（1）影響分析対象物の品質

①ソースは解析しやすい（見やすい）ようになっていますか？

- □ コーディング形態が揃っている、コメントも分かりやすく書かれていている、モジュール化もできているなど、新規メンバーでも理解しやすい
- □ 上と下との間くらい
- □ 部分的にでもコーディング形態は揃っており、最低限のコメントも記載されているなど、新規メンバーでもある程度説明を受ければ理解できる
- □ 上と下との間くらい
- ◉ コーディング形態はバラバラでSE依存、コメントも少ないあるいは記述が分かりにくいなど、新規メンバーが理解するには時間を要する

（解説）言うまでもなく、影響分析を漏れなく的確に行うためには、ソースコードが解析しやすく整備されていることが重要です。ソース検索で引っ掛かっても、その箇所が今回の改修の影響範囲だと判断するにはコメントが分かりやすいこと、最新情報であることも重要です。常に前後のロジックを解析しないと影響範囲かどうか見当が付かないようでは非効率であり、判断ミスにもつながります。
プロセスとしては、コーディング規約が整備されていることはもちろん、それがメンバーに周知され、順守されていることが見やすいソースを維持管理する基本となります。コメントの書き方も規約に明記されていればなお良いでしょう。順守度を高めるには、QAグループなど第三者により定期的にチェックすることも有効です。
チェックに際しては、ここに挙げた一通りのことが出来ていれば最高、全く出来ていなければ最低というレベル感で自己評価してください。目安として、新しくプロジェクトに来たメンバーが理解し易い状態かどうかを考えてもらうと評価しやすいと思います。

②プログラム内の使われないロジックが判別できるようになっていますか？

- □ 改修により使われなくなったロジック（無意味なコード）はソース内に存在しておらず、無駄な影響範囲分析をする懸念がない
- □ 使われないロジックも残っているが、それらはコメントで明記し、影響範囲分析の対象外にできるようにしている
- □ 使われないことが明確に分かるようになっている部分と、曖昧な部分とが混在している
- ◉ ソースを見ただけでは使われないロジックか判別は難しいが、経緯を知っている人などを通じて確認できる
- □ 使われないロジックがどこか全くわからず、常に全体を影響範囲分析の対象とするしかない

（解説）使われなくなったロジックをソースから削除することはリスクもあり、現実には難しいものです。せめてコメントアウトしていれば、影響範囲から除外してよいことがはっきりしますが、ロジックを散々追跡してからコメントになっていることに気づくケースもあります。コメントアウトすらされていない場合には、影響範囲分析した後で有識者から「入力チェックでガードしたから、そこ通らなくなったんだよね」などとレビューで指摘されるケースもよくあります。
無駄な作業を避けることはもちろんのこと、通らない古いロジックに対応するために誤った改修方針を立てて機能性を損なうようなことが無いよう、使われないロジックを明確にすることは重要です。
プロジェクトによっては、過去の経緯を残すためにあえて削除しないケースもあり、その場合は「無意味なコード」ではないと判断できます。しかし、本来は経緯はドキュメント化しておくべきもので、ソースのコメントで管理するには限界があります。できればコーディング規約に無効となるロジックの扱いを明記しておくと良いでしょう。
チェックに際しては、使われないロジックが存在していない状態を最高とし、全く分からない状態を最低というレベル感で自己評価してください。

③影響範囲分析に活用するドキュメント、その存在と状態はどのようになっていますか？

- □ 「CRUD図」や「トレーサビリティマトリクス」などのドキュメントが整備され、改修に対して多角的に関連個所を把握できるようにしている
- □ 上と下との間くらい
- ◉ 「システム構成」や「機能一覧」などのドキュメントが整備され、影響範囲の抽出や改修箇所の特定に活用できるようにしている
- □ 上と下との間くらい
- □ 影響範囲やテスト範囲の抽出に活用できるドキュメントが存在しない、あるいは使える状態に整備されていない

