ＩＴプロジェクト

銀の弾丸●チェックリスト

Checklists as Silver Bullets

ちょっと待った、その開発！　チェックリストで楽々開発

目次

１２．　実務能力問題のチェックリスト　ｐ５８

１３．　リーダーシップ問題のチェックリスト　ｐ６５

序

フレデリック・ブルックスの、ソフトウェアエンジニアリングの問題を一挙に解決する「銀の弾丸などはない」と言う言葉はあまりにも有名です。これはソフトウェア開発にともなう様々な問題の解決には、特効薬などはなく科学的な根拠に裏打ちされた持続的で根気強い改善努力が必要であるという警句です。

ソフトウェア開発の現場で発生している問題は多種多様にわたっており、一個人、一組織の知力で対抗するにはあまりにも巨大すぎ、ともすればこれら全ての問題を一挙に解決してくれる特効薬はないものかと夢想してしまいます。

銀の弾丸がないかと言えば、ないとも言えません。ただしそれは地道な手間ひまがかかるものでしょう。それは我々の身近な日々の開発生活の中にころがっている我々の経験則に求めることができます。多くの経験則は一般的には明文化されず、個々人の記憶の中に内在している暗黙知として存在しています。これらの経験則は実に多種多様であり、これらが明文化され組織的に運用されれば、銀の弾丸の継続的な一斉射撃となり、さしもの狼男や悪魔的なバグや障害および個人間・組織間のコンフリクトを激減させることが可能になるのかも知れません。

経験則を明文化するためには、現場で発生している多種多様な問題のパターン化および構造化が必要であり、それらは最終的にはチェックリストという形で提示される必要があります。

本書で提供されるチェックリストは、それが継続的に実行されることを前提としており、それが実行されれば銀の弾丸にもなり得るし、実行されなければ単なる印刷された文字の羅列に過ぎない結果に終わるでしょう。多くの場合、この銀の弾丸は自分以外の他人だけに向けられ場合は何の効果も現さないのかも知れません。

■ チェックリストの有効性について

ソフトウェア開発のリスク回避手段の代表的なものとしてリスク管理表がありますが、その作成・運用の実態は非常に貧弱なものでしょう。 リスク管理表へのリスク項目の登録にあたって行われていることは、プロマネ自身の経験における記憶から抽出されたリスクやチームメンバーから出された意見などに基づいたものがほとんどだと推定されます。 実態は、リスク管理表なるものがあればまだよい方で、リスク管理表そのものが存在しないプロジェクトも非常に多いのです。

リスク管理表の最大の問題点は、膨大なリスクの中でとりあげるべきリスク項目を何にして良いのか分からないという点にあります。一般的には、人・もの・金・ＱＣＤ・情報などを切り口にしてリスク項目の抽出を行うことになりますが、プロマネ個人の能力や経験に依存する限り失敗プロジェクトは決して減ることはないでしょう。個人の能力や経験に依存している限りは、「２：６：２の法則」に従って、成功者が２割、何らかの傷を負った者が６割、完敗者が２割の結末を迎えることになります。

組織的活動におけるリスクの「基準点」は、個人に求めるのではなく、やはり組織に求めるべきであり、組織におけるリスクの「基準点」とは、組織が蓄積した「過去の失敗事例」に他なりません。 過去の失敗事例は、その原因ごとにパターン化することが可能で、パターン化できるということはチェックリスト化できるということです。さらにこれらのチェックリストは、その因果関係を分析することで構造化が可能となります。

構造が明確になれば、何が基本的なリスクか、すなわちキーリスクは何かが明白になります。これらの構造化されたチェックリスト群は、リスクの網羅性を高め、さらに短時間でのリスク抽出を可能にし、リスクのモレを防止するでしょう。

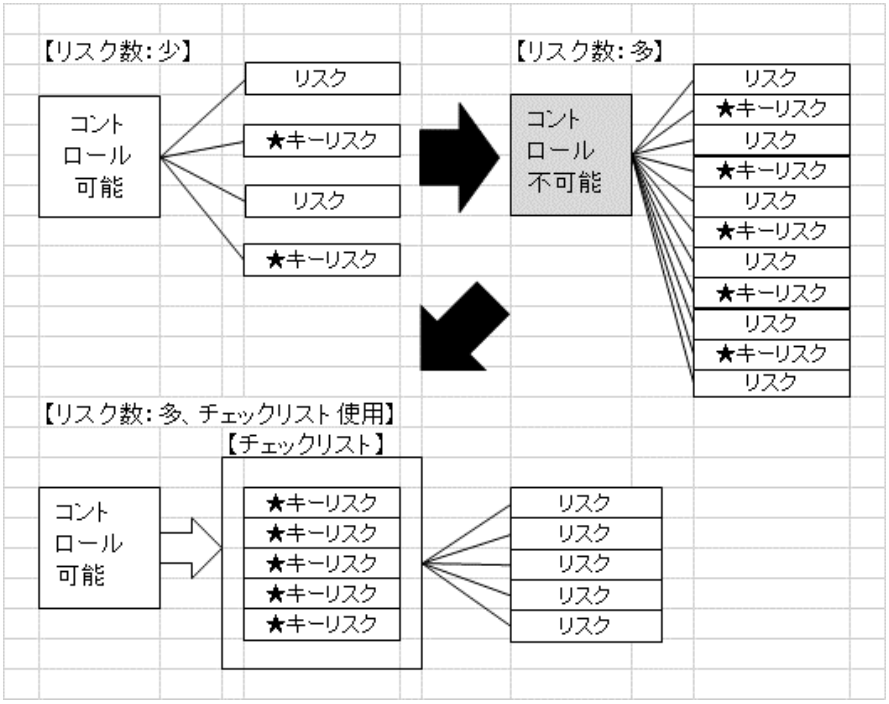
おそらく数個のキーリスクを撃ち落とせば、成功するプロジェクトは倍増するに違いないでしょう。

■ チェックリストとキーリスクについて

ソフトウェア開発の現場は、その初期の工程からリリースに至るまでに、毎日何らかの問題やトラブルに悩まされ続けており、開発業務には実にさまざまなリスクが内在しています。そのことについては、実際の開発現場に居る開発者たちが一番良く知っています。 しかしながら一向に問題が減ることはなく、何度も同じ問題が繰り返されているのは何故でしょうか。

ソフトウェア開発における無数のリスクないしは問題をグルーピングすると、要求仕様問題、見積り問題、コミュニケーション問題、実務能力問題などいくつかのグループに集約することができます。さらに、それぞれのリスクは、そのインパクトの大きさによって致命的なリスクと通常リスクに分けることができます。致命的なリスクは、開発プロジェクトに対してそのＱＣＤおよび人的な被害が非常に大きいもので、ここでは「キーリスク」と呼ぶことにします。一方通常リスクとはリカバリーがある程度可能な軽微なリスクです。

われわれ人間が、一度に認識および対処可能なキーリスクはせいぜい数本の指で数えられる程度だと思われます。それ以上のキーリスクが存在している場合、いかに優れたリーダーと言えども、見逃しや誤算によるキーリスクの問題化を避けることは困難でしょう。しかし、キーリスクをチェックリストで束ねることで、この問題は比較的容易に解決することが可能となります。図１は、このことを明瞭に示しています。



【図１：チェックリストの役割】

■ **リスク解消のアプローチ**

本書におけるリスク問題解決のアプローチは、下記の３点に拠りました。

１．過去の５００件の問題事例をリスク解消の基準点とすること。

２．５００件の問題事例を、その原因別パターンに従ってチェックリスト化すること。

３．抽出されたチェックリスト群を因果関係に従って構造化すること。

■ **失敗のパターン**

筆者において収集した約５００件のリスク事例を元に、問題の共通パターン化分析を行った結果、これらのさまざまな問題は７つの群および１７のグループに集約することができました。

問題のパターン　７群、１７項目を下記に示します。

第１群　①不条理な顧客要求、②見積り問題

第２群　③相互義務の不履行、④リスク回避の失敗

第３群　⑤コミュニケーション不良

第４群　⑥情緒的開発姿勢、⑦モチベーション問題　⑧本質の把握ミス、⑨優先順位問題

第５群　⑩ドキュメント・ベース開発の欠如、⑪実務能力問題

第６群　⑫リーダーシップ問題、⑬チームプレー問題　⑭手抜き問題、⑮時間認識問題

第７群　⑯人材育成／ノウハウの継承問題、⑰学習能力の欠如問題

■ **失敗パターンのトリクルダウン**

失敗のパターンは、図２に示したとおり、第１群の失敗は第２群へと、上位群から下位群へその悪影響を拡大しながら進行し、当該プロジェクトの失敗のみならず次なるプロジェクトの失敗の原因となり、失敗の連鎖を生み出していくことになります。悪貨は良貨を駆逐すると言われる通り、トリクルダウン（したたり落ちる）は悪い事については容易に起きるが、良い事についてはなかなか起きません。 また全ての問題は、最初に組織によって引き起こされ、それが個人問題を引き起こしていることが分かります。 組織問題の解決なくしては個人の問題も解決できないと言うことになりますが、これは個人がこの問題から免責されることを意味しません。なぜなら組織を動かしているのは個人、特にリーダーたちであり、その役割の重さには大きなものがあります。

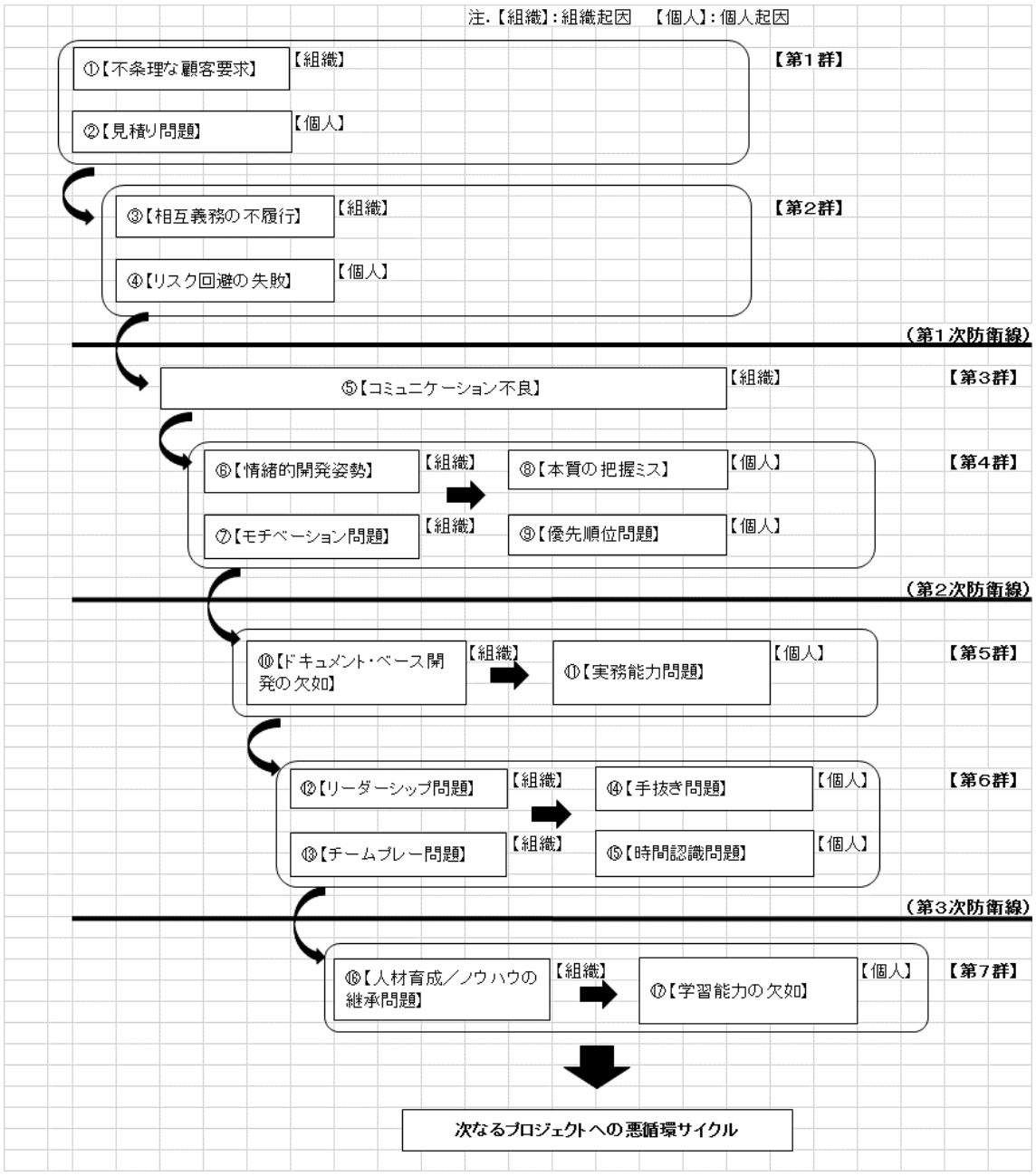


図２　失敗パターンのトリクルダウン

リスクが見逃されるかまたは対処に失敗した場合、これは現実のトラブルとして顕在化し、問題（problem）化することになります。 リスクは７つの群および１７のグループに集約され、要求仕様問題、見積り問題およびコミュニケーション問題の三点が基幹的なリスクであることが明白になりました。 これらの問題は開発の時系列の流れに沿って階層構造を形成しており、初期階層での不手際は、次の階層問題を引き起こし、初期階層から最終階層に向けて問題は拡大していき、まさに失敗パターンのトリクルダウンが起きています。

プロジェクト問題の起点は、第１群の【不条理な顧客要求】および【見積り問題】の二つ集約することができます。これら二つの問題に最初に対処すべきことは、第２群の【相互義務の履行】および【リスクの回避】ですが、この対処の失敗は、相互義務の不履行およびリスク回避の失敗問題として現われてきます。第２群の失敗は、第３群の【コミュニケーション不良】によって回復不可能な傷を負い、第４群以降に示したさまざまな問題をトリクルダウン的に引き起こすことになります。

問題のパターンを失敗の物語として読み解けば次のようになります。

■ 失敗の物語

１．失敗の序曲

問題の発端は、第１群の【不条理な顧客要求】および、それに対する【見積り問題】の二つにあり、それに対して開発の準備工程における第２群の【相互義務の不履行】および【リスク回避の失敗】が最初の防衛線の役割を果たさず、多くの問題が次工程にパススルーされてしまう。

２．第２次防衛線の崩壊

開発組織における最大の弱点である第３群の【コミュニケーション不良】が、第４群の【情緒的開発姿勢】【モチベーション問題】【本質の把握ミス】【優先順位問題】を招くことになる。

３．第３次防衛線の崩壊

第２次防衛線の崩壊は、開発の現場における第５群の【ドキュメント・ベース開発の欠如】【実務能力問題】を引き起こし、さらに第６群の【リーダーシップ問題】【チームプレー問題】【手抜き問題】【時間認識問題】へと拡大していくことになる。

４．失敗の終曲～失敗体質の定着化

以上の状況の中で、第７群に示した【人材育成／ノウハウ継承】が行われることもなく、プロジェクトの最終工程で行われるべき【学習＝振り返り】も行われず、１７個の問題パターンが失敗体質としてその組織に定着していき、将来にわたって拡大再生産されることになります。

■ **失敗の１７パターンとリスク事例５００件の関係**

各１７の失敗パターンに関係するリスク事例の件数および割合は図３の通りです。

失敗要因	件数	%	起因	群	件数	%	防衛線
①不条理な顧客要求関連リスク	37	7%	組織	第1群	50	10%	第1次防衛線
②見積り問題	13	2%	個人				
③丸投げ問題(相互義務の不履行)	19	4%	組織	第2群	84	16%	
④リスク回避失敗	65	12%	個人				**26%**
⑤コミュニケーション不良	66	13%	組織	第3群	66	13%	
⑥情緒的開発姿勢(目標設定不良)	22	4%	組織	第4群	85	16%	第2次防衛線
⑦モチベーション問題	31	6%	組織				
⑧本質の把握ミス	18	3%	個人				
⑨優先順位認識問題	14	3%	個人				**29%**
⑩ドキュメント・ベース開発の欠如	38	7%	組織	第5群	58	11%	
⑪実務能力問題	20	4%	個人				第3次防衛線
⑫リーダーシップ問題	31	6%	組織	第6群	118	23%	
⑬チームプレー問題	26	5%	組織				
⑭手抜き問題	46	9%	個人				
⑮時間認識問題	15	3%	個人				**34%**
⑯人材育成・ノウハウ継承問題	42	8%	組織	第7群	60	12%	**12%**
⑰学習能力の欠如	18	3%	個人				
合計	521	←重複カウントを含む			521	100%	

図３　失敗の１７パターンとリスク事例５００件の関係

■ **リスク事例の分析**

（件数別のワーストリスク）

⑤コミュニケーション不良	６６件　１３％
④リスク回避の失敗	６５件　１２％
⑭手抜き問題	４６件　９％
⑯人材育成／ノウハウの継承問題	４２件　８％
⑩ドキュメント・ベース開発の欠如	３８件　７％
①不条理な顧客要求	３７件　７％
⑦モチベーション問題	３１件　６％

（組織／個人の起因別割合）

組織起因	３１２件　６０％
個人起因	２０９件　４０％

上記データが意味することは、プロジェクトの失敗は、人のコミュニケーションの失敗、リスク回避の失敗や手抜き問題等によって引き起こされているということになります。さらにこれらの個人的な問題は、最初に組織的な問題がトリガーになって引き起こされています。

プロジェクトの問題は、組織的な問題を発端とした人間の思考および行動に関する問題であると言えます。

■　１７のパターンの具体的なリスク内容

各問題パターンの具体的なリスク内容は次の通りです。

問題のパターン	リスク内容
①不条理な顧客要求	・決まらない要求仕様
	・無理な短納期・低コストの要求
②見積り問題	・不十分な仕様調査・仕様理解
	・見積り交渉力の弱さ
③相互義務の不履行	・プロジェクト統合管理の不在
	・説明責任の欠如（丸投げ）
	・情報共有の不足
④リスク回避の失敗	・仕様決定遅れ
	・短納期
⑤コミュニケーション不良	・顧客・開発チーム間の意思疎通・情報共有の不良
	・チーム内の意思疎通・情報共有の不良
⑥情緒的開発姿勢	・感情的・情緒的な思考・行動
	・数値目標不在
⑦モチベーション問題	・モチベーションの低下
⑧本質の把握ミス	・自分の頭で考えない自律性の未熟さ
⑨優先順位問題	・割り込み作業等の優先度判断
	・必須業務に優先順位はないという認識の欠如
	・納期第一優先の誤り。
⑩ドキュメント・ベース開発の欠如	
	・開発行為の科学的根拠（ベースライン）の喪失
	・低品質なドキュメントが招くＱＣＤの失敗
	・文書化能力の低下
⑪実務能力問題	・知識不足（技術、仕様、システム構造、評価テスト）
	・仕様決定能力の低さ
	・設計・製造能力の低さ
⑫リーダーシップ問題	・マネジメントなきプロジェクト管理
⑬チームプレー問題	・相互義務の不履行
	・相互扶助の不在
⑭手抜き問題	・必要工程の中断ないしはスキップ
⑮時間認識問題	・仕様凍結期限意識の欠如
	・期限・タイミング意識の欠如
	・許容時間・必要時間認識の欠如

⑯人材育成／ノウハウの継承問題
・個人のスキル向上の疎外
・組織能力向上の疎外
・プロジェクトＱＣＤの低下
⑰学習能力の欠如問題　・失敗に学ばない
・振り返り（ラップアップ）の未実行
・改善活動の未実行

