	4	Gitのcommitは、作業ディレクトリの変更をローカルリポジトリに履歴として保存する操作です。
			43	pushとpullの違いは何ですか。	pushはローカルからリモートへの送信、pullはその逆	pushは削除操作、pullは復元操作	pushは設定変更、pullはブランチ作成	両方ともテストの自動化操作	1	pushはローカルの変更をリモートに反映する操作で、pullはリモートの変更をローカルに取り込む操作です。
			44	ローカルリポジトリとリモートリポジトリの違いは何ですか。	ローカルは他人と共有できないが、リモートはできる	ローカルは有料、リモートは無料である	ローカルはテスト専用、リモートは本番環境	両者に違いはない	1	ローカルリポジトリは自分のPC内の変更履歴、リモートリポジトリはチーム全体で共有する履歴です。
		ブランチ運用とプルリクエスト	45	ブランチを分けて開発する理由は何ですか。	ソースコードを削除しやすくするため	異なる作業を並行して進め、他の作業に影響を与えずに開発できるため	サーバーの負荷を減らすため	バージョン管理を停止するため	2	ブランチを使えば、機能ごとに独立して開発ができるため、他の作業との干渉を避けられます。
			46	main（またはmaster）ブランチの役割を説明してください。	テスト専用のブランチ	エラーをまとめて表示するブランチ	安定したリリース用のコードを管理する代表的なブランチ	各開発者の個人用ブランチ	3	main（またはmaster）ブランチは、通常リリースされるコードの管理に用いられ、安定した状態が維持されます。
			47	プルリクエストの目的は何ですか。	ファイルをダウンロードするための機能	他人のブランチを削除するための操作	自動テストをスキップするための仕組み	自分の変更をチームにレビューしてもらい、mainブランチへのマージを依頼するため	4	プルリクエストは、ブランチでの作業内容を共有し、レビューや承認を経てmainにマージするプロセスです。
			48	コードレビューを行うことのメリットは何ですか。	開発スピードを落とすこと	作業内容を全員で再度実装すること	バグや設計ミスを事前に発見し、品質を高めること	個人の責任を明確にすること	3	コードレビューにより、実装の品質やチーム内の共通認識が高まり、バグの早期発見にもつながります。
			49	機能ごとにブランチを作成する運用の利点は何ですか。	プロジェクトが単純になる	同時並行での開発が可能になり、マージの際に差分が明確になる	テストが不要になる	開発者が1人で作業できるようになる	2	機能単位のブランチは、変更の単位が明確になり、レビューやマージ時の把握がしやすくなります。
			50	マージ前に確認すべきポイントにはどのようなものがありますか。	コードの文字数と装飾の有無	ファイルの数とサイズ	コンフリクトの有無、テストの通過、レビュー内容	使用言語のバージョン	3	マージ前には、他のブランチとの競合（コンフリクト）の有無やテスト結果、レビュー内容などを確認する必要があります。
		コンフリクトの理解と解決方法	51	コンフリクトとはどのような状態を指しますか。	インターネット接続が切れた状態	異なる開発環境での実行結果の違い	同じファイルの同じ箇所に対して複数人が変更を行い、Gitが自動で統合できない状態	プログラムのバグが修正できない状態	3	Gitのコンフリクトは、同じ箇所の競合によって、Gitが自動でマージできなくなったときに発生します。
			52	コンフリクトが発生する主な原因は何ですか。	リモートリポジトリの容量不足	同一ファイルに複数人が同時に異なる変更を加えた場合	テストケースの実行漏れ	デバッグログの出力ミス	2	複数の変更が同一ファイルの同じ箇所にあった場合、Gitはどちらを優先すべきか判断できず、コンフリクトが起こります。
			53	コンフリクトを解消する際に注意すべき点は何ですか。	変更箇所を全て削除すること	他人の変更を無視すること	両方の変更内容を確認し、意図に合った形で統合すること	コミット履歴をすべてリセットすること	3	どちらの変更も確認し、必要な部分を残しながら一貫性のある内容に統合するのが適切です。