（解説）この項目では「トレーサビリティマトリクス」のような高度なドキュメントが整備されている状態を最高レベルとしていますが、プロジェクトの規模や難易度によっては過剰かも知れません。理想論と現実の適正レベルとが一致しない場合は、自プロジェクトと適正レベルとの差異確認に活用してください。この設問は、影響分析にドキュメントをどのレベルまで活用できているかを客観的に把握するものだとご理解ください。
一般論として、「システム構成」「機能一覧」などは改修要件に対して影響しそうな箇所の大まかな把握や、テストすべき関連個所の把握に有用です。ただ、影響の把握レベルとしては一次元的で、現状の構成などから目星を付けるという程度であることは否めません。
これに対し、CRUD図やトレーサビリティマトリクスのように、処理と項目、要件と要件、要件と実装など2つの軸を関連付けするドキュメントがあれば、改修案件に対して、より多角的に影響箇所の把握ができます。また、INI ファイルやレジストリの定義情報なども、その変更が色々な箇所の動作に影響するので、影響範囲をより動的に把握できます。
また、ドキュメントは最新化されて誰もが見られる状態になっていないと、維持されず使えない文書になっていきます。即ち、自分達が保守してゆく上で必要な最低限の文書を決め、最新状態を維持する必要があります。

（2）影響分析やテスト項目抽出のし易さ

④過去の改修について、依頼内容から最終的な改修内容までの記録が参照できますか？

- □ 依頼内容から改修方針決定の経緯、実際に何をどう改修したかが全て記録され、容易に参照できるようになっている
- ◉ 上と下との間くらい
- □ 記録はあるが網羅されていない（一元管理されていない）。情報を得るにはあちこち参照したり人に聞いたりする必要がある
- □ 上と下との間くらい
- □ 過去の改修については殆ど記録がなく、「なぜこんな改修をしたのか」など、誰も分からず改修方針に苦労することが多い

（解説）長い期間保守を継続していると、「どうしてこんな方法で改修したのか」とか「ここに挿入されているコードは何の意味があるのか」など、経緯が分からなくて困ることが増えてきます。誰も経緯を覚えておらず、今回の改修で手を加えてよいものか全く判断が付かない、ということになれば改修計画はもちろん、品質にも大きな影響を及ぼします。
また、改修には消費増税対応など、過去の改修と類似するケースもよくあります。その場合、過去の要件と改修方針、実際に改修したプログラムが一望できることは、非常に参考になります。
チェックに際しては、記録があること、必要な時に参照できることの両方ができていれば最高、そもそも記録が無いとか、あっても参照できないような状態であれば最低というレベル感で自己評価してください。

⑤ソース上の関連が無い影響範囲についても、業務フローや関連する法規制などから洗い出せますか？

- ◉ ほぼ全員に一定レベルの業務知識があり、過去の対応記録や他の有識者によっても補うことで洗い出せている
- ☐ 上と下との間くらい
- ☐ 何人かの有識者（場合によってはお客様）のフォローに頼ることが多いものの、何とか洗い出しはできる
- ☐ 上と下との間くらい
- ☐ 業務知識が不十分で有識者も巻き込めないことが多く、ソース上追えない部分に影響分析漏れが多い

（解説）データ項目、プログラム関連をツールでたどって影響範囲を特定できれば良いですが、特にデータ項目名については、プログラム単位にローカル名が付けられており、必ずしもツールでたどりきれないケースがあります。ツールで発見できるようにシステム全体を通してデータ項目名を揃えるなど、標準化ができればベストですが、システム全体での統一は容易ではなく、また既に存在しているプログラムをこのために書き直す訳にもいきません。
従って、ツールなどでたどれない部分は業務知識を持ったメンバーや過去の対応記録、有識者レビュー等で、漏れなく影響範囲を特定することが重要です。
プログラム上の関連が無くても、ある部分に改修が加わった場合に、業務フロー上、変更しなければならない部分が出てくるケースなどは、業務知識がないと改修漏れを起こしかねません。このようなケースでは、改修要件を出す側は業務上当然であるが故に要件として明示するのを忘れることが少なくありません。