■ 本書の構成

第１章においては、ソフトウェア開発の全てを網羅した包括的なチェックリストを、そのチェックタイミングとともに示しました。 第２章以降においては、冒頭にて、「First Point」としてその章の全体像および要旨を示し、続いて、パターン化された問題の因果関係の時系列順に、それらの問題を予防するための個別のチェックリストを記載し、章末には、「Last Point」としてその章のまとめを示しました。また各章におけるチェックリストの理解を助ける目的で適時「コラム」欄において若干の解説を加えました。

本書において提示されたチェックリストは、致命的な手順抜けの予防を目的としたもので、問題の解法を示したマニュアルではなく、全ての手順を含むことを意図していません。これらのチェックリストは、読者の所属組織やプロジェクトの実情に応じて追加・改変して使用されることが望まれます。

本書の有効な使い方

- はじめにひと通り通して読む。
- 目次の中で自分が抱えている問題から先に読む。
- チェックリストに○×△印をつけ、現状のレベルおよび課題を知る。
- 各章の冒頭の First Point および末尾の Last Point に注目して読む。
- 巻末に記載したチェックタイミング時系列順 Check List 一覧に沿って読む。
- 索引を活用して実践の手掛かりを見つける。
- 気づいたことや自分の意見などをメモ欄に書き込み実践につなげる。

以降、本文においては、問題の７群、１７のパターンの順に、それぞれのキーリスクで構成されたチェックリストを示すことにします。

１．　ソフトウェア開発リスクの全体像

Point　：　複雑な問題の解決にはチェックリストを

□　チェックリストは、開発問題の解消に有効であることを知る。

□　チェックリストは、その事前のチェックタイミングに用いることでのみ有効となる。

□　チェックリストは、失敗リスク回避の強力なツールである。

ソフトウェアの開発は多くの複雑な手順が必要とされますが、すべての手順を記憶しておくことは不可能であるし、また一々細かな手順書に従って行動することも実際的ではありません。 これから示すチェックリストは、重要な行動項目の抜けや見落としを防ぎ問題の発生を予防するという観点で構成されています。

それぞれのチェックリストの確認は、ある行動に着手する直前に行うことで効果が発揮されます。この確認のタイミング（Check Timing）は一時停止点（Pause Point）とも呼ばれており、物事を開始する前のリスク排除の重要な働きを持っています。

どのような行動においても、それを無事に遂行するためには、事前にその行動の重要なポイントを一度立ち止まって冷静に見るという「静」の時間が理性や合理性を働かせる重要な働きをします。この「静」と実際の行動における「動」が相まって、人間行動の過ちを防ぐ働きをしています。

次に示した「Check List ＃１　開発プロジェクト安全チェックリスト」は、ソフトウェア開発の全体を包括的に俯瞰したもので、プロジェクト開始の前にチェックすることでプロジェクト開始にあたっての全体的なリスクの発見に有効に働くことになります。

Check List ＃1　開発プロジェクト安全チェックリスト

▶Check Timing：開発準備時

【見積り回答前】◀ Check Timing

- □ 要求仕様の骨子が決定され明確に書面にて提示されていること。
- □ 見積り回答書の内容は、要求仕様骨子の実現に見合った開発期間および開発費になっていること。
- □ 見積り回答書には、明確な要求仕様骨子に対してのみ見積られ、不明なものについては見積りに含まれていないことを明記すること。
- □ 見積り回答書に、開発に必要な実行条件がある場合、その条件を明記すること。

【設計着手前】◀ Check Timing

- □ すべての要求仕様が決定され、明確に書面にて提示されていること。
- □ 全ての要求仕様に関して、全ての不明点および疑問点が払拭されていること。

【製造着手前】◀ Check Timing

- □ 基本設計が完了され、基本設計書として明確に書面にて提示されていること。
- □ 詳細設計が完了され、詳細設計書として明確に書面にて提示されていること。
- □ 全ての設計書に関して、全ての不明点および疑問点が払拭されていること。

【単体テスト前】◀ Check Timing

- □ 単体テスト試験書が作成済であること。
- □ テスト対象機能のコーディングが全て完了していること。

【結合テスト前】◀ Check Timing

- □ 結合テスト試験書が作成済であること。
- □ テスト対象機能の単体テストが全て完了していること。

【総合テスト前】◀ Check Timing

- □ 総合テスト試験書ないしは運用（シナリオ）テスト試験書が作成済であること。
- □ 総合テスト対象機能の単体・結合テストが全て完了していること。

【成果物リーリース前】◀ Check Timing

- □ 全ての機能が要求仕様通り、システムとして実運用を満たしていることが確認されたか。
- □ 発見済で未修正の不具合が残存していないか。

Check List ＃2　プロジェクトを成功させるための基本条件

▶Check Timing：開発準備時

【統合管理】

□ ベンダーを中心とした統合的プロジェクト管理を行うこと。

□ 適切なプロジェクト・マネジメント能力を発揮すること。

【コミュニケーション】

□ プロジェクト傘下の全組織および全会社間の密接なコミュニケーションを行うこと。

□ 日次情報共有会議によるチーム内での情報共有を行うこと。

【要求仕様】

□ 主要な仕様を適切な時期までに凍結させること。

【リソースの確保】

□ 見積り交渉において、開発内容に見合った工期・予算を確保すること。

□ 顧客システムおよびその運用業務に精通していること。

□ 開発規模および難易度に応じた開発体制を構築すること。

【開発】

□ 顧客価値の重要度順の仕様凍結ならびに開発を行うこと。

□ ドキュメントに基づいた開発を行うこと。

□ データに基づく開発を行うこと。

□ 開発効率化・失敗の再発防止のために普段の改善活動を行うこと。

２．　不条理な顧客／ベンダー要求のチェックリスト

First Point　：　あいまいな要求仕様と無理な短納期・低コスト

プロジェクトの起点は顧客あるいはベンダーから提示される要求仕様および見積り依頼にあります。これはまた同時に、プロジェクトにおける問題の起点もここにあるということになります。開発の第一段階における要求仕様の品質および見積り回答の妥当性がプロジェクトの成否を決定づけると言っても過言ではありません。

顧客要求に関する主なリスクは次の通りです。

□　決まらない要求仕様

□　無理な短納期・低コスト

これらのリスクの解消法は、「仕様の早期凍結」および「妥当な納期・開発費の獲得」ですが、これを実現するためにはそれ相応の仕様知識、技術力および交渉力が必要になり、その開発組織の能力レベルに応じた結果しか生み出せないでしょう。

◎プロジェクト問題の起点である要求仕様は製品の質と量を規定する。

■　不条理な顧客要求

Check List　＃３　不条理な顧客要求への対抗策

▶Check Timing：見積り回答前

□　開発責任者と要求側責任者の密接かつ直接的な話し合いを実行すること。

□　完全な要求仕様書の早期提示時期の確約をとること。

□　開発仕様はすでに凍結合意されたものに限定すること。

＊困難な要求に対しては、困難な条件の提示を行うこと。

□　要求元からのＰＭ、有識技術者等の投入

□　要求元からの評価テスト支援メンバーの投入

□　開発要員増強のためのプレミアムコストの要求

□　暫定版または優先度順のリリース、分割リリース等の条件提示を行うこと。

□　「暫定版の不具合発生責任は要求者に負っていただく」と言う条件提示を行うこと。

□　「暫定版の市場投入は、限定数にて実験店舗のみ」と言うような条件提示を行うこと。

□　「全店稼働は正式版のみで行うこと」と言うような条件提示を行うこと。

＜コラム＃１＞　敵を知り己を知れば百戦危うからず～タフネゴシエーション

自分の弱みにばかり気を取られていると敵が見えなくなる。敵の弱みにも注目するところからタフな交渉が可能となる。“All or Nothing”的なデジタル思考をやめ，敵をよく知ることから交渉という戦いを始めよう。

■ 仕様凍結

Check List ＃４　早期仕様凍結

▶Check Timing：仕様凍結準備時

□ 早期仕様凍結のために顧客および関連各社の参加・協力の要請を行うこと。
□ 仕様凍結の期限を切り、元請側と下請側で合意をしておくこと。
□ 顧客・元請・下請間の直接コミュニケーションによる要求仕様の期限内凍結を行うこと。
□ 集中検討会ないしは合宿等にて短期集中的に仕様決定を行うこと。
□ 受注者側においても提案型仕様凍結を行うこと。
□ 顧客価値の優先度順に仕様凍結を行うこと。
□ 顧客価値の高い仕様順に開発着手すること。
□ 開発仕様の目的・背景・範囲・内容を文書にて明確化すること。
□ 仕様未凍結状態ないしは疑問点・不明点を残したままで先行開発着手は行わないこと。

＜コラム＃２＞　仕様の全体像および主要仕様の把握は設計工程の前に済ませておくこと

優れた開発者における開発工程ごとの時間の使い方は、初期工程に重く、後になるに従って軽くなっています。さらに開発すべき対象をどれだけ正確に把握しているか、さらにどれだけ早く把握しているかが勝負の分かれ目となります。この勝負は見積り時および要件定義工程においてほぼ決定しているでしょう。プロジェクトを成功させる要因の重さは、”何を作るのか”が分かることで六割、”どのように作るのか”が分かることで三割、その他要因で一割程度だと思われます。

これらの事実を直感的に理解している開発者は見積り時において、仕様の全体像の把握および主要な仕様の把握に全力を上げています。これができれば妥当な見積りが可能となり、基本設計の概要を描くことも可能になります。要するに、進むべき地図を先に明確に描いておくのか、それとも迷いながら進むべき道を探すのかの違いです。どちらが早くしかも正確に目標に到達できるかは一目瞭然です。

Check List ＃5　仕様理解

▶Check Timing：仕様調査時

□ 仕様検討の段階で要求者と徹底的な仕様検討を行うこと。
□ 要求仕様の背景や意味を必ず理解しておくこと。
□ まずは仕様の全体像の把握から始めること。
□ 疑問・不明点の発掘を行い、その解消に向けて要求者に対し積極的な行動を取ること。
□ 仕様決定のＱ＆Ａは直接対話による確認を行うこと。
□ 不明なことは直ちに分かっている人・部署に聞くこと。
□ 習得した内容を、ドキュメントによって他のメンバーに伝えること。
□ 早期の仕様凍結を行うこと。
□ 基幹仕様未決定で開発に着手してはいけない。

Check List ＃6　仕様調査およびＱ＆Ａ運用

▶Check Timing：仕様調査時

□ 仕様内容の骨子の事前の把握と整理を済ませておくこと。
□ 仕様調査は、顧客価値の重要度順および基幹的仕様から始めること。
□ 仕様追加・変更の影響度の事前調査を済ませておくこと。
□ 全ての問題点・疑問点を掘り起こすこと。
□ 重要な仕様・基幹仕様に関するＱ＆Ａは直接対面コミュニケーションで確認すること。
□ 仕様内容および変更内容はチーム内（含む評価チーム）で即時的な情報共有を行うこと。
□ 仕様調査のＱ＆Ａ情報や変更内容は要求仕様書や設計書も同時に更新すること。
□ 仕様決定事項や変更事項は集中的かつ情報共有可能なシステムで管理すること。

＜コラム＃３＞　なぜそんなに大人しいの？

仕様決定の遅延で何度もその後の工程が崩れた経験をしているのに、被害を受けている人たちはなぜか大人しい。愚痴を言いつつただ頑張るだけで良いのだろうか。責任ある人や組織にクレームをつける勇気も元気もないのだろうか。論理的な仕事を遂行しているはずなのに一向に合理的な行動を起こそうとしない。なぜそんな状態になってしまったのか、それをどうしたら良いのか、もう一度自分自身で真剣に考える時が来た。

Check List ＃7　影響度表作成のポイント

▶Check Timing：設計時

□ 正確な構造設計書・テーブル関連図・プロセスフローから影響度表を作成すること。
□ 事前の実装調査にて判明した他機能への影響情報を記録すること。
□ 設計作業時において知りえた影響情報を記録しておくこと。
□ 製造時において知りえた影響情報を記録しておくこと。
□ デバッグ時に判明した他機能への影響情報を記録すること。
□ 単体・結合・総合の各テストにおいて判明した他機能への影響情報を記録すること。
□ 市場障害対応にて判明した他機能への影響情報を記録すること。

■ 要求仕様書の検証

Check List ＃8　要求仕様書のチェックポイント

Check Timing：要求仕様書発行・受領時

１．　妥当であること

□ 顧客やユーザーのニーズと一致していること。
□ 上位のシステム要求仕様書などの関連する他のドキュメントとの矛盾がないこと。
□ 未確定項目がある場合は、どのように合意するか、依頼者と合意形成方法を決めておく。

２．　あいまいでないこと

□ 要求仕様書に記述されている要求が、ただ一通りに解釈できること。
□ 要求仕様書の“良し悪し”を判断する手段や基準をもつこと。
□ 「範囲」を読み取れるように要求を表現すること。
□ 仕様は「仕様である」ことを明示し、説明は「説明である」ことを明示して記述すること。
□ 要求仕様書では、記述内容が“特定”できる表現になっているものを“仕様”とすること。
□ 要求仕様書の構成や内容は、後工程の読者に分かるように書くこと。
□ 「等」や「ｅｔｃ」の文言は使用しないこと。使用する場合は、○月○日までに決めるとコメントをつけること。

３．　完全であること

□ 顧客やユーザーの、情報システムに対するニーズが漏れなく要求仕様書に記述されており、かつ図表の参照や用語の定義などの、要求仕様書の形式が整っていること。
□ 「境界」は早い段階で決めること。
□ 「要求」のモレを防ぐために、カテゴリの分類や要求の分割・階層化に漏れがない、隙間がないことを確認できるようにすること。
□ 要求仕様書には、「操作性」「保守性」「交換性」などの「品質要求」を記載すること。
□ 階層化の基準として、以下を（状況によっては組み合わせて）使い、「隙間」なく分割すること。
時系列分割（時間軸分割）／構成分割／状態分割／共通分割

□ モレなく書くこと。

□ 要求仕様の番号をテストケースの番号とひもづけし、テストケースにモレがないことを確認すること。

□ 仕様をグループに分け、さらに集合を小さくし、混じり気のない仕様のグループを作る。

□ ＜グループ名＞に要求の性質を持たせるためには、範囲をあらわしていることを意識してグループ名を選ぶこと。

□ 「・・・・は、・・・・しない」という「否定表現」を避け、ｔｈｅｎとｅｌｓｅの両方を明らかにすること。

４． 矛盾がないこと

□ 要求仕様書内部で矛盾や衝突がないこと。

□ ほかの機能の仕様と衝突していることに気づくためにも、仕様は早期に展開すること。

□ 早い段階で全体の仕様化を行うこと。

５． 重要度と安定度のランク付けがされていること

□ 各要求について、重要度と安定度を示す指標を明確につけておくこと。

□ 確認中の仕様をそのまま記述し、変わる可能性があることを明記すること。

６． 検証可能であること

□ 開発されたソフトウェアが、要求仕様書に記述された要求を満たしているかどうかを確認可能であること。

□ 検査部門の人に、「検査可能」という側面から要求仕様書のレビューを実施してもらう。

□ 品質要求（「操作性」「保守性」「交換性」など）はテストでも確認すること。

７． 変更が容易であること

□ 要求仕様書に対する変更が、容易に、完全に、一貫して行えるようになっていること。

ａ）目次や索引、明確な相互参照が整備され、使いやすい構造になっていること。

ｂ）冗長でない、つまり、同じ要求が要求仕様書内で複数個所に記述されていないこと。

ｃ）他の要求と混ざらず、各要求を独立・分離して表現している。つまり、要求が互いに依存していないこと。

□ 重複なく書くこと。

□ 仕様書全体を「均一」に記述することにこだわらないこと。関係者間で共有できている認定仕様まで、詳細に記載しなくてもよい。

□ 仕様番号の確定作業は、仕様化の最初の段階では行わないこと。グループ分け確定後に行うこと。

□ 似た記述が続く場合に、何が違うかをすぐに読み取れるようにすること。

８． 追跡可能であること

□ 要求仕様書に記述された個々の要求に関し、その起源が明確であり、開発が進行するに伴って作成された文書等との対応付けがとれること。

ａ）後方追跡可能性

ｂ）前方追跡可能性

□ 設計や実装の工程で明らかになった「仕様」は、要求仕様書に書き戻すこと。

□ 「要求」と「理由」をセットで表現すること。
□ 要求仕様には固有の記番号を付けること。
（参考資料： ＩＥＥＥ８３０品質特性、ＵＳＤＭ）

Last Point ： 要求仕様の早期凍結と開発時間の確保

１．要求仕様の早期凍結

要求仕様を早期に決定できない主な理由として，次の４つが挙げられます。

□ 仕様知識の低下
□ 開発技術力の低下
□ プロジェクト・マネジメント力の低下
□ 外注依存の高さ

＊これらの問題に対してベンダー側および下請け側に必要な行動は次の通りです。

【ベンダー側】
□ 仕様ノウハウを学習・蓄積すること。
□ 一定の割合で設計・製造を実行し，開発技術力を保持しておくこと。
□ 外注まで含めた統合的プロジェクトマネジメントを実行すること。

【下請け側】
□ 仕様ノウハウを学習・蓄積すること。
□ 設計能力を進化させること。
□ 製造能力を進化させること。
□ 評価能力を進化させること。
□ プロジェクトマネジメントを実行すること。

要求仕様の精度はプロジェクト成功の必須要件の一つであり，それは要求する側および請ける側双方の同時連携が必要となります。これは「相互義務の履行」という重要な組織行動規範の一つです。

２．開発時間の確保

開発時間の確保に必要な行動は次の通りです。

□ 無理な短納期要求に妥協せず，合理的なアプローチによるタフネゴシエーションを行うこと。
□ 要求仕様の全貌およびリスクの把握により必要期間・コストの見積り間違いをなくすこと。
□ 開発対象システムの仕様・機能を体系的に理解すること。
□ 開発着手前に基本部の要求仕様の凍結を行うこと。
□ 仕様検討期間中に要求仕様の疑問点・不明点をすべて解消すること。

□ 過去の失敗に学び同様の失敗による手戻り起さないこと。
□ プロジェクトマネジメント能力を磨き、リスク解消およびプロセスの遵守に努めること。
□ 改善活動を通じて技術力の向上に努めること

時間を確保するための主要な三つの要素は，「見積もりにおける妥当な開発期間の獲得」，「要件定義の早期凍結」および「開発行為における失敗の回避」であり、この三つについて全力を尽くせばプロジェクトの成功は確かなものになります。

３．　見積り問題のチェックリスト

First Point　：　勘と度胸の見積りからの脱却

「見積り」は開発行為の中で最も重要なものです。見積りの失敗はプロジェクトの失敗に直結します。見積りは契約行為であり，開発者や組織の実力が映し出される鏡であり，不条理な要求に無定見に従うのではなく，利をあせることなく，どこに妥当な落としどころがあるのかを探る，実に人間臭い行為だと考えたほうがいいでしょう。

見積り問題における主なリスクは次の通りです。

□　仕様調査不足および仕様理解不足

□　見積り交渉力の弱さ

◎「見積り」は開発に許される時間と資金を規定する。

＜コラム＃４＞　見積リが高いと言われないために

たとえ簡単な仕様であったとしても、議事録などを要求仕様書の代わりにしても不思議に思わない発注担当者や受注側担当者がいます。これらの行為からみえることは、今から作ろうとしている製品やそれに要するコストや時間に関して、全くの他人事的な姿勢です。個人で車や家を購入する場合に、このような雑な行動は決してしないはずです。 過不足のない要求仕様書がなければ、発注側・受注側の双方において妥当な金額の見当さえつきません。