			54	コンフリクトを未然に防ぐための工夫を挙げてください。	常に最新の状態をpullし、作業前に同期を取る	リモートを使わずローカルで完結させる	一人で作業する	コンフリクトは必ず発生するので、事前の工夫は無意味である	1	作業前にpullしてローカルとリモートの状態を同期させておくことで、差分によるコンフリクトを避けやすくなります。
			55	チーム開発において頻繁にpullを行うことの意義は何ですか。	最新の開発状況を常に反映し、他メンバーとの差異を小さく保てる	ファイルサイズが削減される	リモートのブランチを削除できる	コードを自動的に圧縮できる	1	pullを頻繁に行うことで、自分の作業環境を最新に保ち、コンフリクトを回避しやすくなります。
			56	コンフリクト解消後に必ず行うべき確認作業は何ですか。	関係ないファイルの削除	編集ソフトの再起動	コードが意図した通り動作するかのテスト	ブランチの名前の変更	3	手動で修正した後は、意図しない動作がないかをテストで確認する必要があります。
			57	コンフリクト対応時にチーム内で共有すべき情報は何ですか。	誰がミスをしたか	エラーの回数	どのファイルのどの部分に競合が発生したか、解消内容の方針	個人の作業ログの全記録	3	どこで何が競合したか、どのように対応したかをチームで共有することで、再発防止や認識合わせに役立ちます。
	テストと品質管理の考え方	単体テストと結合テストの違い	58	単体テストの目的は何ですか。	プロジェクトの進捗を管理すること	システム全体の動作を確認すること	ネットワーク接続を確認すること	個々のモジュールや関数が正しく動作するかを検証すること	4	単体テスト（ユニットテスト）は、関数やクラスなどの小さな単位の動作を検証するために行われます。
			59	結合テストではどのような点を確認しますか。	テストコードの速度	異なるモジュールや機能同士が正しく連携して動作するか	セキュリティの脆弱性	コーディングスタイルの一貫性	2	結合テストでは、複数のユニットやモジュールが連携して正しく動作するかを確認します。
			60	単体テストと結合テストの実施タイミングの違いは何ですか。	単体テストは納品後、結合テストは設計時	単体テストは最終段階、結合テストは最初に行う	単体テストは各モジュール完成直後、結合テストはモジュール統合後	タイミングに違いはない	3	単体テストは各モジュール開発後すぐに、結合テストは複数のモジュールを連携させた後に行います。
			61	テストを行わずに開発を進めるリスクは何ですか。	ファイル容量が増える	作業が自動化される	バグや不具合に気づかず、重大な問題が後工程で発覚する可能性が高まる	メンバーのスキルが上がる	3	テストを省略すると、不具合が未発見のまま進行し、修正に大きなコストがかかる可能性があります。
			62	テストケースとは何ですか。	テストで確認する入力と期待される出力の組み合わせ	プログラミング言語の種類	テスト対象のパソコン本体	テスト結果の報告書のこと	1	テストケースは、「どのような条件で、何を入力し、何が返ってくるべきか」を記述したものです。
		自動テストの基礎と活用例	63	自動テストとはどのようなテストですか。	開発者が目視で確認するテスト	テスト用紙に手書きで記録するテスト	ユーザーに依頼して行うアンケート形式のテスト	プログラムでテストを実行し、結果を自動的に評価するテスト	4	自動テストは、テストスクリプトによりプログラムを実行し、期待する出力と一致するかを自動で検証します。
			64	自動テストを導入するメリットは何ですか。	チームの人数が減らせる	テスト実行の速度と正確性が向上し、繰り返し実行にも強くなる	コードの可読性が上がる	ソフトウェアの見た目が美しくなる	2	自動テストにより、毎回同じ処理を正確に高速で繰り返せるため、品質管理が効率化されます。
			65	自動テストと手動テストを使い分ける理由を説明してください。	手動テストの方が常に優れているから	予算が余っている場合に手動テストを選ぶため	自動化が難しい複雑な画面操作や直感的な評価は手動が適しているから	テスト担当者の気分で使い分けるため	3	UIの確認や複雑なフロー、直感的な使いやすさの評価には、人の判断が必要であり、手動テストが有効です。