⑥プロジェクトメンバーは必要なスキル（業務知識、現行の理解、技術など）を有していますか？

- ☐ 個々のメンバーのスキルが一定レベルに保たれ、プロジェクトとして影響分析などが的確に行えるだけの高いスキルを持っている
- ☐ 上と下との間くらい
- ☐ 難易度の高い改修などではスキル不足による考慮漏れなどが発生することはあるが、通常レベルの改修については問題なく対処できる
- ☐ 上と下との間くらい
- ◉ スキル不足により指示の誤解が後で問題になるケースも多く、プロジェクト内で補える状況になっていない

（解説）システム全体を網羅的に理解しているメンバーはまれであり、開発に携わった箇所や保守を担当している箇所など、知識が特定の分野に偏っている状態が普通です。プロジェクト内のメンバーで補い合って高いスキルを維持できていれば良いという考え方もありますが、一定レベルまでのスキルは全員が持ち、必要なノウハウは整理して参照できるようにするなどの工夫も必要です。そうでないと、「あいつに任せたのが間違いだった」とか「彼にやってもらえば良かった」といった属人的な結論になってしまいます。
新たに加わるメンバーについても同様で、新規メンバーは特に体系的にかつ迅速に教育を施す必要があるため、そのための資料や情報を普段から整備しておくことが重要です。
影響分析やテストの漏れをなくすためにも、スキルの維持が出来ているのか、この項目をチェックすることで一度見直してみてください。

⑦ソースの修正部分に対応して実施すべきテスト項目が抽出できるようになっていますか？

- ☐ 特定のソースを修正したら絶対に流さねばならないテスト項目が洗い出されており、プロジェクトで共有され、改修の度に情報を更新している
- ☐ 上と下との間くらい
- ☐ 改修内容によるが、経験的に必要なテスト項目がある程度は抽出できる
- ◉ 上と下との間くらい
- ☐ 毎回、必要なテスト項目を抽出し直しており、時間の余裕が無いときは的確なものだけに絞り込めず、テスト漏れが良く発生する

（解説）ソース修正に対応したホワイトボックステストは比較的簡単に行うことができますが、業務視点、システム運用視点でのテストケースの洗い出しには業務とシステムへの深い理解が必要となります。
改修の度に全件テストできれば良いですが、現実にはその余裕はなく、必要十分なテスト項目に絞りたいと、誰もがおもっているはずです。一番怖いのは、実施すべきテストが行われずに、不具合を残したままリリースしてしまうことです。新規システムと違い、既に稼働しているシステムでは、リリース後の障害発生は致命的です。
有識者が常にいれば良いですが、必ずしもそうではない場合のために、テストケースが整備され再利用可能であることが望ましく、更にテストケースにテストの意味合いが記述されていると、システムそのものの理解にも役立ちます。