ベンダー側においては正確な要求仕様書を提示する義務があり、受注側においても不十分な要求仕様書に対して不足部分・不明点・疑問点などを見積り回答の前にベンダー側に確認する義務があります。これらの確認を行った後で、必要ならばソースコードの調査による基本構造および影響範囲の特定を行い、過去の類似開発の実績も考慮した上で見積りを行えば妥当な工数および期間を見積もることが可能となるでしょう。

ベンダー側に対して、これらの調査内容の資料に基づいて説明を行えば、単に高いと言うコメントは出てこないはずです。必要工数について異論が出なければ、後は単価の問題であり、これは会社間での協議事項となります。

見積りには当然リスク分も含めることになりますが、リスク分については絶対に相手に話をしてはいけません。言えば必ずカットされます。またリスク分の金額が本体分に対して異常に高いことは避けなければいけません。余りにもリスク分が大きい場合は、そのリスクを解消した後に見積るか、見積り条件を限定すべきでしょう。

Check List ＃９　見積り回答リスク回避

▶Check Timing：見積り回答前

□ 適切な開発費と開発期間の確保ができているか。
□ リスク物件における分割見積り・分割開発・分割リリース等の交渉を行っているか。
□ 複数社体制プロジェクトにおいて、自社責任範囲を見積り回答書に明記しているか。
□ 過去の類似開発の見積り／実績データを参考にした見積りを行っているか。
□ 見積りの使用目的を確認すること。
例；顧客要求のため、受注に直結しない参考値のため、など。
□ 口頭での依頼に対しては口頭で返し、口頭レベルの依頼・回答は一切正式なものとしては扱わず、責任も持たない旨を通告しておくこと。
□ 顧客の正式要求のものならば、要求日および妥当な回答希望日を記入した見積り依頼書を両社の正式なルートを通して、要求仕様書と共に提出していただくよう伝えること。
□ 正式要求書の場合、回答期限に間に合いそうもないと判断された場合は、その理由を明確にし、時間を置かず直ちに要求者と期限の交渉を行うこと。

Check List ＃１０　見積精度向上

▶Check Timing：見積り回答前

□ 見積り対象の仕様知識に習熟しているか。
□ 見積り対象システムのプログラム構造を理解しているか。
□ 見積り前に要求仕様の事前調査を済ませているか。
□ 既存の設計書等が不備な場合、影響範囲に絞ったソースコードの調査を行ったか。
□ 事前調査の結果をドキュメントに残したか。
□ 見積り対象仕様開発における過去の失敗リスクを把握しているか。
□ 見積り対象仕様開発に必要な技術を保有しているか。
□ 過去の開発案件の見積りと実績の差異の原因把握および改善を行っているか。

Check List ＃１１　見積り回答書

▶Check Timing：見積り回答前

□ 見積り回答期限の事前確認は行ったか。
□ 見積り対象の仕様は明確になっているか。
□ 見積りにインプット条件は全て網羅したか。
□ 見積りにアウトプット条件は全て網羅したか。
□ 仕様の疑問点・不明点は全て顧客に確認したか。
□ 仕様変更が及ぼす影響範囲は特定したか。
□ 特別な見積り条件の要求があった場合、その考慮は行ったか。
□ 見積り範囲やリスクに関する条件を見積り回答書に記述したか。

Check List ＃１２　概算見積り

▶Check Timing：見積り回答前

□　要求仕様の全体像を、その目的・意味・背景を含めて把握すること。

□　自分で見積り可能なものとそうでないものを分けること。

□　未経験項目および重要リスク項目をリストアップすること。

□　それぞれに対して、必要工数を大雑把に３段階（大・中・小）程度に分けて記入する。各段階に要する工数は、例えば、大＝１ ヶ月、中＝２ 週間、小＝ １週間等に決めておく。重要リスクについても、もしそれが起きた場合についても同様に工数の大・小を記入しておく。自分で決められないものについては経験者や上長の意見を聞く。

□　これらの仕事を何人で実行するかを想定しておく。

□　算出した開発期間の合計を想定人数で割ったものが想定されるスケジュール期間となる。

＜コラム＃５＞　主な見積り手法

正確な見積りをするためには、まず仕様理解力と設計能力が必要となります。何をどう作るかが分かっていれば、基本的には見積りは可能です。要求仕様が明確なのに正確な見積りができないということは、自分の仕様理解力と設計能力が未熟だということになります。見積り手法はあくまでも道具にすぎず、その手法があれば見積りができるというものではありません。ソロバンがあっても使えなければ意味がないのと同じことです。まずは自分の設計技術スキル向上と経験の積み重ねが必要となります。

見積り手法の主なものには、①経験的手法　②ＬＯＣ（Line　of　Code）法　③ＦＰ（ファンクション・ポイント）法などがあります。

経験的手法は広く行われている方法ですが、見積りの精度を上げるためには、過去のプロジェクトにおける見積り対実績の差およびその原因についての分析データの蓄積が必要となります。

ＬＯＣ法は、工数＝ステップ数÷開発生産性ｘ（１＋間接費要員比率）ｘ余裕率で求められます。

ＦＰ法は、工数＝基準値ｘ（０．６５＋調整値÷１００）で求められます。

ＬＯＣおよびＦＰの手法は、その開発組織としての今までの実績のデータの積み重ねがないと、計算式のパラメータ値を決めることができません。

Last Point ： 見積回答書の重要性

受注者の最大の目的は，その仕事によって利益を上げることでしょう。見積回答書は，達成すべき仕事内容，その対価および実行期間を約束してしまうため，見積りミスや仕事の失敗は直ちに赤字を招き，企業の存続を危うくする場合もあります。

さらに見積回答書は開発チームの仕事の原点であり，その後の開発における人・モノ・カネ・時間を規定するものであり，すべての書類やドキュメントの中でも最重要な書類だと言えます。見積回答書は，その会社や組織のすべての実力を映し出す鏡といっても過言ではありません。見積回答書の基本要件は次の 2 点だけです。

□ わかっている内容のみ見積ること。
□ わかっていない内容は見積りに含んでいないことを明記すること。

４．　相互義務不履行問題のチェックリスト

First Point　：　まず先に自分が動くこと

相手がある仕事においては必ず双方に果たすべき義務があり、そのことを「相互義務の履行」と言います。ソフトウェア開発における最初の重要な「相互義務の履行」は、最初の工程である見積りと要求仕様凍結において実行される必要があります。当然のことながら、最初に発注者側が“何を”開発して欲しいのかを明確に要求仕様書として提示する義務があり、受注者側においては、その要求書の不明点や疑問点について発注者側に問いかけ明らかにする義務があります。当事者双方におけるそれらの行為の品質が、これから始めようとする開発の成否を決定することになるでしょう。

相互義務の不履行に関する主なリスクは次の通りです。

- □ プロジェクト統合管理の欠如
- □ 説明責任の欠如（丸投げ）
- □ 組織間の必要な情報の提供不足

＜コラム＃６＞　「相互義務」があることを知る

仕事の依頼者と請負者の間には相互に果たすべき義務があります。依頼者は自分の依頼するものを説明する義務があり，請負者はそれを実行する義務があります。依頼者はお金を出したから後はよろしくなどという態度は許されません。何を依頼するのかその内容を提示することなく，何とか作ってよと言われても何を作ってよいのか皆目わかりません。自動車を作ってと言われても，自家用車なのがトラックなのか，何馬力なのか，などの仕様が明確に提示されなければ何を作ってよいのかわからないのは当たり前のことです。要求仕様を提示しない依頼者も悪いが，それに対して忍従している請負者も相当の愚か者といわざるを得ません。どこまでも譲れるものではないでしょう。

■ 丸投げ問題

Check List ＃１３　仕事の丸投げ状況

▶Check Timing：見積り回答前･後

- □ 見積り回答内容以外の仕事を強要してはいないか。／されてはいないか。
- □ 責任範疇外の仕事を強要してはいないか。／されてはいないか。
- □ 過不足のない要求仕様書を提示しているか。／提示されているか。
- □ 合理性･妥当性に反した開発期間･内容を強要してはいないか。／されてはいないか。
- □ 合理性･妥当性に反した開発コストを強要してはいないか。／されてはいないか。
- □ 仕事に関する疑問･質問に誠意をもって回答しているか。／回答を受け取っているか。
- □ 相互の約束がお互いに履行されているか。
- □ 優位な立場を悪用した不条理な責任転嫁が行われていないか。

Check List ＃１４　仕事の丸投げに対抗するポイント

▶Check Timing：事前準備工程

- □ 顧客側の役割および職務責任を契約書等にて明確にすること。
- □ ベンダー側の役割および職務責任を契約書等にて明確にすること。
- □ 下請け側の役割および職務責任を契約書等にて明確にすること。
- □ 統合的プロジェクト管理により全ての関係組織･会社間の役割･責任を明確にすること。
- □ 上司および部下の役割および職務責任を社内規定等にて明確にすること。
- □ 自分における役割および職務責任を明確に意識すること。
- □ 当事者同士で解決できない場合は、さらに一段上のマネジメントに相談すること。
- □ 事前準備･改善活動等による効率化を行い、時間切迫が理由の丸投げをなくすこと。
- □ 改善活動等によるスキルアップを行い、能力不足が理由の丸投げをなくすとこ。
- □ 丸投げ行為は職務放棄とみなし厳罰をもって臨むこと。
- □ 丸投げに泣き寝入りすることなく組織的な対抗措置をとること。

＜コラム＃７＞　丸投げは破滅地獄への一里塚

業務委託が責任放棄のにいたる悪魔のステップを次に示します。

ｓｔｅｐ１　：　仕事が忙しくて間に合わない。

ｓｔｅｐ２　：　外部の業者から人を派遣してもらい、仕事を手伝ってもらう。

このときはまだ仕事の責任は自分にあると思っている。派遣者のミスが自分の指示の悪さに有ったとしたら自分が責任を負うのは当然だと思っている。

ｓｔｅｐ３　：　派遣者で穴埋めしてきたが、仕事の忙しさが限度を越えてきたので、業務のまとまった部分を外部の業者に委託する。最初のうちは委託した業務についての進行状況や品質や問題点について委託先と綿密にコミュニケーションをとっており余り問題は発生しなかった。

ｓｔｅｐ４　：　さらに忙しさが増し複数の仕事を抱えざるを得ない状況になってきた。もう委託先の仕事の状況を見る余裕すらなくなってきた。委託業務の細かい指示もできなくなり、自分の専門領域の技術を学習する時間も無くなり、自分の主な仕事は多数の外注に仕事を割り振ることだけになってきた。

ｓｔｅｐ５　：　外注から納入される成果物に多くの欠陥が発生するようになった。

ｓｔｅｐ６　：　多数の欠陥品に対して自分では到底その責任を負い切れないので、委託先外注に対してその非を責め続け製品責任の転嫁をせざるを得ないようになってしまった。

ｓｔｅｐ７　：　ｓｔｅｐ４～ｓｔｅｐ６の悪循環がくりかえされると同時に、学習不足の結果は自分の専門的なスキルを劣化させ、外注へまともな指示すら出せなくなってしまった。

ｓｔｅｐ８　：　回復不可能な大障害が発生してしまった。

この悪魔のサイクルはｓｔｅｐ３で止めなければいけなかった。悪魔のサイクルを止める役割・責任はマネージャにあり、仕事の分散だけではこの問題は解決できない。仕事のプロセスの見直しや仕事のやり方の見直しも必要となる。更に開発力の強化も必要となる。多くのムリ・ムダの排除のためのリスク排除活動も必要となる。担当者に直接的な責任はないが、そのひどい状況およびそれを放置していたらとんでもないことになるリスクについて諦めることなくマネジャーたちに発信し続ける義務はある。

■ 相互義務の履行

Check List ＃１５　チームプレーにおける相互義務の履行

▶Check Timing：開発全工程

□ 顧客チームとベンダーチーム間の相互義務の履行および連携はできているか。
□ 同一プロジェクトに参加している他社間の相互義務の履行および連携はできているか。
□ ベンダー・顧客側においては何を開発するのかを要求仕様書等にて明示すること。
□ 請ける側においては要求に対しての実現方法を見積り回答・設計書等で明示すること。
□ 要件定義チームと開発チーム間の相互義務の履行および連携はできているか。
□ 開発チームと評価テストチーム間の相互義務の履行および連携はできているか。
□ 上司と部下の間の相互義務の履行および連携はできているか。

＜コラム＃８＞　ノブレス・オブリージュ（Noblesse oblige）の精神

「位高きは，徳高きを要す。優位の者こそ大きな義務を負う」（レヴィ公爵，『格言と省察』）

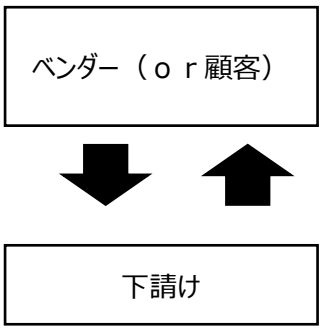
相互義務の履行の反対は丸投げです。多重請負構造の致命的な欠陥はこの丸投げによってもたらされる業務品質および製品品質の著しい劣化にあります。安くできるなどという安易な気持ちならば足元をすくわれるような危機的な状況を必ず招くことになります。

古来日本人は常に連帯を保ち、その共同体の上位の者も下位の者もともにその応分の責務を果たし、優位の者こそ大きな義務を負うこととされてきました。このノブレス・オブリージュの精神こそが繁栄の永続的な循環を生み出すことに気づくべきでしょう。

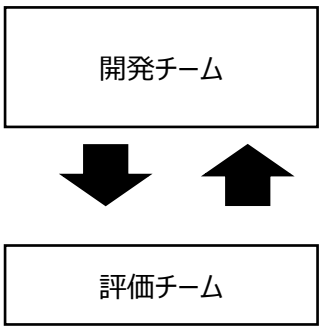
Last Point ： 共にやるべき義務を果たすこと

組織対組織および個人対個人における相互義務の関係を図示すると次に示します。
あなたたちの状況はいかがでしょうか。

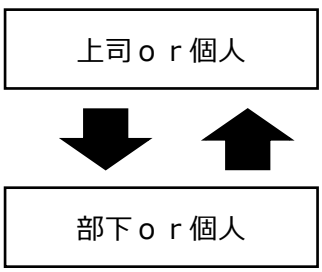
【図４ 相互義務の相関関係】



（会社間の相互義務）
・明確な要求仕様の提示義務
・不明点・疑問点についての回答の義務
・合理的かつ妥当性のある開発委託の義務
・統合プロジェクト管理の義務
・明確な見積り回答の義務
・不明点・疑問点についての質問の義務
・合理的かつ妥当性のある開発実施の義務



（組織間の相互義務）
・内部リリースの約束期日を守る義務
・評価テストに耐えうる製品品質提供の義務
・評価テストに必要な情報提供の義務
・不具合の適時修正の義務
・仕様適合、運用適合確認の義務
・客先リリースの約束期日を守る義務
・不具合発見・報告の義務



（個人間、上司・部下間の相互義務）
・指示ｏｒ依頼内容の意味・背景説明の義務
・適時フォローｏｒ相互支援の義務
・指示内容理解の義務
・適時フォローｏｒ相互支援の義務

５．　リスク回避失敗のチェックリスト

First Point　：　初期工程に集中しているリスク

ソフトウェア開発プロジェクトにおける主なリスクを挙げるとすれば、見積りの失敗、決まらない要求仕様および開発行動の失敗の三つに集約することができます。医療分野や航空宇宙分野などのミッション・クリティカル以外のソフトウェア開発においては、見積り問題および要求仕様問題が最大のリスクであり、これらの問題が開発行動における失敗の大部分を誘発しているものと思われます。そのような実情からして、一般的なアプリケーション開発の主なリスクはプロジェクトの初期工程に集中しており、妥当な見積りおよび要件定義の実現がリスク排除の主なターゲットであると認識すべきでしょう。

ソフトウェア開発の主なリスクとして下記の４点が挙げられます。

□　仕様決定の遅れ

□　短納期

□　未経験分野の開発

□　仕様凍結前の事前着手

＜コラム＃９＞ソフトウェア開発を取り巻く諸状況　～工程別分業方式の致命的な欠陥

近年オフショア開発の進展に伴って、要件定義はベンダー、設計は国内下請け会社、製造は海外オフショア会社、評価は国内下請け会社もしくはオフショアなどと、一つのプロジェクトの各工程を複数の会社で分業するような開発形態が増えてきました。この分業体制には多くのリスクや問題点があります。実際この「工程別分業方式」による開発で多くのプロジェクトが失敗しているのを見てきました。この方式の最大の問題点は、情報の共有が極めて困難であるということです。従来の開発体制は、ベンダー側にてプロジェクトの統合管理、要件定義およびシステムの基幹部の開発および評価を担い、不足要員を国内下請け会社で補う形が一般的でしたが、現在は外形的にはベンダー側における開発や評価業務は下請け会社にシフトされており、特に製造工程はコストメリットの追求のために海外オフショア会社に発注されている場合が多いようです。 この工程別分業方式の致命的な欠陥を以下に示します。

１．各工程が異なった会社に分離・分散されたことによる共有すべき情報の分断化

一社で全ての開発工程を担う場合に比べて、複数社で分割した工程を担うことは極めて難しいことは誰にも容易に分かるはずですが、多くのベンダーにおいてはその認識が極めて薄いようです。ソフトウェア開発は、一社で全てを実行しても失敗する危険性が高いのにもかかわらず、どのようなリスクがあるのか深く考えず、単にリスク分散だとかコスト削減だとかの旗印のもとに複数社に分割発注することは正気の沙汰とは思えません。工程別分業方式における主なリスクを以下に示します。

□　**統合管理および統合コミュニケーションの欠落**

元請会社において下請け各社の進捗管理や品質管理をまとめる統合管理および統合コミュニケーシ

ョンの実行がなされない場合があること。

□ **ドキュメント・ベースの開発の喪失**

ベンダーも含めた各社における完全な要求仕様書や設計書などのドキュメントに基づく開発および評価の実行がなされない場合があること。

□ **統合的プロジェクト体制の構築失敗**

ベンダーも含めた各社における開発力、すなわち必要な人材や体制の投入状況に関する検証の実行がなされない場合があること。

ベンダー側において統合プロジェクト管理を実行していなければ、仕事も責任も放棄した丸投げ発注となり、プロジェクトは必ず失敗するでしょう。ベンダーの下に有機的に統合された分業体制を「共有分業」と呼ぶならば、そうでない分業体制は悪しき「分離分業」と言えます。

プロジェクトにおける統合管理、情報共有、ドキュメント・ベース開発および適切な開発体制の構築は、分業の有無にかかわらず本来一つの開発組織においても実行されていなければならないプロジェクトを成功させる重要な要因です。下請け会社においては、ベンダー側において欠落している組織機能の補完を行う行動をとることが新たなビジネスチャンスともなるでしょう。

２．文化・言語が異なる海外オフショア会社との間のコミュニケーションの阻害

十分な統合管理もできず、ドキュメント・ベースの開発もできず、さらに十分な開発体制の構築もできないような組織において、文化も言語も異なる海外の会社に設計・製造・評価などの工程を発注したらどうなるかは誰にでも分かることでしょう。すでに分離・分断されている工程間にさらに深い溝を掘ってしまう結果になる。要求仕様書の行間を読めというような非科学的な姿勢しか持ち合わせていない人たちが作成した貧弱な仕様書に基づいて設計や製造を発注されたオフショア会社においては、きっと書かれているものだけしか設計・製造できないでしょう。おまけにあいまいな日本語は非ロジカル言語であり、その日本語で書かれた貧弱な要求仕様書は中国やベトナムの技術者にとって意味不明なしろものなのかも知れません。まともなプログラムができなくて当たりまえ。それを補うために新たにまたブリッジＳＥなる職種を作りだしたり、プログラムの受け入れ検査組織を作ったり、補修するための新たな技術者を用意したり、結局オフショアのコストメリットが無くなるばかりか、赤字に陥り、劣悪な品質や納期遅延の結果を生んでしまうことでしょう。