			66	CI（継続的インテグレーション）と自動テストの関係は何ですか。	CIはバグの修正作業を可視化するツールである	CIは自動テストと関係なく、デプロイだけを行う	CIはユーザーが直接テストを行う仕組みである	CIではコードの変更時に自動テストを実行し、品質を保ちながら統合する	4	CIの一環として、自動テストを実行することで、統合時のバグを早期に検出できます。
			67	自動テストが向いていないケースにはどのようなものがありますか。	入出力が明確な機能の検証	UIやUXのような主観的評価が必要なケース	毎日繰り返し同じ動作を確認する必要があるケース	バックエンドAPIのレスポンス検証	2	自動テストは定型的な検証に向いており、見た目や操作感など人間の判断が求められる項目には不向きです。
			68	テストコードを保守する際に注意すべき点は何ですか。	毎回書き直すことを前提にする	テスト対象の仕様変更に応じて更新を忘れないこと	テストコードは本番コードと無関係なので放置しとよい	実行結果だけ記録すればテストコードは不要	2	テスト対象の仕様が変わった際に、テストコードも更新しなければ、テストが正しく機能しなくなります。
		バグ報告と再現性のある伝え方	69	バグ報告を行う際に、読み手に誤解を与えないために必要な工夫は何ですか？	バグの重大性を強調する主観的な表現を加える	報告者の推測を中心に記述する	客観的な事実と状況を時系列で整理する	報告内容を簡潔に1行でまとめる	3	バグ報告では、誰が読んでも同じ状況を再現できるよう、客観的な事実・状況を時系列で整理することが重要です。主観や省略された情報は誤解を生む原因になります。
			70	不具合の「発生条件」を明確に記載しないと、どのような問題が起こりますか？	バグの再現性が高まる	再現できず、修正が見送られる可能性がある	テストの工数を削減できる	不具合の影響範囲が正確に判断される	2	発生条件が不明確だと、「再現できない＝問題ない」と誤って判断される恐れがあります。その結果、本来修正すべきバグが見逃されるリスクがあります。
			71	一時的な不具合をチームに伝える際に有効な情報共有手段を挙げてください。	チケット管理ツールなどに記録し共有する	定例会議で口頭でのみ共有する	個人のメモに留めておく	開発が終わった後にまとめて報告する	1	チケット管理ツールなどに可視化された形で記録・共有することで、他のメンバーが確認しやすくなり、再発や調査が必要になった際に役立ちます。
			72	スクリーンショットやログを添付することで、どのような利点がありますか？	報告の文字数を減らせる	問題が発生したことを開発者が信じやすくなる	問題の発生状況を視覚的・正確に伝えられる	報告者の感情を伝えやすくなる	3	スクリーンショットやログは、報告の裏付け情報として有効であり、再現条件の特定や原因調査の助けになります。文字だけでは伝わりにくい情報を補完できます。
			73	バグ報告の品質がプロジェクト全体に与える影響にはどのようなものがありますか？	バグ報告の品質はプロジェクト全体に影響を与えない	高品質な報告が多いほど修正の優先度判断がしやすくなる	報告が多いほどプロジェクトは遅れる	報告内容が雑な方がスピード感があると評価されやすい	2	バグの内容・影響・緊急度が明確な報告があると、開発チームは優先順位付けや対処方針を迅速に判断できます。これはプロジェクトの品質や進行速度に直結します。

			74	テスターと開発者の間で認識齟齬が生まれやすいポイントにはどのようなものがありますか？	報告書のフォントサイズ	開発者が作業時間を記録しないこと	テスト環境のネットワーク速度	不具合の「あるべき動作」の理解の差	4	「期待される動作」についての認識のずれは、最も多く発生する齟齬の一つです。仕様書の記述や前提条件の共有が不十分だと、報告と実装がかみ合わない原因になります。
			75	開発初期段階に報告されたバグ情報を蓄積・分析することの意義は何ですか？	バグが減ってくると不要になるため、初期段階では分析不要である	初期段階のバグは品質に影響しないため無視してよい	初期のバグから傾向や弱点を分析でき、品質向上に役立つ	後工程に任せることで効率が上がる	3	初期の段階で現れるバグには、設計上の弱点や環境依存の問題が含まれていることが多く、それらを蓄積・分析することで品質管理の観点から早期改善に活かせます。