（3）プロセスの整備と活用状況

⑧改修要件の確認プロセスが明確で、順守できていますか？また有効に機能していますか？

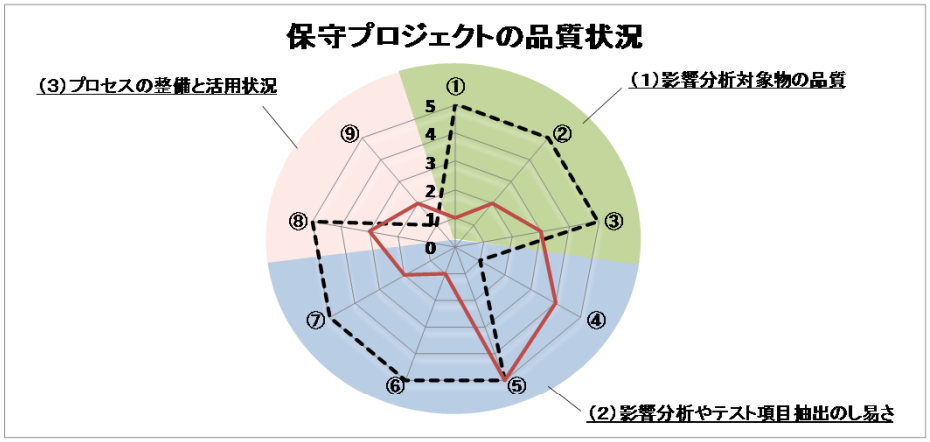
- ☐ 影響分析に入る前に、担当者と有識者とで要件確認するプロセスがルール化され、そのルールを順守している
- ☐ 上と下との間くらい
- ◉ プロセスはある程度決められているが、順守していないこともしばしばある
- ☐ 上と下との間くらい
- ☐ プロセスは定義されておらず、慣習的に実施している場合もあるが、有識者の確認無しに影響分析に入ってしまうこともある

（解説）そもそも改修要件に理解の齟齬があると、その後の全ての作業が無駄になったり、手戻りが発生したりするだけでなく、品質にも影響しかねません。要件を提示する人の知識や背景、受ける側の知識や背景が異なると、同じ言葉でも違った解釈になることはよくあります。即ち、改修要件の本来の意味を確認することが重要で、そのためには複数の人でチェックし合うことが必要です。
例えば、要件を記述する人がシステム部門である場合、要件ではなくその人が考えた「手段」までもが提示されるケースがあります。指示に素直に従ってしまうと、本来行うべき影響分析ができなくなる場合がありますので、「要件は何か」「何のために行う改修なのか」を常に意識する必要があります。
そのためにも、改修案件の受入手順がルール化され、できればQAグループなど第三者により順守状況を定期的にチェックすればなお良いでしょう。

⑨レビュープロセスが明確で、順守できていますか？また、有効に機能していますか？

- ☐ 実施方法や参加者、レビュー様式、レビュー品質指標などがルール化され、そのルールを順守している
- ☐ 上と下との間くらい
- ☐ プロセスはある程度決められているが、順守していないこともしばしばある
- ☐ 上と下との間くらい
- ☐ レビュープロセスは定義されておらず、慣習的に実施している場合もあるが、レビュー無しに改修作業に入ってしまうこともある

（解説）「リリースまでの期間が短い」「それぞれ仕事を抱えており、レビュー時間が取れない」など、さまざまな理由でレビューは簡略化されたり、省略されがちです。レビューには誤字脱字を含む表現上のチェックから、機能面、業務面、運用面と多角的な視点が必要です。質のよいレビューを行うためには、レビュープロセスや参加者の定義がされ、関係者がそれらをよく理解していることが必要になります。
レビュー実施においても、誤字脱字の発見に終始したり、レビュワー個人が興味のある点だけに集中した場合には、レビューの本質的な目的が達成できないことが考えられます。ルールに則ったレビュー運用を行って初めて、レビューの品質指標が有効なものになると言えるでしょう。
また、レビュー結果をもれなく記録し、反映するとともに、そこから不具合の傾向を学んでフィードバックすることは、改修プロセスの品質を向上する効果があります。
そのためにも、レビュー手順がルールとして明文化され、できればQAグループなど第三者により順守状況を定期的にチェックすればなお良いでしょう。