オフショアのみならず開発を成功させることのできる開発組織は、前記の統合管理、ドキュメント・ベース開発および優れた開発能力をもった組織だけだと言えます。

■ ベンダーのリスク

Check List ＃１６　基本的なベンダーリスク

▶Check Timing：見積り時、要件定義時

- □ 実現不可能な納期・開発費を強制されていないか。
- □ 複数社・複数工程間の統合的なプロジェクト管理、問題情報共有が行われているか。
- □ 適正な要求仕様書が妥当な時期までに提示されているか。
- □ 仕様等の不明点・疑問点についての回答時期・内容は適正か。
- □ ベンダー側担当者の仕様知識、技術能力、管理能力は妥当か。
- □ オフショア開発に対する進捗・品質管理に問題はないか。

Check List ＃１７　ベンダーリスクへの対応のポイント

▶Check Timing：見積り・要件定義時

- □ 不条理な要求に対してデータに基づく交渉等により妥当な納期・コストの獲得を行うこと。
- □ 仕様の提案や集中検討会などを通して早期の仕様凍結に全力を尽くすこと。
- □ 要求仕様には必ず抜けや誤りがあるという前提を持つこと。
- □ 密接なコミュニケーションを維持し、疑問点・不明点を払拭すること。
- □ 契約外の要求に対しては追加期間・コストの要求を行うこと。
- □ 評価時間が確保できないようなリリース直前の仕様変更要求を請けないこと。
- □ 工程のスキップや中断などの不正な指示には従わないこと。
- □ 開発や評価に必要な情報・機材・開発環境が適正な時期に提供されること。
- □ 問題の解決にあたっては、一方的な丸投げを行うことなく、両者が協力して臨むこと。
- □ ベンダー側にて不足している役割を新たな仕事として取り込む提案ができるか。

■ 開発工程のリスク

Check List ＃１８　開発工程の３時点でのリスク回避

１．開発の入り口で押さえるリスク　　▶Check Timing：見積り･要件定義時

□ 適正な見積りにて妥当な開発期間と費用を獲得すること。

□ あいまいな要求仕様の明確化および早期の仕様凍結に全力を注ぐこと。

□ 要求仕様書を常時メンテナンスし”使えるドキュメント”として残すこと。

□ ＱＣＤの数値目標の設定を行うこと。

２．開発中に押さえるリスク　　▶Check Timing：開発中

□ あらゆる無駄の排除とリスク項目の解消を実行すること。

□ 設計書をリアルタイムでメンテナンスし、常にプログラム内容との同一性を保持しておくこと。

□ 失敗の真因･再発防止策をドキュメントとして残すこと。

□ 創意工夫の内容を設計書やガイドラインなどのドキュメントに残すこと。

３．開発の出口で振返ること（ラップアップミーティング）　　▶Check Timing：開発終了時

□ ＱＣＤの目標値と実績値を比較･分析し、問題の原因を数値データと共に明確にし、対策した結果をまとめておき、プロジェクト完了報告書にて他チームとも情報共有を行うこと。

□ 失敗に学ぶ。失敗あるいは創意工夫の記録を振返り、次の開発･開発者に申し送る。

Check List ＃１９　プロジェクトリスク回避　▶Check Timing：見積り･要件定義時

＊プロジェクトのリスクを認識し事前の対策を講じているか。

□ 見積り交渉における妥当な開発費および期間を獲得すること。

□ 要求仕様の早期凍結を行うこと。

□ 開発業務品質向上のための改善活動を実行すること。

□ リーダーによる開発仕様の全体像の把握を行うこと。

□ 開発メンバーによる開発仕様の十分な理解を行うこと。

□ 開発メンバーに対する適切な仕事の配分を行うこと。

Check List ＃２０　スケジュール遅延リスク回避

▶Check Timing：見積り・要件定義時

□　実行すべき内容が明確になっているか。
□　開発に必要な期間・工数の見積りに間違いはないか。
□　無理なスケジュールが組まれていないか。
□　仕様凍結遅延ないしは仕様変更が多発していないか。
□　予定目標と実績進捗は日々管理されているか。
□　リーダーにおける開発内容の全体像の把握が十分か。
□　担当者が仕様を十分に理解しているか。
□　担当者ごとの生産性・能力・性格を考慮した仕事の配分が行われているか。

＜コラム＃１０＞　複数社担当開発におけるリスクの回避

＊複数社担当開発における主なリスクには下記のようなものがあります。
□　仕様の理解が相互に異なったためにおこるデグレや実装漏れなど。
□　同一モジュール製造に複数社がかかわった場合、１社の開発遅延がロードブロックとなり、他社が製造開始できないこと。
□　１社の不具合によってテストが進捗できないこと。
□　複数社におけるベースソース部改修の同期がとれず正式版のビルドができないために結合・総合テストが開始できないこと。

＊オフショア会社に関する注意点は次の通りです。
□　意思の疎通が取りにくく、時間がかかること。
□　修正対応が遅いこと。
□　仕様に記述されていないことは質問もしてこないし、製造されることもない。
□　日本語仕様の解釈能力が低いこと。ただし日本語仕様書自体の品質も悪い。

これらのリスクを解消するためには、週一回程度の定例会議だけでは不十分であり、自社のチーム内では日次の短時間情報共有会議を毎日実行し、日々の進捗の確認、全ての問題・課題の解消、情報共有を毎日実行する必要があり、ベンダーおよび他社とのコミュニケーションも密接に行う必要があります。 さらにベンダー側においては必ず複数社に対する統合管理の実施が必須であり、下記の役割の実行が不可欠になります。

【プロジェクト統合管理の役割】
□　関係各社における進捗の相互調整
□　関係各社における問題・課題の相互調整
□　関係各社間の情報共有

□ 関係各社間の協力体制および共通開発ルールの確立

□ 開発仕様の早期提示

Last Point ： リスクは人にあり

結局、リスクは人にありと言う事で、要求仕様を提示する人が誰で、見積りを行う人が誰であるかということが非常に重要なポイントです。

リスク回避の達成レベルは、それぞれを担当する実行者の能力レベルに応じたものにしかなり得ず、プロジェクトの成果も同様であると言えます。

６．　コミュニケーション不良問題のチェックリスト

First Point　：　コミュニケーションは共通の目的・目標を発見するために

コミュニケーションの役割は次の通りです。

①コミュニケーションは、人的な情報系の中枢を司る。

②コミュニケーションは、何をどのように実行すべきかを明らかにする。

③コミュニケーションは、共通の目的に向かって関係者全員のベクトルを揃える。

ソフトウェア開発のすべての工程で最も重要なことは，顧客との間，およびチーム内で良好なコミュニケーションが実行されなければならないということです。潤沢な開発費，余裕のある開発期間，整備された開発環境がそろっていたとしても，孤立した人間関係しか持てないチームは必ず失敗を招くことになります。 良好な意思疎通は，共通の目的に向かって関係者全員のベクトルを揃え，実行すべきことは何か（What）を明らかにし，それをどのように実現するのか（How）を明らかにするでしょう。開発は，人と人との関係性の中からしか生み出され得ないものだということをもう一度再認識する必要があります。

コミュニケーションに関する主なリスクは次の通りです。

□　顧客・開発チーム間の意思疎通・情報共有の不良

□　開発チーム内の意思疎通・情報共有の不良

◎プロジェクト問題の起点である要求仕様は製品の質と量を規定する。

◎「見積り」は開発に許される時間と資金を規定する。

◎コミュニケーション問題は、開発における人間の思考および行動を支配する。

良質なコミュニケーションなくしては満足のいく製品のリリースおよびＱＣＤ目標の達成は不可能です。コミュニケーション問題は、他の問題パターンすべてに対して強力な影響力を持っています。

＜コラム＃１１＞　日次情報共有会議の励行を

ソフトウェア開発における問題の多くは、チーム内でのコミュニケーション不足によって引き起こされているのではないでしょうか。例えば、毎日 20 分程度の短時間日次会議を実行して、その中でチームメンバー全員から、「①昨日何を実行したか。②明日何を実行する予定か。③今抱えている問題は何か。」の３点について報告・応答を続ければ、「内部レビューで指摘した内容が全て反映されていない、仕様確認を怠ったまま設計／コーディングを行ってしまう、コーディングミスが多い、見直しをしていない、デバッグをしていない」というような低レベルの問題は毎日ベースで発見でき、毎日ベースで部下の教育・しつけもできるということになります。一カ月の進捗遅れが突然発覚するというようなこともなくなるでしょう。

毎日ベースのこのショートミーティングを実行してみる価値は十分にあります。

■ コミュニケーションに関するリスク

Check List ＃２１ コミュニケーション（基礎編）

▶Check Timing：会議・会話の前・中・後

- □ 顧客との会議に、予想される想定問答など十分な事前準備をして臨んでいるか。
- □ 他人の評判だけに従って自分の行動を決めてはいないか。
- □ 発言すべきか否かを、必要性によらず感情によって決めてはいないか。
- □ 先に相手の話を聞くなど、丁寧なコミュニケーションを行っているか。
- □ 相手の話の途中に割り込むようなことをしてはいないか。
- □ 毎日、短時間でもコミュニケーションを継続することで良好な人間関係を維持しているか。

＜コラム＃１２＞ 他人からの評判だけに従って自分の行動を決めてはいけない

自分の所属する集団の反応によって自分の行動を決めるという姿勢は、日本人全般にみられる習慣の一つです。これは日本人の稲作農耕に由来する伝統的慣習である集団行動主義に起因したものと考えられます。その良い面としては、集団全員の思考・行動の方向性を一つにまとめ困難な目標を達成する強力な手段となりますが、反面個々人の自主的な思考・行動を制限することにより自律性を損ない集団を誤った方向に導きやすいという負の局面を持っています。いわゆる「空気を読む」という行動はこの延長線上にあり、過度に他人の思いを意識することで個人の自由闊達な思考・行動を押さえ込んでしまいがちです。 他人の目を過度に意識することにエネルギーを消耗させるのではなく、”良い仕事をするためには何をすべきか”ということに気持ちを集中させれば、もっと自由闊達なコミュニケーションが行われることでしょう。その結果、報告・連絡・相談も自発的に頻繁に行われ、良いコミュニケーションが信用・信頼を醸成し仕事のスキルも向上できるに違いありません。

Check List ＃２２ コミュニケーション（中級編）

▶Check Timing：会議・会話の前・中・後

- □ 話をする前に、自分の中で伝えたい内容を整理しているか。
- □ 伝えるべきこと、伝えてはいけないことをわきまえているか。
- □ 重要な取り決めは相互の合意を文書にて残しているか。
- □ 相手の話を良く聞かずに、自分の意見ばかりを話していないか。
- □ 事実を伝えるよりも良く見せることに注意が傾いてはいないか。
- □ 恥や外聞を恐れ、不明点・疑問点をその場で確認することを避けてはいないか。
- □ 合理的かつ妥当な結論を導き出すことよりも、議論の勝ち負けにこだわってはいないか。
- □ 客観的なドキュメントや数値データに基づいてコミュニケーションを行っているか。

＜コラム＃１３＞　話し言葉だけでは伝わりにくい

相手に話をうまく伝えるためのポイントは次の二つに絞られます。

□　客観的な資料や数値データを示しながら説明をすること。

□　相手の能力レベルに合わせた話し方をすること。

口頭でのコミュニケーションは感情的ないしは情緒的な表現に傾きがちで、あいまいさが混入しやすいものです。これらのあいまいさを排除するためには、資料を使って数値やデータを示しながら話をすることが一番です。また話す相手の知性や技術的能力のレベルに合った話し方をしなければ話は伝わりません。話の要所ごとに相手の理解度を確認することが必要です。理解度の確認は、相手に理解した内容を復唱させれば分かるでしょう。また抽象度の高い内容は、具象的な例え話をすることで相手の理解を深めることができます。話の上手な人ほど例え話がうまいものです。

＜コラム＃１４＞　簡単な事柄でも何度も同じ質問を受けるようなら文書での伝達を

相手に口頭で伝えても良い場合は、すでにその相手がその仕事の基本的な意味や背景について共通の理解を持っていると判断できる場合だけに限られます。それ以外の場合は簡単な事柄についても最低限自分のメモに基づいた伝達が必要であり、その内容についての相手の理解が正しいかどうかをその場で確認する必要があります。また、ある事柄について何度も複数の人から尋ねられたり、自分から言う必要性を感じたりした場合は、そのものごとについて複数の相手に理解が行き届いていないことになり、文書化・ドキュメントによる伝達が必要な時期が来たと考える必要があります。

Check List ＃２３　コミュニケーション（上級編）

▶Check Timing：会議・会話の前・中・後

□　相手が最も聞きたいと思われることを最初に伝えているか。

□　相手の能力レベルに合わせた話し方をしているか。

□　仕事の指示や依頼において、相手の理解度を確認しているか。

□　会話を深めるに、相手との共有体験を持つことや教養の幅を広げているか。

□　会社間の重要事項に関するコミュニケーションはリーダー対リーダーで行っているか。

＜コラム＃１５＞　相手が最も聞きたいと思われることを最初に伝えよう

質問に対しては、まずは相手が聞きたいことは何かを十分に把握し、それに対して直接関係のあることを答える必要があります。いろいろな条件や見解については相手の反応をみてから順次話をすればよいことです。これは報告書などのビジネス文書でも同様です。最初に相手が聞きたい結論を語ることが重要です。その結論の説明や条件などは、結論の後に書かれていればよいでしょう。

また口頭による会話では一度に多くのことを話したくなるものです。特に不意打ち的な質問を受けた場合、自分の中で正解を探す時間を稼ぐために色々と他の話題を出してしまうこともあるでしょう。このような場面を少なくするには、普段から仕事の懸案事項や問題点について良く考えておき頭の中で整理をし

ておく必要があります。目先の仕事に追われつつも、全体を見通すことを怠らないようにすることが大切です。会話や報告においては、まずは、相手が聞きたいことを先に伝えるように努力したいものです。

■ 会議に関するリスク

Check List ＃２４　顧客との会議のポイント

▶Check Timing：顧客会議前

□ 開発仕様の全体像の把握、仕様の目的・意味・背景を理解しておくこと。
□ 事前準備・事前検討を行い、顧客からの想定質問への答えを用意しておくこと。
□ 顧客からの宿題事項は、状況を聞かれる前に答えること。
□ 自分の言いたいことよりも顧客の話を良く聞くこと。
□ 疑問点・不明点については臆することなく、その場で確認を取ること。
□ リーダークラスにおいては、発表力やプレゼンテーション力を強化するために、普段の開発業務の中で発表やプレゼンの機会を自ら求めること。
□ リーダークラスにおいては、議事進行や調整能力を磨くために、社内会議等にて議長役や書記役を自ら買って出ること。

Check List ＃２５　会議運営のポイント

▶Check Timing：会議前

□ 会議の目的・背景・討議内容を事前に出席者に伝えているか。
□ 何を討議するのか事前に決めているか。
□ 何を決めるのか事前に明確にしているか。
□ 誰が・何を・いつまでに実行するのかを決めるようにしているか。
□ 不必要な人を招集してはいないか。
□ ダラダラ雑談に終了宣言を行うことができるか。

Check List ＃２６　日次情報共有会議のポイント

▶Check Timing：情報共有会議前

□ 昨日（ｏｒ今日）何を実行したかの報告を行うこと。
□ 今日（ｏｒ明日）何を実行するのかの予定報告を行うこと。
□ 今抱えている問題や困っていることを報告すること。
□ メンバーは、事前に報告事項を日報に記入する習慣を持つこと。
□ リーダーは、各報告に対して適時のアドバイスを行うこと。
□ リーダーは、全員に共有すべき情報を伝えること。
□ 長時間を要する問題は当事者とリーダーで別途打ち合わせをすること。

■ レビューに関するリスク

＜コラム＃１６＞　レビューの成否はドキュメントの出来栄えに依存する

レビューを実施する前に、効果的なレビューが可能になるような形式で各設計書を記述しておく必要があります。詳細設計書の項目と試験設計書の試験項目の対応づけが明確になるように両方のドキュメントを作成しておけば、評価テストレビューはスムーズになり、抜けも発見しやすくなります。 誤記、記入抜け、同じことを複数個所にバラバラに記述、分かりにくい冗長な日本語記述、などの設計書では効率的かつ正確なレビューを行うことはできません。過不足のない誰が読んでも同じ解釈が容易に行える設計書を用意する必要があります。レビューがうまく行かない最大の原因はまともな設計書などのドキュメントがないことにあり、その意味で開発ドキュメントの最初の出発点である要求仕様書および基本設計書の完成度の高さが全ての工程における効率性および品質の高さの原点となっているという認識が必要です。

Check List ＃２７ レビューの基本

▶Check Timing：レビュー前

- □ 重要機能順（顧客価値の順）にレビューを行っているか。
- □ 仕様および機能の目的・意味・背景を正しく理解しているか。
- □ 必要な仕様書や設計書および資料を用意してレビューを行っているか。
- □ 入力条件は正しいか、処理ロジックは正しいか、出力は正しいか。
- □ パフォーマンス・レスポンスの考慮は適切か。
- □ メモリーやＣＰＵ速度等のハードウェア条件の考慮は適切か。
- □ 想定仕様によって開発をしている部分はないか。
- □ 過去の類似の失敗例を参考にしてレビューを行っているか。

＜コラム＃１７＞　聞き上手、すなわち質問上手になること

聞き上手になることは難しいことです。何か疑問はありますかなどと言ってもなかなかメンバーは口を開きません。聞き上手になるためには、質問上手になることが一番の近道です。何か質問はありますかなどと言う漠然とした聞き方ではなく、○○についてどう思いますか？とか△△についての注意点は何でしょうか？分からなかったところや疑問に思ったことは何でしょうか？などと仕事内容の骨子の部分に沿った具体的な質問が効果的です。また条件付質問、例えば、仕様書にはＡの場合Ｂの処理をすることだけが書かれているが、Ａの条件が成立しない場合はどうすべきだと思いますか？などという質問の仕方も有効でしょう。仕様検討やレビューにおける重要なポイントは、部下がその仕様についてどれだけ正確に理解しているかを確認することです。そのためには部下からできるだけ多くの疑問点・不明点を聞き出す必要があります。「何か質問はありませんか？・・・何も無いようですのでこれで終わりにします」と言ういつものパターンは極力避けたいものです。

リーダーにおいては、部下に質問を投げかける前に、自分自身に同じ問いかけをしてみるとよいでしょう。

その自問自答にどれだけ正確に答えられるでしょうか。

Check List ＃２８　効果的なレビューのポイント

▶Check Timing：レビュー前

- □ レビューの時間を計画的に確保しておくこと。
- □ レビューは対象物の全件チェックではなく基本部のチェックに絞り込むこと。
- □ レビュー対象物の一覧表を作成すること。
- □ 過去のミス・不具合の傾向に基づいてレビューの重点を絞っておくこと。
- □ 基幹機能に関する基本的な理解が正しいか。
- □ 基幹機能の実現方法は適切か。
- □ 中間レビューを行うこと。
- □ 共通的なレビュー項目のひな型（チェックリスト）を作成しておくこと。
- □ 仕様変更の影響部分のレビューも行うこと。
- □ 担当者は、事前に自己チェックを済ませておくこと。
- □ レビュワーは、担当者が見落としがちな異常系、過去の評価モレ等の広い視点を持つこと。

＜コラム＃１８＞　結合テスト不良は、自分と相手のコミュニケーション不良に起因する

ソフトウェアのインターフェース不良の問題は、担当者同士の人間的なインターフェースの不良、すなわちコミュニケーションの不良に起因していると言えます。この問題は仕事における仲間同士のコミュニケーションや情報共有が徹底していないチームにおいて必ず発生します。ハードウェアに例えるなら、口径５ミリのナットとビスを作らなければならないのに、ナットの作成者は６ミリのナットを作り、ビスの担当者は４ミリのビスをそれぞれ勝手に作っているようなものです。それぞれの担当者の頭の中には、ただナットとビスを作ることしかなく、何ミリでなければ用を足さないのかという基本的な要件についての話合いや了解が抜け落ちています。とりあえず作ってみようという様な安易な考え方では、まともな結合は無理でしょう。

複数の担当者で組織的な開発を行う場合には、常にリーダーが指導力を発揮し全員の情報共有を継続的に行う必要があります。誰がリーダーだか分からないような集団は必ず失敗するでしょう。2003年にこれと同様な問題をＮ社が大規模システムのオフショア開発で発生させ、基幹モジュール群の結合に失敗し、結局国内で全て作り直し膨大な欠損を出したというニュースは記憶に新しいところです。

Last Point ： コミュニケーションのポイント

コミュニケーションのポイントをまとめると次のようになります。

□ 恥や失敗を恐れるより，行動しない自分を恥じること。
□ 人の好き嫌いではなく必要か否かで行うこと。
□ 礼儀知らずの相手には、ビジネスライクな対応を。
□ 相手に対する思いやりから始めること。
□ 思い込みの呪縛を解くこと。
□ 言葉だけでも，文書だけでも伝わらない。
□ コミュニケーションのルール・礼儀を守ること。
□ 情報共有とは密接なコミュニケーションのこと。

コミュニケーションを円滑に進めるために上記のことを実行すると同時に，コミュニケーションすなわち意思疎通の基本は，自分の都合よりも相手の都合を先に立てるということを忘れないようにしたいものです。

７．　情緒的開発姿勢のチェックリスト

First Point　：　本当の目標を目指して

「目標なんてわ分かっている」という声が聞こえてきそうです。でも，あなたやチームの目標はずいぶん前から「動くソフトウェアを作ること」だけになってはいませんか。動作するソフトウェアを提供することなど当り前のことです。本当に目指すべき目標のことも知らずに，一体みんなはどこに向かって歩いて行こうというのでしょうか。前回より良い品質指標・コスト指標・生産性指標などの目標は一体どこにいってしまったのでしょうか。

情緒的開発姿勢に関する主なリスクは次の通りです。

□　感情的・情緒的な思考や行動

□　数値目標の不在

■　情緒的思考行動に関するリスク

Check List　＃２９　感情本位から目的本位へ

▶Check Timing：モチベーション低下時

□　自分の関心を自分自身から仕事そのものに移すこと。

□　顧客の要求するものは何かということを良く理解し、その実現に向けて情熱を注ぐこと。

□　たとえ嫌いな相手であっても仕事に必要なことは必ず実行すること。

□　人の好き嫌いで自分の行動を変えず、誰に対しても一貫した姿勢と行動を保つこと。

□　過度な自己中心性を捨て、顧客要求の実現のためにチーム／メンバーに貢献すること。

自分を害するものを減らしたければ、これらのことを実行する必要がある。

＜コラム＃１９＞　感情本位から目的本位への転換を

あらゆる仕事において、人の好き嫌いを基準にしてしまうとものごとはうまく行きません。仕事は、人の好き嫌いなどという感情本位で行うものではなく、その仕事は何のために行うのかという目的本位で進めなければならないという認識を強く持つ必要があります。誰にとっても人の好き嫌いはありますが、仕事やメンバーの教育において、そのような感情的あるいは情緒的な態度では目的を達成することは不可能でしょう。仕事の仲間は、顧客の期待に応えるべく、好き嫌いという感情を乗り越えて、全員の思考と行動を集中して目的を達成する集団であり、決して仲良しクラブなどではありません。

人の好き嫌いで仕事を行ってきたことが、今までに自分にどのような不都合な結果をもたらしてきたのかを考えてみれば、このような姿勢は自分に益をもたらすよりも大きな害をもたらしているということが分かります。人の好き嫌いで仕事をやろうとする姿勢の原因は、自分の中の過度の自己中心性や精神的な幼弱性にあり、まずこれらを克服することから始めたいと思います。

Check List ＃３０　感情的･情緒的思考行動の解消

▶Check Timing：問題対応時

□ バグ発生に対して怒ることよりも冷静に原因追及および修復作業に努めているか。
□ 計画通りに物事が進まないことを自分やメンバーの性格のせいにしてはいないか。
□ 問題に直面した場合に合理的な判断ができているか。
　□ 今までの計画に想定違いや欠陥がなかったか。
　□ 計画を阻害するどのような新たな要因が発生したのか。
　□ 目の前のデータや事実から原因･理由を見つけて新たな対策を打つこと。

＜コラム＃２０＞　情緒性を排して積極的なコミュニケーションを

ベンダー側から追加テストの要求を受けても、その妥当性の判断がつかないために、不要だと思いながらも仕方なく要求に従っているような状況は少なくありません。しかしながら、反論できないとか、もめごとを避けたいためにテストをするというような情緒性に基づいた行動では負荷が増すばかりです。

このような場合、まずは自分が考える不要の理由を相手に言う必要があります。相手の意見に更に疑問があれば「何故？」の質問を自分が納得できるまで繰り返すことです。妥当な結論を見つけられるまで話をすべきでしょう。やはりテストが必要かも知れないし、テスト範囲が少なくなるかも知れないし、不要になるかも知れません。

次にやるべきことは仕様について完全な理解と把握をすることです。そのためには仕事に着手する前に、相手に対して直接的な会話およびＱ＆Ａ表などを通して不明点や疑問点を全て解消しておくことが必要となります。すぐに引き下がるような受身的な姿勢ではなく、分かるまでしつこく粘るような積極性が必要です。

このような積極的なコミュニケーションと合理的な仕事のやり方を通してしか自分のスキルアップは図れないでしょう。

■ 目標設定のリスク

Check List ＃３１　目標設定

▶Check Timing：開発着手前

□ 仕事をこなすだけで、目標の設定は不要だと思っていないか。
□ 現実的で達成可能なＱＣＤの目標値を設定しているか。
□ 目標は測定可能な数値で表されているか。
□ 目標は過去の実績を基準に設定されているか。
□ 目標達成のための具体的な手段を持っているか。
□ 目標値は実行手段に見合った数値になっているか。
□ 目標は、その変更も含めて、常時、チーム内の全員で共有されているか。

Last Point ： 目標の設定と実行

目標の設定および実行方法について以下に示します。

□ 失敗の振り返りにより QCD の目標値を見つけること。

□ 納期・コスト・品質はすべて同列の優先順位であること。

□ スローガンではない具体的な数値で示される目標の設定をすること。

（目標の条件）

□ 現実的で達成可能であること

□ 測定可能な数値で示されること

（目標の立て方）

□ 現状の問題点を洗い出し，その数値化を行うこと

□ 問題点の真因を明確にし，改善策を立案すること

□ 改善策の期待効果に見合った目標値を設定すること

□ 目標達成に向けて開発業務の中で改善活動を実行すること

目標の設定とはたとえで言えば，どこの山に登るのかを決め，それに必要な装備と体制を用意することに他ならない。エベレストに登るのと高尾山に登るのでは必要な装備もパーティの組み方も大幅に異なってくる。困難なプロジェクトを実行するに当たって必要な目標の設定をしないということは，エベレストにピクニック気分で行くのと同じことで、必ず遭難することになります。目標の設定はリーダーの重要な職責の一つです。

８．　モチベーション問題のチェックリスト

First Point　：　ヘタレのモチベーション

やる気やモチベーションは，自分が出すものであって誰かに出してもらうものではないでしょう。それが証拠に，会社から与えられるご褒美やインセンティブによって社員のモチベーションが継続的に維持向上したなどという話は聞いたことがありません。

本章では，仕事の重要な推進力であるモチベーションについて考えてみましょう。

＜コラム＃２１＞　過酷な状況を乗り越える方法

疲労困ぱいしている時は誰しもやる気が減退するものですが、それでもやり通せる人とそうでない人の差はどこから来るのでしょうか。過酷な状況を乗り越えるために必要な三つの条件は次のようでしょう。

【過酷な状況を乗り越えるための三つの条件】

□　完了時期を見えるようにすること。
□　不明点や疑問点の完全な解消を行い、作業内容および量を明確にすること。
□　一緒に乗り越えようとしている仲間がいること。

従来から抱えている開発問題を放置したままで、目標も設定せず、改善活動も実行せず、各自の協力・信頼関係も希薄なまま、流されるだけの仕事のやり方ではこれらの３つの条件を満たすことはできません。逆にやるべき内容を明確にでき、達成可能時期の見込みも立ち、仲間との連携もうまくかみ合っている場合は、疲労困ぱいはしていても、決してやる気とかモチベーションが落ちることはないでしょう。これらのやるべき事をやり尽くした上でこれ以上やりようがない状況が本当の限界と言えます。責任感の発揮とはこれらのやるべき事を実行することであり、死んでも頑張りますと言うような精神論的なものではありません。

Check List ＃３２　モチベーション

▶Check Timing：モチベーション低下時

□　仕事の意味を理解せずに形式的にこなすだけになってはいないか。
□　単純作業にも創意工夫を働かせているか。
□　意味や背景を考えて仕事をしているか。
□　言われたから仕方なくやると言うような、やらされ感的な仕事になってはいないか。
□　ＱＣＤの数値目標を持って、惰性に流されない仕事を行っているか。
□　仕事の量や質の全体を把握することで出口（完了時期）が見えるようにしているか。
□　業務中に頻繁にまたは長時間にわたって私的行為をしている者に警告しているか。
□　失敗やミスの原因を明確にし、歯止め対策を講じているか。

□ やる気のない者と同じ行動をしてはいないか。
□ 自分自身や他人を過剰にあるいは過少に評価してはいないか。
□ 個人に対して卑下あるいは侮辱などの行為をしてはいないか。
□ 他人より多いあるいは少ないという比較だけにこだわってはいないか。

Check List ＃３３　仕事への取組み姿勢

▶Check Timing：モチベーション低下時

□ 好き嫌いという情緒的な姿勢で仕事をしてはいないか。
□ 義務感を超えた誠実さをもって仕事に取り組んでいるか。
□ 仕事内容の意味や背景を知るために、“なぜ。”の問いかけを行っているか。
□ 事前準備・事前調査を行っているか。
□ 放置している案件がないか。
□ 複数の仕事を並列的に進める工夫をしているか。
□ 優先度を判断して仕事を進めているか。
□ データやドキュメントに拠る合理性に基づいた仕事を行っているか。
□ 人々の納得が得られるような妥当性のある仕事を行っているか。
□ 残務を先送りにしてはいないか。
□ 顧客からの電話に居留守を使うことはないか。

Check List ＃３４　困難な仕事への対応

▶Check Timing：モチベーション低下時

□ 自分で対策を考えずに、他人に解決策を頼ってはいないか。
□ 消極的ないしは逃避的姿勢になっていないか。
□ 何が困難で不安なのかを具体的にリストアップしてみたか。
□ 事前の学習をしているか。
□ 事前準備をしているか。
□ 仲間との連携を検討しているか。
□ 条件付行動を選択しているか。
□ オール・オア・ナッシング思考をやめ、合理的かつ妥当性のある行動を選択しているか。
□ チームでリスク解消に取り組んでいるか。
□ チーム内で責任分散を行っているか。

Last Point　：　モチベーション低下の歯止め

モチベーションを左右する主な要素には次の 2 つがあります。

□　自分が期待する獲得利益に合致しているか否か。

□　自分の安心・安全の保障が充足されるか否か。

モチベーション低下に歯止めをかける思考・行動には次のものがあります。

□　単純作業にも意味を見出すこと。

□　苦手・困難な仕事を克服すること。

□　強制に対抗する手段をもつこと。

□　約束違反はどこまでも許されるものではないと認識すること。

□　目標が見えるようにすること。

□　低評価に対して感情的な反応を抑えること。

□　価値が見出せないと言う前に相手の立場で考えること。

□　緊急な支援要請に対する実行条件を提示すること。

とくにモチベーションを著しく低下させる原因としては，「味方による裏切り」「約束違反」および「体調不良」の 3 つが挙げられます。味方による裏切りとしてもっともインパクトが大きいものは，上司の無関心さや裏切り等があります。部下が困難な状況に陥っているときの，「オレは知らないからな」とか「自分で何とかしろ」という言葉の一撃で，部下は復讐を固く心に誓うことでしょう。
「一緒に考えよう」の一言が言えるリーダーでありたいものです。

９．　本質把握ミスのチェックリスト

First Point　：　五里霧中に霞む本質

不合理な努力は報いられないと言われる通り、的外れな努力はいくら積んでも決して成果を得ることはできません。ある問題に取り組むにあたっては、その対象となる物事の本当の姿を把握しない限りは、その問題を解決することはできません。本章においては、ものごとの本質を見抜く方法について考えてみることにしましょう。

本質の把握ミスに関するリスクは次の通りです。

□　自分の頭で考えない自律性の未熟さ

＜コラム＃２２＞　アップル・トゥ・アップル比較　（Apple to Apple）

競合製品の比較分析においてＡ社の高級機種とＢ社の中級機種を比較しても意味はありません。競合製品の比較をするためには同じ価格帯同士の機能比較をすることで自社製品の優劣を判定することができます。これは当り前のことのようですが、ベースとなる条件が異なっているものを比較分析する過ちは意外と多いものです。また、他人の目をあざむくために恣意的にリンゴとミカンを比較するような人には要注意です。。

Check List　＃３５　本質を見抜く

▶Check Timing：問題検討時

□　自分の目で観察したか。
□　自分の頭で考え、判断したか。
□　複数の情報にあたってみたか。
□　見聞を深める行動をしているか。

Check List　＃３６　思い込みの解消

▶Check Timing：問題検討時

□　湧いた疑問に対して、何故。の問いかけをしているか。
□　根拠となる事実や定義に従って、原典を確認した開発行動を実行しているか。
□　変更の影響度を確認した上で仕様変更を行っているか。
□　常識に照らし合わせた理解や判断を行っているか。
□　仕様変更、日程変更等の重要情報は、原本配布／直接対話で伝えているか。
□　複雑な問題に対して、複数の解決策を比較検討しているか。

Check List ＃３７　誤解を避ける

▶Check Timing：問題検討時

□ 文書によらず口頭のみの情報で仕事をしている場合がないか。
□ 数値や論理記号に乏しいあいまいな日本語記述の文書を使っていないか。
□ 勝手な想定ではなく、定義されたドキュメントに基づいた開発を行っているか。
□ 仕様の目的・意味・背景を知った上で開発を行っているか。
□ 全体的視野、顧客の視点で問題を考えているか。
□ 他者の視点で考えてみたか。

Check List ＃３８　不明な問題を解く

▶Check Timing：問題検討時

（数学の考え方１７か条、秋山仁）
□ 分類や整理をしてみたか。
□ 図や表にしてみたか。
□ 簡単な模型を作ってみたか。
□ 基準をそろえたか。
□ 数学やロジックの言葉で表してみたか。
□ 小さな例で試してみたか。
□ 難しい問題は分割して考えてみたか。
□ 必要条件で絞り込んでみたか。
□ 特定の要素に注目してみたか。
□ 視点を変えてみたか。
□ 逆に考えてみたか。
□ 操作は１つずつ片付けてみたか。
□ 知っている事実を活用してみたか。
□ 規則性を探してみたか。
□ 対応をつけて考えてみたか。
□ 自然からヒントを得てみたか。
□ 部分から全体を把握してみたか。

Last Point　：　本質を見極めるために

本質を見極めるために必要な思考や行動には以下のようなものがあります。

□ 仕事や仕様の意味・背景・目的を知ること。
□ 仕事や仕様の事実の確認を根拠や証拠に基づいた文書・ドキュメント・ベースで行うこと。
□ 複数の視点で物事を見て分析すること。
□ 口頭・文書ともに必ず誤りが含まれるという問題意識を持ち、何故の疑問提起を行うこと。
□ 自己の役割・責任範囲を良く知ること。
□ 仕事の許容時間・必要時間を正しく把握すること。
□ 心身の限界を知ること。

あせりにあせって何の考慮もなく行動に移してもつまずくのがおちでしょう。事に対するあたっては必ず一呼吸の間が必要となります。この静寂の間こそが、本質を見抜くための「なぜ？」を発するための時間となります。

１０．　優先順位問題のチェックリスト

First Point　：　優先順位の三つの視点

作業の優先順位の決定にあたっては、複数の作業の緊急度・重要度の優先順位判断の視点、および手持ち時間と必要時間の比較判断の視点、および一つの作業内における処理の順番の妥当性の視点、の三つの視点での検討が必要になります。

この三つの視点における判断ミスおよび必要時間の確保の失敗は、ＱＣＤを大きく傷つける結果を招くことになります。

優先順位認識に関する主なリスクは次の通りです。

□　割り込み作業等の優先度判断。

□　必須業務に優先順位はないという認識の欠如。

□　納期第一優先の誤り。

Check List　＃３９　優先順位の設定

▶Check Timing：作業着手前

□　仕事の意味・意図・背景を最初の時点で正確に把握すること。

□　仕事内容をブレークダウンすること。

□　ブレークダウンした仕事内容に優先順位を設定すること。

□　タイムリミットが直近に迫っているものから処理をすること。

□　時間的余裕があるものは重要機能の順に処理すること。

□　優先順位が判断できない場合は早めに上長に相談すること。

□　この仕事に許される自分の残り時間を毎日意識すること。

＜コラム＃２３＞　顧客価値の優先順位を知る

顧客が何に優先順位を置いているのかを知るためには、まず顧客の立場になって顧客と密接なコミュニケーションをとり、次にプロとしての常識力を発揮する必要があります。

たとえば、ある顧客の店舗システムのソフトウェアを更新したところ不具合が発生し、顧客からはすぐに対処するようにとの矢の催促。パニックになった担当者は必死になって不具合の原因調査を始めた。その間、店舗システムはバグを抱えたまま稼動していた。最初にやるべきことは、バグの修正ではなく、店舗システムを元の問題のないバージョンに戻すことにありました。

Check List ＃４０　仕事の優先順位認識

▶Check Timing：作業着手前

□　ＱＣＤ（品質、コスト、納期）の優先順位は同一であることを理解しているか。

□　標準的なプロセスで定義されている作業は全て必須であることを理解しているか。

　＊仕様の不明点・疑問点の事前調査や各工程のレビュー等は必須業務である。

□　顧客や上司からの要求が常に第一優先とは限らないという認識を持っているか。

□　自分の責任範囲を超える優先順位の変更は上司の承認を得ているか。

□　要求内容の優先順位を顧客に確認しているか。

□　タイムリミットが迫っているものを第一優先にしているか。

□　顧客価値の重要度順で仕事を進めているか。

□　緊急度・優先度の決定にあたって、要求者あるいは指示者との調整ができているか。

□　中断された仕事について、再開に必要な事項のメモを残しているか。

□　割り込み作業は必ず発生するという認識で、作業の前倒しを行っているか。

＜コラム＃２４＞　鶴の一声、何度も鳴けば効き目なし

仕事の優先順位は往々にして外部の有力者からの鶴の一声で入れ替えさせられます。合理的な理由もなければ妥当性も見当たらない場合ですら、みんなこの一声には異常に弱いのです。一声で済めばまだ良いですが、度々鳴かれると仕事が進みません。余りにもひどければ、何度も鳴くなと鶴に言う必要があります。