チェック分類	チェック項目	基準値	評価値
(1)影響分析対象物の品質	①ソースは解析しやすい(見やすい)ようになっていますか？	5	1
	②プログラム内の使われないロジックが判別できるようになっていますか？	5	2
	③影響範囲分析に活用するドキュメント、その存在と状態はどのようになっていますか？	5	3
(2)影響分析やテスト項目抽出のし易さ	④過去の改修について、依頼内容から最終的な改修内容までの記録が参照できますか？	1	4
	⑤ソース上の関連が無い影響範囲についても、業務フローや関連する法規制などから洗い出せますか？	5	5
	⑥プロジェクトメンバーは必要なスキル(業務知識、現行の理解、技術など)を有していますか？	5	1
	⑦ソースの修正部分に対応して実施すべきテスト項目が抽出できるようになっていますか？	5	2
(3)プロセスの整備と活用状況	⑧改修要件の確認プロセスが明確で、順守できていますか？また有効に機能していますか？	5	3
	⑨レビュープロセスが明確で、順守できていますか？また、有効に機能していますか？	1	2

＜事例編＞

項目：①ソースは解析しやすい（見やすい）ようになっていますか？

レベル5になったプロジェクトは、保守を開始して間が無いか、そうでなければよほど管理が行き届いている、あるいは複雑な改修があまり無いものと推察されます。
メンバーの入れ替わりや改修を繰り返すうちにルールが徹底されなくなり、分かりやすいコメントを付ける余裕も無くなってしまうのが普通で、レベル3くらいを何とか保っているというケースが多いのではないでしょうか。
熟練メンバーは慣れていて問題無いかも知れませんが、新規メンバーを受け入れた時に大きな障害を発生しがちです。レベル1に近づかないよう、定期的にルールの見直しを行いましょう。

事例1：この会社では下記のような仕組みを導入し、ソースの解析し易さを維持している

データ項目名称とそれを構成する名詞要素が登録されており、未登録データ項目名称の使用や命名規則を守らない名称登録が機械チェックされるようになっている。このチェックにより、名称サーチによる使用箇所特定の精度が向上する。

プログラム構造のひな型を数種類に限定し、スケルトン(骨格コード)に基づいたコーディングを義務づけており、これにより可読性と解析性が向上する。

プログラム仕様書にモジュール構造図を漏れなく添付している。これにより可読性と解析性が向上する。

また、特定のシステムではあるが、SEが作成するジョブ説明書とプログラミング構造を分離。SEは日本語で仕様を記述し、内部構造はプログラマが独自の内部設計ドキュメントを作成して管理することで、ドキュメントの可読性を向上している。

事例2：過去の改修箇所に説明がないために対応に困った例

あるプログラムの画面表示に数秒のDelayが入っており、これが何のためなのか誰も分からなかった。開発当時の仕様書をひも解いても記述が無かったが、更に何人もの担当者に確認したところ、ようやく、キーパンチする人の入力が速すぎるために、非同期処理が終了するまで、意図的に画面表示をDelayさせるためものだということが分かった。

意味が分からないままであれば、この処理には手を付けず、改修箇所のテストのみ行ってリリースしてしまうしかなかったが、意味が分かった後は非同期処理のタイミングとの関係で、修正・テスト方針が決定できるようになった。コードの意味が最初から仕様書・ソースのコメントなどに記述されていれば、悩む必要はなかった。

事例3：データチェックの意味が分からないまま、毎回余計なテストを強いられた例

複数のシステム（入力装置）から来るデータの初期値が、「文字型の空白」「数値のゼロ」「オールビットオフ」の3つを取りうる前提の処理がある。現在では入力装置が減り、処理も統一化されて3つもパターンが無いと思われるが、誰もパターンを減らしても良いという断定ができず、今でも3パターンを常に考慮した修正やテストをしていて時間とコストが掛かっている。

当初から各々のシステム（入力装置）から連係されるデータの初期値（どのパターンがどの入力装置特有のものなのか）が記述されていれば、その廃止などに伴いユーザと確認しつつ、処理の適正な見直しが行えたはずである。記述が無いと、それぞれの状態を想定したテストが必要になるため、テストケース作成時の注意が必要となり、テスト工数も通常より多いものとなる。

実際には、チェックリストの９項目全てについて事例が提供されていますが、ここでは項目１のみについてサンプルを掲載しました。