Check List ＃４１　割り込み作業対応

▶Check Timing：作業着手前

□　全ての工程において、割込み作業や不測の問題は必ず発生するという認識を持つこと。

□　顧客との会話を絶やさず、早い時点での割り込み要求情報をキャッチすること。

□　全ての工程のスケジューリングは必ずバッファを持たせること。

□　バッファを確保するために、常に業務の改善や効率化を行うこと。

□　その場で内容を聞く余裕がまったくない場合は後刻の時間を指定して後で聞くこと。

□　現状の仕事と割り込みの仕事についてチーム全体としての優先順位を考え、緊急度・重要度の高いものを優先させること。

□　割り込みにて中断された作業については、中断情報を記録しておき再開に備えること。

□　自分の担当業務に割り込ませることが不可能と判断した場合は、他の人に割り振ることや他の代替案を示すこと。

□　割り込み対応が全く不可能な場合や開発工程の後半での仕様の追加および変更は、次回リリースにしていただくように交渉すること。

＜コラム＃２５＞　溺れるものは必ず藁をつかむ

焦っている場合，本来の段取りを無視して枝葉末節なことがらに先に手をつけがちになります。これがかえって仇になり，根幹の問題の進捗を阻害したり，先に手をつけたものがまったく無駄になったりすることがあります。切羽詰ったときにつかんだものはやはり藁が多いようです。

Last Point　：　優先順位の意味

優先順位の決定とは、平たく言えば、何の仕事を先に済ませるかということになりますが、簡単なようで実はそう簡単な話しではありません。優先順位という言葉は、作業手順の順番の決定および優先順の決定という二つの意味を含んでいます。

作業手順について、Ａ→Ｂの順でなければ正しい結果が得られないものを、先にＢから着手し失敗してしまう例としては、仕様未決定状態における事前着手問題などがあります。

また割り込み要求などで複数の作業を抱えてしまった場合の優先順位の決定においては、緊急度および重要度の順に処理順位の決定が必要になりますが、時間の調整や交渉に失敗した場合、不十分な作業内容による品質問題を起こしてしまうことになりかねません。

上記二つの問題に共通していることは“時間不足”という問題です。手持ち時間が不足した結果、あせって事前着手に走り多くの手戻り作業を発生させ、また複数の仕事を同時に処理できず不十分な作業内容による品質低下を招くことになります。

優先順位の決定は、同時に必要時間の獲得が必須条件となることを忘れないようにしましょう。

１１．　ドキュメント・ベース開発欠如のチェックリスト

First Point　：　ドキュメントこそが開発業務の妥当性の証拠

適切な要求仕様書や設計書に基づかない開発業務の妥当性を証明する方法はありません。多くのプロジェクトの実態は、二転三転する要求仕様のために、まともな要求仕様書も設計書も作れず、とりあえずヒアリングや議事録メモベースでプログラムコードを作るのが精一杯なようです。設計書はプログラム作成後に後付けで修正され、ドキュメントとしての価値を失っています。さらにひどい場合は、ドキュメントはメンテナンスもされずに放置されている場合もあります。

ドキュメント・ベース開発の欠如に関する主なリスクには次のものがあります。

□　開発行為の科学的根拠（ベースライン）の喪失
□　低品質なドキュメントが招くＱＣＤの失敗
□　文書化能力の低下

＜コラム＃２６＞　主要工程におけるミス対策のポイント

製造品質が悪い原因は、粗悪な詳細設計書、ソースコードのレビュー不足、製造担当者のコーディングミス、単体試験仕様書の作成時期の遅れ、などが想定されますが、実際に製造品質の悪さを原因別に分析し数値化したものがなければ、何を重点的に対策しなければならないかを決めることはできません。単体テスト結果における不具合の原因を、例えば、要求仕様ミス３０%、基本設計ミス（要求仕様の理解ミスも含む）２０%、詳細設計書ミス３０%、製造ミス２０%だったとしたならば、さらにそれぞれの工程におけるミスの原因を分類して、誤記・抜け・あいまいさ・誤解・言語能力不足・論理の間違い等の詳細原因にまで分解する必要があります。これらの分析ができて初めて具体的な対策が可能となります。

工程別のミスに対する対策を下記に示します。

□　要求仕様ミスに対しては要求仕様書内容の正確な記述、適切な時期における凍結および効果的仕様レビューや仕様検討が必要。
□　基本設計ミスに対しては要求仕様の理解力の向上、基本設計書記述能力の向上および効果的基本設計レビューの実施が必要。
□　詳細設計書ミスに対しては基本設計書の理解能力の向上、詳細設計書の記述能力の向上および効果的詳細設計レビューの実施が必要。
□　製造ミスに対しては、言語能力の向上、プログラミング技術の向上および効果的ソースコードレビューの実施が必要。

Check List ＃４２　ドキュメント・ベース開発の遂行

▶Check Timing：各工程開始前

□ 基本部の要求仕様書は、見積り時に提示されること。
□ 詳細要求仕様書は妥当な時期までに凍結されること。
□ 要求仕様書に基づいて基本設計書が作成されること。
□ 基本設計書に基づいて詳細設計書が作成されること。
□ 詳細設計書に基づいてプログラム作成が行われること。
□ 詳細設計書に基づいて単体テストチェックリストが作成されること。
□ 単体テストチェックリストに基づいて単体テストが行われること。
□ 基本設計書に基づいて結合テストチェックリストが作成されること。
□ 結合テストチェックリストに基づいて結合テストが行われること。
□ 要求仕様書・実運用シナリオ資料に基づいて総合テストが行われること。

Check List ＃４３　ドキュメント更新

▶Check Timing：変更・修正発生時

□ 仕様調査時のＱ＆Ａ資料の内容は都度、設計書等に反映すること。
□ 仕様書・設計書等は、テスト等での不具合修正の都度、更新すること。
□ 仕様書・設計書等は、欠陥が発見された都度、更新すること。
□ 仕様書・設計書等は、派生開発の都度、更新すること。
□ ソフトウェア構造設計書は、派生開発の都度、明確になった部分を更新すること。
□ 仕様変更影響度表は、仕様調査情報および評価テスト情報に基づき更新すること。

＜コラム＃２７＞　ドキュメント・ベース開発を支えるチェックリスト及びガイドライン

複雑多岐にわたる開発業務の失敗を少なくし、業務効率を高める手段として各種のチェックリストやマニュアル、手順書、ガイドラインが必要となります。 開発および評価テストに関して以下のチェックリストやマニュアル、手順書は必須のドキュメントだと言えます。

□ 要求仕様書チェックリスト、要求仕様書ガイドライン
□ 見積り回答チェックリスト、見積り回答ガイドライン
□ 仕様調査チェックリスト、仕様調査ガイドライン、仕様Ｑ＆Ａガイドライン
□ 設計チェックリスト、設計ガイドライン、評価設計ガイドライン
□ コーディング・チェックリスト、各種言語のコーディング・ガイドラン
□ テスト・チェックリスト、テストガイドライン
□ 開発ツールマニュアル、評価ツールマニュアル
□ 構成管理チェックリスト、構成管理ガイドライン

Check List ＃４４　ドキュメント・ベース業務の遂行

▶Check Timing：決定事項発生時

□ 顧客会議／連絡における依頼・指示・決定内容は議事録にて管理されているか。
□ 関係各社との会議／連絡における依頼・決定内容は議事録等にて管理されているか。
□ 社内会議／連絡における依頼・決定内容は議事録等にて管理されているか。
□ 顧客・ベンダーからの仕様変更は、仕様変更管理表にて管理されているか。
□ 依頼・指示・決定内容等は、「誰が、何を、いつまでに、どのように」が明記されているか。

Check List ＃４５　ドキュメントの見える化のポイント

▶Check Timing：ドキュメント作成時

□ 書類の品質の均一性保持のために、統一的にフォーマット化すること。
□ 仕様や機能の目的・背景・意味を記述すること。
□ 論理記号、数学的表記、図・表・フローチャート等を使用し、論理的な記述をすること。
□ 複雑な事象表現にマトリクス図や共通パターン表などを使用すること。
□ 異常系処理を記述すること。
□ 重複を避け、シンプルな記述にすること。
□ パフォーマンス・レスポンス条件を記述すること。
□ 仕様変更による修正影響度表を作成すること。
□ 誤記、抜けおよび変更は発見・発生と同時に更新すること。
□ 開発ドキュメント内容とプログラムコードは、常時一対一で整合性を保つこと。
□ 複数の仕様書間、設計書間の矛盾した記述をなくすこと。

＜コラム＃２８＞　単体テスト用チェックリスト作成に関するポイント

有効な単体テストチェックリストを作成するには、その元資料である詳細設計書の精度を高くする必要があります。当然のことながら、詳細設計書はプログラム作成前に完成され、プログラミングは詳細設計書に基づいて実行されなければいけません。つまり精度の高い詳細設計書がなければ、精度の高いプログラミングも単体テストチェックリストも作成はできないと言うことです。 さらに詳細設計書の精度は基本設計書の精度に左右され、基本設計書の精度は要求仕様書の精度に依存しています。これらの上流工程における技術ドキュメントが整備されていなければテスト工程における効果的なチェックリストの作成は困難です。 その時になってあわててチェック項目をひねり出そうとしてもどこからも湧き出してはこないでしょう。

時間がないから精度の高いドキュメントを作る暇がないと言う人においては、「Check List ＃７３　自分の時間を確保するためのポイント」および「Check List ＃７４　自分の時間を生み出すためのポイント」を参照してください。

Check List ＃４６　業務文書作成のポイント

▶Check Timing：業務文書作成前

【準備】

□　書く前に、相手に伝えたいことを重要なもの順にメモに書き出しておくこと。

【受け取る側の利益・心情を優先】

□　最初に結論の記述から始めること。相手の要望や疑問にまず答えること。

□　自分の言いたいことを優先しないこと。特に不始末の詫び状などでは言い訳から記述を始めるのは禁物。

【分かりやすさ】

□　報告書は基本的に最初の１ページで全貌が分かるように書くこと。

□　文章の主題・件名と本文の内容を合致させること。

□　結論に始まり、続いてその経緯について簡略にまとめること。
どんなに長い内容でも本文は１～２ページにまとめ、詳細内容は添付資料とすること。

□　文章は重要なものの順に書くこと。

□　一つの文章はできるだけ短くすること。複雑な内容は複数の文章に区切って書くこと。

□　箇条書き等によって簡潔な表現に努めること。

□　同じ内容を、形を変えて何度も繰り返さないこと。

□　重要な事とそうでないことを峻別すること。

【正確性の確保】

□　「誰が、何を、いつまでに、どのように、どれくらい」等が明確に記述されているか。

□　数値やデータによって質や量を表現することで、文書の客観性が保たれているか。

□　あいまい表現はしない。例えば、”非常に”とか”おおよそ”などの表現は厳禁。

□　文章だけで説明が難しいものは、マトリクス・図・表などを使用すること。

□　技術用語は、顧客・プロジェクト・社内で統一用語を使用すること。

＜コラム＃２９＞　日本語文章のあいまいさを避ける

論理性や正確性が要求される業務文書においては，日本語の豊かな情緒性やあいまいな表現が大きな障害となっており，以下に示した注意点を押さえておく必要があります。

- □ 主語がない文章には主語を書き加えること。
- □ 可能な限り短い文にすること。
- □ 並列的な表現は箇条書きにすること。
- □ 時間軸にて整理したチャートで現在・過去・未来の時制を明確にすること。
- □ 読み聞かせで聞き手がきちんと理解できるか確認してみること。
- □ 単位・期限・期間・対象範囲などの基準を明確にすること。
- □ %表記は何を 100%としたのかを明記すること。
- □ 程度表現（例：すごく，適当，ちょうど，およそ）を避け数値で表すこと。
- □ 強調表現（例：ちゃんと，しっかり）を避け数値で表すこと。
- □ 推測表現（例：～くらい，およそ～）を避け数値で表すこと。
- □ その他の曖昧表現（霊：～の予定，できるだけ～）を避けること。

日本語と英語の違い

日本語	**英語**
□あいまいで明言を避ける	□直接的，断定的な表現
□省略語（主語などの省略）が多い	□省略語が少ない
□文が長くなりがち	□比較的簡潔
□無生物は主語になりにくい	□無生物でも主語になる
□結論（話題の中心）が最後	□結論（話題の中心）が最初

Last Point　：　エンジニアリング

ソフトウェア開発の生産性および品質の向上を図るためには、ドキュメントに基づいた開発を実行することが最も効果的な方法です。時間がなかったからドキュメントをメンテできなかったのではなく、ドキュメントをメンテしなかったから時間がなくなったのだと言う認識の転換が必要です。ぼろぼろの仕様書・設計書による仕事などにエンジニアリングという科学性の一片すら感じることは誰においてもできることではありません。

１２．　実務能力問題のチェックリスト

First Point　：　自分の頭で考える自律性こそが開発の基本

開発者に求められる基本的な能力の筆頭として、自律性があります。

自律性とは、自分が遭遇した問題に対して、自分自身の観察力で判断し自分の力でそれを解決していく能力のことを意味します。自律性の発揮は、自力更正とか自助・自主自立という形で現れてきます。この自律性の力は人間の思考や行動のベースとなる重要な能力です。

自律性の欠如は、開発の実務行動においてさまざまな問題を広範囲に起こす原因となっています。

実務能力に関する主なリスクは次の通りです。

□　知識不足（技術、仕様、システム構造、評価テスト）

□　仕様決定能力の低さ

□　設計・製造能力の低さ

＜コラム＃３０＞　地道な日々の問題・リスク情報の共有活動を

プロジェクトにおけるリスクの回避やソフトウェアの不具合問題などを一度に解消できる方法などは存在しません。開発業務の進行と同期をとったリスク対応や不具合傾向の把握を可能にするには、それなりの準備が必要となります。

リスクについては、リスクが問題化した時点および対策を実行した時点においてその要因などの情報を記録しておき、不具合についても、発見および修正時に不具合票にその真因や発生工程などの情報を同時に記録しておく必要があります。これらのことを全員が都度こまめに実行していれば、これらの累積されたデータは常時グラフや表として分析することが可能となります。

これらのことを実行せずに、発生不具合やリスクを一時期にまとめて分析することなど不可能なことです。

リーダーにおいてはプロジェクト全体の、メンバーにおいては各自の担当部の不具合発生・対策状況、リスク管理表の更新、課題管理表の更新を毎日ベースで登録する必要があります。そうすれば作業の進捗に連動した効果的な対策を打つことが可能になります。何日も更新されないスケジュール表・不具合管理表・リスク管理表・課題管理表は何の役にも立たないただの紙くずになってしまいます。さらに各プロジェクトの終了時に必ずプロジェクト完了報告書にてＱＣＤ（品質・コスト・生産性）の振り返り、および重要不具合・重要リスク・重要課題について数値に基づいた振り返りをしておくことが、次のプロジェクトにおけるＱＣＤおよび効率化・生産性向上につながります。

■ 基本的な開発能力

Check List ＃４７ ソフトウェア開発チームに必要な能力

▶Check Timing：開発着手前

- □ チームにおける日常的で密な直接コミュニケーション力
- □ ドキュメントの常備はもとよりちゃんと動作するソフトウェアの素早い開発力
- □ 顧客との日々の直接コミュニケーションによる協調
- □ 変更要求に対する俊敏な対応力
- □ 顧客価値優先度の把握力
- □ 意欲ある人材で構成されたプロジェクト構築力
- □ 短期開発力
- □ 一定のペースを持続できる開発力
- □ 技術的な優位性の保持、良好な設計力の保持
- □ 無駄・不要な仕事の削減力
- □ 自律的チーム力
- □ 定期的・効果的振り返りによる行動の変更・調節力

■ 緊急・危機対応

Check List ＃４８ 緊急対応時の行動原則

▶Check Timing：緊急対応前

- □ まずメンバーを集め、データを集めること。
- □ 集めたデータを分析・統合しその意味を判断すること。
- □ 対策案を決め、実行すること。
- □ コストよりも時間を優先すること。
- □ 複数の緊急事項が重なった場合に時間的優先度を考慮すること。

Check List ＃４９ 危機的状況からの脱出

▶Check Timing：プロジェクト危機時

- □ 直ちに関係者全員を招集すること
- □ 各人が保有する情報を全て一箇所に集めること。
- □ 問題となる情報を整理分類し、可能な限り仕事の全体像（最終目標）を見えるようにすること。
- □ 最終目標に対して自分たちがどの地点まで到達しているのかを想定して見ること。
- □ 目標と乖離している残件項目を明らかにし、必要な工数の概算を見積ること。
- □ 各担当の残件負荷のバランスを比較し、役割を再配分することで平準化を図ること。
  現有勢力で対応が不可能と判断したら必要な人材と人数を投入すること。
- □毎日の短時間情報共有会議を行い、各人において「①今日やったこと ②明日やる予定
  ③今抱えている問題」の三点につき報告をおこない、リーダーは適時アドバイスを行うこと。

■ プロジェクト管理のリスク

Check List ＃５０　進捗管理

▶Check Timing：進捗管理時

□ プロセス管理表にて、主要イベントの時期、責任部署、提出物、受領物を管理すること。

□ 進捗管理表にて、主要日程および詳細業務の優先順位を考慮した管理を行うこと。

□ コミュニケーション管理表にて、関係者間の打ち合わせ等が予め計画され、予測される全ての問題リスクの解消を行うこと。

□ 顧客提出用スケジュールと開発内部スケジュール間に矛盾や食違いがないこと。

Check List ＃５１　ＱＣＤ（品質･コスト･生産性）管理

▶Check Timing：ＱＣＤ管理時

□ プロジェクト毎に、過去の品質数値と現在の品質数値を管理しているか。

□ 目標品質数値と実績品質数値の±差異の原因分析および対策を実行しているか。

□ プロジェクト毎に見積り提示金額（工数）と実績金額（工数）を管理しているか。

□ 見積り額と実績値の±差額の原因分析および対策を実行しているか。

□ プロジェクト毎に、過去の生産性数値と現在の生産性数値を管理しているか。

□ 目標生産性数値と実績生産性数値の±差異の原因分析および対策を実行しているか。

Check List ＃５２　品質の妥当性確保のポイント

▶Check Timing：品質目標設定前

□ 品質が根拠のある数値で表されているか。

□ 現在の品質数値と目標数値の設定がなされているか。

□ 納期優先という名目により品質が犠牲にされていないか。

□ 仕様変更の影響度調査を行っているか。

□ 開発規模に見合わないプロセス管理（ＣＭＭＩ）が強制されていないか。

　□ 必要最小限に絞った簡易型ＣＭＭＩプロセスの作成･運用

　□ 自社の経験則に基づくプロセス管理表の作成･運用

＜コラム＃３１＞　事前準備をやり尽くすこと

納期優先の結果、最初に犠牲にされるのは間違いなく品質です。納期優先の考え方の行き着く先はバグ多発あるいは機能しないシステムです。多くの開発者においては、このような実例を何度も見聞きし、体験しているはずでしょう。無理やり納期を守ったとしても、劣悪な品質の製品をリリースすることのどこに意味や価値があるのでしょうか。このことは誰にも分かる簡単な道理です。しかしながら、なぜ開発チームやリーダーはそのような状態に追い込まれてしまうのでしょうか。いつも通りに普通に開発を行っていれば、ソフト開発はうまく行くとでも思っているのでしょうか。プロジェクトのリスクについて開発開始前に真剣に考

えるリーダーは少数です。早期の仕様凍結を目指すリーダーはもっと少数でしょう。失敗プロジェクトの真因を追究し再発防止を決心するリーダーはさらに少数です。一旦、納期が非常にきつい状態に陥ってしまうと回復はほぼ不可能と考えた方がいいでしょう。リーダーの頭に"納期優先"という言葉が浮かんだ時点で、そのプロジェクトはほぼ失敗に向かって進んでいるのです。プロジェクトの成否は、最初の見積りと要件定義によってほぼ決定しています。誤解を恐れずに言えば、プロジェクトの成否はプロジェクトの開始時点でほぼ決定しているのです。

プロジェクトを成功させたければ、プロジェクト開始前後における事前準備に力を注ぐことです。すなわち、過去の失敗に学び同じ過ちやリスクを絶対におかさないための改善活動の実行、妥当な見積りによる妥当な開発費と期間の獲得、早期の仕様凍結による開発内容の全貌の把握などを段取り良く実行すること以外に方法はありません。

プロジェクト開発におけるＱＣＤ（品質・コスト・納期）は皆第一優先事項であって、どれか一つが欠けても失敗プロジェクトであるという強い認識をもつ必要があります。人間システムに例えれば、脳・心臓・肺のどれか一つでも機能しなければ死に至ることと同じです。

■ 生産性に関するリスク

Check List ＃５３　仕事のスピード・効率

▶Check Timing：開発着手前

□ 事前準備を行っているか。
仕事にとりかかる前に、その仕事の内容の調査や全体像把握の実行、その仕事の段取りの計画、その仕事のリスクの把握などを実行すること。

□ 仕事の最中においてはＱＣＤの数値データを記録しているか。

□ 振り返り（ラップアップ）を実行しているか。
仕事が終了した時点で、仕事中に起こしたミス、失敗についての真因を把握し再発防止策を立て、次の仕事に備えること。

□ 以上のことをすべて記録しておき、ものごとを数値で語れるようにしているか。

□ 仕事内容を完全に把握しておらず、不明点や疑問点を放置してはいないか。

□ 仕事を実行するために必要な知識および能力に不足はないか。

□ 仕事の質および量の見積り間違いはないか。

□ チーム内でのコミュニケーション不足による情報不足はないか。

■ 評価テストに関するリスク

Check List ＃５４　単体･結合テストの効果的やり方

▶Check Timing：単体･結合テスト前

□ 製造の初期段階でソースレビューを実施し単体･結合テストの負担を軽くしておくこと。
□ 製造時のデバッグの質を上げること。
□ 流用可能なチェックリストは流用する。但し、内容を良く見極め単なるコピペはしないこと。
□ テスト項目は必ずテスト仕様書を作成して行うこと。
□ テスト実施項目の要･不要を精査すること。
□ 設計書の機能項目とテスト仕様書のテスト項目の“対応ひも付け”をすること。
□ テスト結果のエビデンス（印刷媒体）や整理は電子的自動化ツールにて行うこと。
□ 自動化できるテストは全て自動化すること。
□ テスト結果のレビューを必ず行うこと。
□ 日次会議にて不具合傾向を監視し、注力が必要な項目とそうでないものを峻別すること。

＜コラム＃３２＞　やるべき事をやるべき時に

無定見なベンダー側担当者は、単体テスト時に結合テスト項目を、結合テスト時に総合テスト項目を要求してくることがあります。スケジュールの前倒しや「念のため」とかの理由が多いようですが、常識的に考えれば、基本的なテストが済んでいないものを、さらに上級のテストをしてもうまくいく訳もありません。このような何の合理的な根拠もないような要求に従うべきではありません。 さらに単体内における変更がその他に及ぼす影響範囲は限定的であり、「念のため」の単体テストなどあり得ません。単体内の影響部を特定し、そのためのテストを実施するのが単体テストの目的です。単体モジュールが外部に対する影響は結合テストの役割です。また結合レベルでも特定できない影響範囲については総合テストの役割です。どうしても特定できない影響範囲に関しては総合テストにおける実運用テストを実施すれば問題の有無ははっきりするでしょう。

やるべき事をやるべき時に実行しなければ、その工程の仕事は非常に不完全なものとなり、次工程の仕事をさらに混乱に陥れる結果となり、スケジュールの前倒しどころか遅延の原因を自分で作っているようなものです。

＜コラム＃３３＞　バグ認識の甘さ

不具合の区分において”改善要望”の区分は客先が実施する本部検収テストや運用テストにおいてのみ使用される区分で、いわゆる利便性に関する問題です。例えば仕様通り動作しているが操作性が悪いとか見栄えが悪いとか言うような問題に相当します。

一方、開発チーム内のテスト工程において”改善要望”などの区分は設けるべきではありません。要求仕様に外れている問題は全て不具合として処置すべきです。開発チームは“改善要望”などと言える立場にはありません。判断が付かない場合は、要求仕様作成者に問い合わせるべきであり、不具合修正

を嫌うあまりに勝手な自己判断で“改善要望”などという区分を作るべきではありません。

Check List ＃５５　総合テスト（システムテスト）の要件

▶Check Timing：総合テスト前

□　総合テスト仕様書およびチェックリストは、要求仕様書および設計書の内容を過不足なく反映し作成されたか。

□　総合テストは、総合テスト仕様書およびチェックリストに基づいて実施されたか。

□　要求仕様書および設計書に規定された仕様通りに全ての機能が動作したか。

□　全ての機能は、操作手順に従い、所定の入力条件に対して正しい出力が得られたか。

□　実運用テスト用のシナリオ・テスト資料は作成されたか。

□　シナリオ・テスト資料に基づいたテストは実行されたか。

□　顧客の期待する実運用の役割通りに、システムが動作するのを確認したか。

＜コラム＃３４＞　総合テストはバグ出し工程ではなく、仕様確認の工程である

最後の工程である総合テストは品質の最終的な歯止めですが、当然前工程で不具合が処理されればそれだけ工数も時間も大幅に節約できます。現状では、要件定義工程や設計工程起因の不具合が最も多く、製造工程自体のミスはそれほどではないと推定されます。総合テストで発見される不具合に関して、それらの発生元工程の件数や発生率のデータを示し、各工程の責任者に改善実行を促す必要があります。総合テストはあくまでも仕様確認のためにあり、不具合に関しては最後の万が一のための歯止めであると言う認識を上流工程のリーダーたちは持つ必要があります。苦しい開発をしないためにも、品質問題で顧客の信頼や自社の利益を失わないためにも、プロジェクトリーダーのもとに各工程のリーダーたちで知恵を出し合い、品質改善活動を日常的に継続実行する必要があります。総合テストは、本来はバグ出し工程ではなく、システムが仕様通りに動作することを確認する仕様確認工程なのです。

＜コラム＃３５＞　不具合発生の傾向分析から新たなテストパターンの追加を

過去にも品質問題を抱えているシステムの再評価が色々なプロジェクトで行なわれたにもかかわらず、闇雲に行なうテストでは何も成果が上がりませんでした。

過不足のないテストパターンによる効果的なテストを行うためには、評価対象機能の完全な理解およびその機能開発による影響範囲の完全な把握が必要となります。平均的な評価チームにおいては、開発された機能についての一定の理解は可能だですが、その機能の背景や実運用に関する理解は開発チームと同様に非常に乏しいと言わざるを得ません。また開発機能の影響範囲の特定に関しても構造設計書の不備のために非常に限定的になっています。このような不利な条件下において妥当性のある評価を行う方法としては、基本的なテストパターンを実施しながら、発見される不具合の場所や特徴および傾向などを日々分析することで、それらの傾向に沿った新たなテストパターンを加えていくやり方が有効です。不具合の傾向分析を行わずに単に希望的観測に基づいた闇雲なテストはほとんど無駄でしょう。

更に、○○の仕様に対しては△△のテストが有効だというテストパターンに関する経験知を評価チーム

で積上げ、データベースとして蓄積するような活動が必要となります。

＜コラム＃３６＞　回帰テストの自動化を

派生開発においては仕様変更に影響しないと思われる既存機能部分に対する回帰テストがコストおよび時間の重荷となっています。いわゆる念のためのテストです。

回帰テストの範囲を決めることは難しく、手を加えたソフトウェアの回数、階層の深さ、影響領域の広さ、などに比例して回帰テストに必要な量と時間は増えていきます。

正確な影響度範囲を特定できれば良いですが、複雑なソフト構造ではこれにも限度があります。従って気になるところは全部やり直しという、力ずくの回帰テストにならざるを得ません。その結果、回帰テストは余り実行されていません。そういう訳で現在取れる現実的な手段は回帰テストの自動化しかありません。自動化にあたっては、そのシステムで行われる全ての取引および上位システムとの通信処理に関して一連の操作を全てパターン化および自動化し、開発の前と後における出力データの比較を自動的に行える仕組みが必要となります。

Check List ＃５６　構成管理のポイント

▶Check Timing：製品リリース前

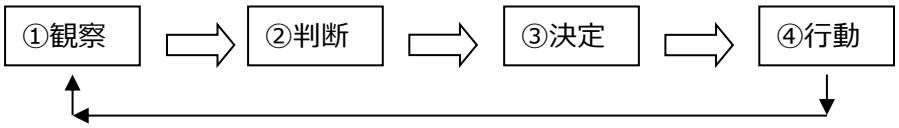
□　人の判断や操作に依存する管理を極限まで減らすこと。

□　開発中の成果物と中間リリースする成果物を管理するＰＣを分けること。フォルダを分けるだけではまだ人の判断ミスを誘発する可能性がある。

□　リリース前に、作成したメディアによる実際のインストレーションおよび基本仕様の動作確認テストを行うこと。

Last Point　：　自律的思考・行動のサイクル

自律性の思考・行動のプロセスサイクルは次のフローで表すことができます。



①目前の問題や課題などの現実を自分の目で観察する。
②観察した状況を自分の頭で判断し、意味を発見する。
③判断の内容に従って自分の意思で対応策を決定する。
④決定した対応策を自分で主体的に行動に移す。

上記プロセスは現実の問題や課題が解決されるまで何回も繰り返す必要があります。競争においては、このプロセスサイクルの回転速度が速いものが必ず勝利する結果となります。

１３．　リーダーシップ問題のチェックリスト

First Point　：　プロジェクトをマネジメントするということ

マネジメントをするということは，みんなの先頭に立って道を切り開き，障害物を取り除き，敵の盾となり，進むべき方向を指し示し，チームのメンバーが最大限の力を発揮できるようにすることです。兵たちの後方に立って全員突撃と叫ぶ将校ではなく，俺に続けと先頭を駆け抜けるリーダーになりたいものです。

リーダーシップ問題のリスクは次の通りです。

□　マネジメントされていないプロジェクト管理

＜コラム＃３７＞　プロジェクトに余裕を生み出す方法

時間や心の余裕のない場合は、他人に対しての誠意のある対応が難しいだけではなく自分の仕事においてもミスを出しやすくなります。自分の実力以上の仕事量だと感じた場合は速やかに上長と相談する必要があります。また年中仕事に追われて全く余裕のない状態が続いているのならその原因について考え、対策を講じる必要があります。仕事を他の人に丸投げしなければならないほど余裕のない状態に自分をおくべきではありません。丸投げはＱＣＤすべてをだめにするだけではなく信頼関係を失う原因にもなります。仕事に余裕を生み出すためには以下のことを実行する必要があります。

１．振り返りの実行

仕事の終了時に必ず自分および他人の失敗の真因を明確にし、再発防止のアクションを行うこと。再発防止は改善活動として組織的な取り組みを行うこと。

２．事前準備の実行

次に着手する仕事の事前準備として開発内容および段取りを明確にしておくこと。不明点や疑問点は直ちに明確にすること。

３．改善活動の実行（弱点の克服と生産性の向上）

チームの弱点を明確にし、その改善活動を行う中で、苦手な分野を克服し、新たな知識を継続的に獲得すること。 改善活動は、生産性および品質の向上を同時に図るために、仕様の早期把握および技術力強化に焦点を絞って実行すること。

Check List ＃５７ プロジェクトリーダーの役割

▶Check Timing：開発着手前

- □ プロジェクト全体の「ヒト、モノ、カネ、情報」の統合管理ができているか。
- □ 開発内容の全体像を把握できているか。
- □ 顧客要求の優先順位を把握できているか。
- □ 適正な見積り回答が実行できているか。
- □ 顧客との密接なコミュニケーションを通して早期仕様凍結が実行できているか。
- □ プロジェクト計画書が作成されているか。
- □ プロセス管理ができているか。
- □ 進捗管理ができているか。
- □ リスク管理ができているか。
- □ 課題管理ができているか。
- □ 損益管理ができているか。
- □ プロジェクト完了報告および振り返りを行っているか。
- □ チーム内の日次情報共有会議を実行しているか。
- □ プロジェクト関係他社との情報共有・連携ができているか。

Check List ＃５８ リーダーの過重負荷軽減

▶Check Timing：開発全工程

- □ 一人であらゆる仕事を抱え込んではいないか。
- □ 抱えているルーチンワーク的作業を他のメンバーに分散しているか。
- □ ノウハウの継承等によりサブリーダーを育成しているか。
- □ リスク管理表などで問題の事前掘り起こしおよび解消を行っているか。

Check List ＃５９ プロセス管理

▶Check Timing：開発着手前

- □ プロセス管理表の作成および運用をおこなっているか。
- □ 仕様凍結遅延防止の活動を適時に行っているか。
- □ 妥当な設計・製造・評価工程期間の確保ができているか。
- □ 開発各工程のスケジュールの遵守および主要イベントの進捗管理ができているか。
- □ 各工程におけるレビューが確実に実行されているか。
- □ 各工程における成果物の妥当性が確実に管理されているか。

Check List ＃６０ プロジェクト計画書

▶Check Timing：開発着手前

□ プロジェクトの特徴・難易度に応じた妥当性のある開発／評価体制を構築しているか。
□ 開発環境の事前準備ないしは整備を行っているか。
□ 詳細かつ妥当な開発スケジュール表を策定しているか。
□ メンバーの育成計画を考慮しているか。
□ 採用ルールの規定に基づいた協力会社の適正人材の投入計画を策定しているか。

Check List ＃６１ 組織編制のポイント

▶Check Timing：開発着手前

□ リーダー、サブリーダー、メンバーなどのスキル階層別のピラミッド構成を築くこと。
□ リーダーは複数のサブリーダーを育成し、サブリーダーはメンバーを育成すること。
□ 人員異動の要求に対してはチーム能力の急激な低下を防ぐために二番手のサブリーダーおよび二番手のメンバーを異動させること。
□ 異動させたメンバーに関しても自分の組織に必須な人材は、将来元の所属に戻す約束を取り付けておくこと。
□ 特定顧客に対するノウハウの属人的な保有のリスクを減らすために、必要なノウハウは必ず最新状態として技術ドキュメントに記載し更新を怠らないこと。

Check List ＃６２ 人材管理

▶Check Timing：開発着手前

□ プロジェクトは上級者から初級者に至るまで適切なスキルの人材で構成されているか。
□ 人材管理カードなどで個人別のスキル管理が行われているか。
□ 人材の基本的な条件が満たされているか。
　□ 必要な技術的能力を保有しているか
　□ コミュニケーションに必要な自律的な報告・連絡・相談などの行動ができるか。
　□ チームプレーに必要な協調性・誠実性などがあるか。

Check List ＃６３ メンタルヘルス

▶Check Timing：開発全工程

□ メンバーを孤立無援の状態に置いてはいないか。
□ メンバーの弱点を特徴として生かせる方法を模索しているか。
□ 他の人の一助になる行動をしているか。
□ 心身不調の者が支援を求めやすい雰囲気を醸成しているか。
□ 上位に位置する役職者は、問題が深刻化する前に発見や手当をしているか。
□ 仕事起因で発生した心身の障害は役職者の責任であると言う自覚を持っているか。

＜コラム＃３８＞　メンバーが勝手な行動をすると嘆く前に

メンバーがベンダー側担当者に確認せず、勝手な自己判断で不具合を発生させる、あるいは不明点や疑問点を質問せずにいつまでも考えた挙句に時間を浪費する、といった状況が発生している場合の問題の多くは、メンバー自身の問題と言うよりもリーダー自身の問題であると考えられます。メンバーは何らかの理由でリーダーに相談しにくい状況に置かれているのでしょう。

メンバーが何故リーダーに相談に来ないのか良く考えてみる必要があります。

リーダーは普段から部下とのコミュニケーションを深め、部下が勝手な自己判断で作業をしていないか、何か問題を抱えていないかなどの気配りが必要です。リーダーとメンバーの間の精神的な距離を縮めるためにも日常の直接会話と同時に日次情報共有会議を励行する必要があります。

Check List　＃６４　プロジェクト完了報告・振り返り

▶Check Timing：開発完了時

□　失敗の真因および再発防止策のまとめ等の振り返りが行われているか。

□　ＱＣＤの目標値／実績値の対比、原因分析、対策が行われているか。

□　今後の課題のまとめと振り返り結果の他チームへの横展開が行われているか。

＜コラム＃３９＞　複数プロジェクトの管理方法

複数のプロジェクトを管理するためには管理専門者を置けば良いのですが、小規模のプロジェクトが複数の場合は予算規模の理由によって管理専門者を置く余裕はないのが現実です。そのような状況のもとでは一人の開発リーダーが自分も開発実務を抱えながら複数のプロジェクトを管理しなければならなくなります。このような状況下で複数のプロジェクトを管理するためには下記のことを実行する必要があります。

１．管理方法のパターン化

日々出てくる問題に対処療法的に対応していたのでは、いくら時間があっても足りません。そのためには個々のプロジェクトの管理手法が確立されている必要があります。さらにそのためには、何をいつチェックすべきか、ということが可視化されている必要があり、これらに必要な管理資料としては、プロセス管理表、要件管理表、品質管理表、進捗管理表、リスク管理表、課題管理表などがあり、常にこれらの資料にプロジェクトの現在状況を反映し役に立つように整備しておけば複数プロジェクトを一人で管理することも可能になります。プロジェクトの規模が小さければこれらの管理資料の作成・運用に必要な時間も相対的に少なく済み、さらにこれらの管理表への登録を開発メンバーのルーチンワークに組み込むことでリーダーの負担を減らすことも可能です。このような管理ドキュメントによるプロセス管理に慣れない間は多少の負荷増加になりますが、単なる口頭ベースの対処療法的な管理による混乱や無駄な時間の消費に比べれば、ずっと効率的なやり方と言えます。

２．日次情報共有会議の実行

個別の問題に振り回されることを避けるため、ないしはメンバーの教育や情報共有のために、プロジェクトメンバー全員参加の、短時間の情報共有会議を毎日実行することが効果的です。

Last Point　：　プロジェクト管理

プロジェクトの管理問題は大きく分けて次の二点に集約されます。

□　プロジェクト管理の全体像を理解していないこと。

□　プロジェクト管理の実行の方法を理解していないこと。

要するに何を（What）どのような方法（How）で実行してよいのか理解できていないところに問題の真因があります。この問題は開発実務における，確定された要求仕様（What）に基づいて設計・製造・評価（How）を行う場面においても同様の問題を引き起こしています。

プロジェクト・マネジメントにおいて何を実行すべきかが分かっていなければ，実行段階に進めないことは当たり前で，何を（What）すべきかを最初に理解しておくことが最重要なことだと言えます。

１４．　チームプレー問題のチェックリスト

First Point　：　チームプレーの本質は相互義務の履行と相互扶助の発揮

チームプレーはうっとうしいだけだと思っていませんか。動物的な能力に劣っている人類が他の動物に勝利したのは集団として行動する能力に優れていたからだとも言われています。この能力は人間集団同士の競争においても，その勝敗を決する要因となっています。プロジェクトチームは集団行動の小さな単位そのものでしょう。集団を維持する重要な要素には，相互義務の履行と相互扶助の発揮の２つがあります。相互に果たすべき義務を果たし，相互の弱点を補完し合うということです。個人および組織が生き残り，更なる発展をするための原点がチームプレーにあります。

チームプレーにおける主なリスクは次の通りです。

□　相互義務の不履行

□　相互扶助の不在

＜コラム＃４０＞　自分の知識・時間・労力の譲りは自他をともに進化させる

自分の持てる知識・時間・労力を他の困っているメンバーのために使うことはチームの業務品質を上げるのみならず、団結力を強化し、組織力を増強させます。さらに支援を受けた人においては、いつか将来きっとあなたにその恩を返したいと思う人もいるでしょう。

困っている時の他人からの支援程ありがたいものはありません。自分がして欲しいことを他人にすることが譲りの第一歩となります。支援を受けた人のほとんどはこのことを必ず覚えているものです。「情けは人のためならず」と言うことです。この譲りの行為はまた、ノウハウの継承という形で表れて来ます。譲りの行為が個人的対応から組織的な対応へ拡大されることによって、その個人たちおよび組織は目に見えない強大なパワーを手にすることができます。

Check List　＃６５　チームプレー問題（組織起因）

▶Check Timing：事前準備工程

□　目標が設定されていない仕事が行われていないか。

□　過酷なスケジュール（長時間・長期間残業）が強いられていないか。

□　チーム内で相互義務の履行および相互扶助が行われているか。

□　毎日の声かけや日次情報共有会議を実行しているか。

Check List ＃６６　チームプレー問題（個人起因）

▶Check Timing：事前準備工程

□ やらされ意識で、仕事が行われていないか。
□ 繁忙時の他人からの質問等に邪見な対応をしてはいないか。
□ 他人を意識し過ぎて、過剰防衛的な孤立状態に陥ってはいないか。
□ 自分の利益に関係なく困っているメンバーの支援を行えるか。
□ 依頼や指示をされた仕事に対して、“無理”ですという反応ばかりしてはいないか。
□ 一見、困難な仕事に対して、できるための実行条件を提示しているか。
□ 日々の報告・連絡・相談などを行っているか。
□ 実行済内容、実行予定、抱えている問題等について日報に記録しているか。
□ 自分に可能な範囲から助け合いの行動を始めているか。
□ 行動できないことを自分の性格のせいにしてはいないか。
□ 仲間意識よりも競争相手意識が強すぎないか。
□ 助け合いや連帯は損なことだと思ってはいないか。
□ 他人に無関心ではないか。
□ 支援を受けることで他人に借りを作るのが嫌だと思っていないか。

Check List ＃６７　開発チームと評価チームの連携

▶Check Timing：事前準備工程

□ 開発チームと評価チームは開発初期の工程から連携し必要な情報を共有すること。
□ 開発チームは、妥当な時期に必要な部材や情報を評価チームに提供すること。
　□ 評価テストに耐えうるプログラムの提供
　□ 評価に支障を来す機能等の制限情報と対応時期情報の提供
　□ 単体および結合テスト成績表の提供
　□ 変更仕様の影響度情報の提供
　□ 評価テストに必要な実機環境および関連部材の提供（ベンダー側提供の場合もある）
　＊上記の開発部提供資料はリリース・ノートと呼ばれている。

＜コラム＃４１＞　前後の工程に対して連携をもって影響力を行使すること

自分が担当する工程の前後の工程に対する考慮は、開発・評価業務の全てにおいて非常に重要かつ必須なことです。自工程の前後の工程に対するさまざまな考慮は、ややもすれば分離分断されがちな各工程を有機的に結び付ける強力な手段となります。プロジェクトのメンバーが開発開始時点からそのように協力しあうことの認識を強く共有し、プロジェクトのあらゆる活動において本当の意味での連携を計画的に実行すれば、すべての工程の業務品質は格段に向上し、結果として自工程で精一杯という状況も軽減されます。このような効果が出るだろうということは多少の知恵をもった開発者であれば誰でも気づくことですが、現実的にはほとんどの人が実行できていません。それは何をさておき自分の担当業務だけを

まず最優先にしなければという偏狭的な視野の狭さが原因です。開発活動は一人では絶対に実現できません。何が本当に自他共に得になる方法かを考え一定の結論を得たら、プロジェクトの主要メンバーとそのことについて話し合いベクトルを合わせ、共同歩調を行なう必要があります。自分の担当工程の前後の工程においてどのような不都合なことが発生しているかをよく見て、それに対して何をすべきか、何ができるかを洗い出し、実行につなげる必要があります。この問題は工程間のインターフェースないしは担当者間のインターフェースに関する問題だと言えます。

Last Point ： All for one,One for all

チームプレーを阻害する主な要因を次に示します。

□ 過剰な自己防衛的姿勢

□ 連帯・連携不足

□ 余裕のなさ

チームプレーの精神は「All for one,One for all」（みんなは一人のために，一人はみんなのために）の言葉でよく知られています。チームプレーを阻害する最大の要因は，“過剰”な自己防衛にあります。自己防衛は人間の生存本能の基本ですが，問題なのはそれが“過剰”に発揮される場合です。たった 10 分の時間も，少しのノウハウの伝承も惜しむような心根は“せこい”という言葉で表されます。“せこさ”が自分や組織を衰弱させることをあらためて認識する必要があるでしょう。二宮尊徳が言うように，自分が持っているものはもともと親や誰か他人がくれたものであり，自分の譲りの大きさに比例したものが自分に戻ってくるのだということに早く気づく必要があります。

１５．　手抜き問題のチェックリスト

First Point　：　あせりは手抜きを招き失敗を生む

不条理な要求ないしは見積もりの失敗によって最初から不可能な開発期間しか与えられなかった場合や，いつまでたっても決まらず二転三転する要求仕様に振り回されて多くの時間を失ってしまった場合，開発チームの全員はまちがいなく“あせり”モードになってしまいます。“あせり”とは目標未達の危険性に対する感情的な反応であり，開発に必要な最低限の期間が実際に失われてしまったのならば，このプロジェクトは間違いなく失敗することになります。必要時間の不足は，業務品質の劣化を招き，調査・検討不足に始まり，工程中断，ひいてはある工程自体のスキップという重病を招いています。

手抜き問題の主なリスクは次の通りです。

□　必要工程の中断による製品品質および人間品質の悪化

□　必要工程のスキップによる致命的な製品品質および人間品質の悪化

＜コラム＃４２＞　ＱＣＤに優先順位はない

納期・コスト・品質に優先順位はありません。皆必須条件です。人間において脳を取りますか心臓を取りますか選んで下さいというのと同じです。世の中のプロジェクトの優先順位の実態は本当のところ、コスト＞納期＞品質の順になっています（2009 年日経コンピュータ プロジェクトの成功率 プロジェクト実態調査 800 社）。コストはマネージャ以外には見えにくい指標であるから、一般の開発者に見える部分では納期優先となっています。約束の時期を外すことはできないため、最後の評価・デバッグの時間が不十分だったとしてもタイムアウトということでバグだらけで出荷されるソフトウェアが後を絶ちません。コストを守るために、必要な時間をカットして納期の帳尻を合わせ、品質を犠牲にしているという図式です。悪い品質のものを買わされて被害を受けるのは顧客自身です。みんな何の為に仕事をしているのかということを真剣に考える必要があります。 誤動作するような品質のソフトを納期通りに納めても、それにどのような価値があるのか考えてもみてください。一部の人間が、そもそも自分たちの計画性のなさによって招いた時間不足を棚上げにしておいて、品質を犠牲にしても良いから納期を守れ、などと言うこと自体、責任感・使命感を放棄した行為です。本来やるべきことは、明確な要求仕様の適切な時期における凍結とそれに基づいた妥当な開発期間の確保と効率的な開発努力しかないはずです。

■ 設計・製造における手抜き

Check List ＃６８　主な手抜き項目（設計・製造）

▶Check Timing：設計・製造着手前

- □ 仕様の事前調査・検討をスキップしていないか。
- □ ドキュメントに基づく設計・製造・評価が行われているか。
- □ 基本設計工程をスキップしていないか（基本設計書未作成）。
- □ 詳細設計工程をスキップしていないか（詳細設計書未作成）。
- □ 仕様実現に必要なメモリー容量、ハードウェア諸元を確認したか。
- □ パフォーマンス性能、レスポンス性能の考慮が抜けていないか。
- □ 異常系処理の考慮が抜けていないか。
- □ 各工程における自己チェックをスキップしていないか。
- □ 各工程の内部／外部レビューをスキップしていないか。
- □ ソースコードやドキュメントの無節操なコピー＆ペーストが行われていないか。
- □ プロジェクト完了時の振り返りやラップアップをスキップしていないか。

Check List ＃６９　異常系処理

▶Check Timing：設計着手前

- □ 各種Ｉ／Ｏ系の異常系処理は、ガイドブック等で文書化されているか。
- □ ＭＩＮ／ＭＡＸ制御の閾値はチェックされているか。
  - □ メモリーリソース
  - □ データ長
  - □ 伝文長
  - □ カウンターサイズ
  - □ ポインター値
- □ 異常系項目はチェックされているか。
  - □ 排他制御
  - □ 非同期制御
  - □ 処理タイミング
  - □ リトライ処理
  - □ タイマー値
  - □ レジストリ
  - □ 設定間矛盾

＜コラム＃４３＞　単純"コピペ（copy & paste）の重大な弊害

要求仕様書、設計書、ソースコード、チェックリストなどにおける、誤字、脱字、モレ、コピペミスは慢性的になっています。はっきり言えることはこれらの作成者は、作成した後に一度も見返していないということです。見返してないという行為は"手抜き仕事"をしているということに他なりません。時間や工数が足りないなどという言い訳は全く通りません。時間がなかったから信号を無視して交差点に突入して事故を起こしてしまいましたということと同じです。これらの手抜き行為で最も危険な行為は"コピペ"でしょう。自分の観察や判断を一切行わず、どこかの誰かが書いたものを単純にコピー＆ペーストすることなど仕事をしているどころか、自分で不具合のもとを量産しているようなものです。もし流用したいソースコードやチェックリストがあったなら、その全文を自分で検証し添削を入れた後に流用すべきです。流用元における入出力やその他の条件が異なっている場合が多いのにもかかわらず、何らの検証もなく"コピペ"する行為は開発・評価業務中における最悪の行為の一つであるという認識を開発者全員が持つ必要があります。

またいいかげんな評価工数の見積りの原因は、いいかげんな設計見積りにあり、いいかげんな設計見積りの原因は、いいかげんな要求仕様の把握にまで遡れます。設計チームは評価チームとも連携し、最上流の要求仕様の明確化および早期凍結に最大の力を発揮することで、妥当な見積り、妥当な工数の獲得を実現する必要があります。

Check List　＃７０　プログラマーがやってはいけない手抜きの１２ヶ条

▶Check Timing：プログラミング着手前

□　ソースコードの"コピペ"を無節操に行なっていないか。
□　ベースのプログラム構造を理解しないままソースコード変更に着手していないか。
□　データ処理の仕掛けを理解しないままソースコード変更に着手していないか。
□　変更対象の機能の呼び出され方、関連機能との連携の仕方を理解せずソースコード変更に着手していないか。
□作られたデータはどの機能がどのタイミングでアクセスしているのか理解しないままソースコード変更に着手していないか。
□　他人の書いたソースコード理解するためにそれを読むことだけしかしていないのでは。
□　一つの変更要件に対して複数の担当者が関係している場合、早い段階でお互いに検討内容等の相互確認・レビューをしているか。また、みなバラバラにソース修正にとりかかり、寄せ集めたソースで一気にテストをしているのではないか。
□　お互いにどこを変更したのか誰も知らないのではないか。
□　一つの機能に関する多数の関数において同一巨大パラメータ構造体（２５６バイト等）を共通に使用してはいないか。
□　開発中に発見された、以前から含まれていた潜在バグについていきなりソースコードの修正を行なっているのではないか。
□「プロジェクト計画書」を書いていないのではないか。
□「データ」をとっていないのではないか。

（出典；清水吉男著「派生開発を成功させるプロセス改善の技術と極意」および「要求を仕様化する技術　表現する技術」）

■　評価テストにおける手抜き

Check List　＃７１　主な手抜き項目（評価テスト）

▶Check Timing：テスト着手前

□　インターフェースにおける最終的な出力確認をスキップしていないか。
□　テスト項目未消化のまま次工程に進んでいないか。
□　単体テストをスキップしていないか。
□　結合テストをスキップしていないか。
□　総合テスト項目未消化のまま客先にリリースしていないか。
□　発見済み未修正バグを含んだままリリースしていないか。

＜コラム＃４４＞　最終成果物は必ず現物確認を

リリースしたソフトが動作しなかったという事例は構成管理失敗の典型的な例の一つです。最終の出荷確認は正式にビルドが完了したパッケージの現物を使って基本動作を確認しなければいけません。開発環境上にあるプログラムを使って最終確認をすると、このような問題を引き起こします。この最終現物動作確認は構成管理用のチェックリストだけではなく、本来はプロジェクトのプロセス管理表に最後の重要プロセスとして記載されるべきものです。開発業務において、”多分大丈夫”という考え方は、必ず失敗の原因となります。この問題は時間を経てまた再発する危険性が高いため、リリースの最後の現物確認については、都度チーム全員に注意を喚起し、チェックリストなどのドキュメントに従った開発を励行する必要があります。これらのことを無視した最もひどい例としては、ソフトが書き込まれていない空のメディアがリリースされた例もあります。

■ 手抜き防止

Check List ＃７２　手抜き防止

▶Check Timing：開発全工程

□ 仕事の意味を考えて作業を行っているか。
□ 時間がないことを手抜きの言い訳にしていないか。
□ 予算不足を理由に手抜き仕事が行われてはいないか。
□ 合理的なプロセス管理表に基づいて開発を実行しているか。
□ 重要な手順抜けを防ぐためのチェックリストを運用しているか。
□ 手順慣れで必要なチェックを飛ばしてはいないか。
□ コーディングやドキュメント作成において安易なコピー＆ペーストを行ってはいないか。
□ ソフトウェアやドキュメント等の成果物の出荷前の現物確認や動作確認を行っているか。
□ 深刻な疲労に陥ってはいないか。
□ 手抜きで発生した実例を本人に示しているか。
□ 日次情報共有会議等で問題の共有を毎日実行しているか。
□ 前記問題点の改善活動をリーダー主導で行っているか。

Last Point　：　時間制御の失敗

設計・製造に共通する基本的な問題は“時間制御の失敗”にあると言えます。この問題の解消方法は次の通りです。

□ 説得力のある見積もりに基づき妥当な開発期間・費用の獲得を図ること。
□ 早期の仕様凍結を行うこと。
□ 必要時間の削減のために，業務の効率化および失敗の削減を常時実行すること。

あせりによる手抜き行為を防ぎ正常な開発活動を実現するために、上記の三点の実行に個人はもとより組織を挙げて、その全勢力を注ぐ必要があります。ポイントは必要時間の確保および無駄な時間の削減にあります。

１６．　時間認識問題のチェックリスト

First Point　：　目には見えない時間

人間が最も苦手とする認識が時間に関する認識です。人間は，「自分の見たものがすべて」という強い心理的傾向をもっているために，目には見えない時間というものを上手にコントロールできません。いつも時間に追われている人々が多いのもそのためでしょう。いくらあせったとしても時間が延びるわけでもありません。あせりから脱却するには用意周到な準備を行う以外には方法はありません。

時間認識に関する主な問題には次のものがあります。

□　仕様凍結期限意識の欠如

□　期限・タイミング意識の欠如

□　許容時間認識の欠如

Check List　＃７３　自分の時間を確保するためのポイント

▶Check Timing：開発全工程

やる必要のないことをやらないで済むようにしているか。

□　リスクの排除

見積り精度の向上、要求仕様の早期凍結などのリスクを事前に排除すること。

□　無理・無駄の排除

無理・無駄を徹底的に発見し排除すること。

やるべきでない事をやらないようにしているか

□　責任範囲外の活動は基本的に行わないこと。

□　メンバーの育成と同時に各自のレベルにふさわしい仕事を行うこと。

□　部下を信頼せず、部下がやるべき仕事をやってはいないか。

□　**やるべき事は完全に実行すること。**

＜コラム＃４５＞　必要時間と許容時間

個人であれ組織であれ、それぞれが持っている時間は、言うまでもなく有限です。我々が仕事をするにあたって、最初に意識しなければならないことは、有限な時間においても、必要時間と許容時間（手持ち時間）の２種類があるという当たり前のことです。

必要時間とは、あるものごとを遂行するために必要な時間のことで、その長さは個人ないしは組織の業務遂行能力（パフォーマンス、生産性）に依存しています。 一方、許容時間（手持ち時間）とは、業務遂行にあたって自分たちに許されている時間のことです。必要時間と許容時間が一致していれば時間不足という問題は発生しませんが、競争社会の中にあっては常に他より早くという圧力を受け続けるために、許容時間が必要時間よりも短くされることが多くなります。

時間不足を無くすためには、許容時間を増やすか必要時間を減らすか、両方を同時に実現することしかありません。 許容時間とは顧客と約束した時間そのものです。許容時間は見積り交渉によって決定されるため、妥当な許容時間を確保するためには、しっかりとした裏づけに基づいた説得力のある見積り回答が必要になってきます。

一方、必要時間を減らすためには、開発組織の業務遂行能力を上げること、すなわちその生産性を上げることが必要になってきます。生産性の向上は、業務の効率化および失敗の削減の二面によって決定されます。

Check List ＃７４　自分の時間を生み出すためのポイント

▶Check Timing：開発全工程

□ 要求仕様の早期凍結のための対策を実行しているか。
□ 精度の高い見積りおよびタフな交渉で妥当な納期・開発費を獲得しているか。
□ 顧客価値の重要度順を把握し、その順に開発・評価を実行しているか。
□ 失敗の原因になりそうなリスクの回避・排除を事前に実行しているか。
□ 問題のパターン化による対策のモレや無駄を防止しているか。
□ 業務効率化および無駄の排除等の改善活動を実行しているか。
□ データやドキュメントに基づいた合理的な仕事の進め方を行っているか。
□ 柔軟かつシンプルな設計を行っているか。
□ 中間レビューを実行しているか。
□ 進捗を正確に把握するために、量だけではなく質の管理も行っているか。
□ 顧客間や社内における良好なコミュニケーションを実行しているか。
　□ 求められている成果を正確に確認しておくこと。
　□ 期限を確認しておくこと
　□ 作業指示において、目的や意味を正しく伝えること。
　□ 顧客との密接かつ直接的なコミュニケーションを維持すること。
　□ チーム内の短時間日次情報共有会議を毎日実行すること。

＜コラム＃４６＞　検討・調査時間の使い方の３分割法

ある検討・調査をするにあたって最初にすべきことはこの検討・調査で許される時間や日数がどれ位なのかを知っておくことです。次に許された時間を３等分し、最初の１／３の時間や期間でまず仕事の全貌を把握します。把握の途中で不明点は一旦そのままにしておき、赤印などで印をつけておき、全体を見終わったところで、印をつけた箇所について、①全く分からないもの、②短時間の検討・調査で分かる項目、③検討・調査に長時間かかりそうなもの、に３分類し、①についてはすぐに依頼者に質問を出したり、有識者に聞いたりします。

次の１／３の時間で②と③の検討・調査を行いますが、ある程度の時間をかけても分からなければ、その時点で②③について依頼者に質問を出したり有識者に聞いたりします。不明点の解消はこのサイクル

内で完了するように心掛けることが必要です。

最後の１／３の時間で依頼者からの要求と自分が検討・調査した結果が整合性のあるものになったかどうかを確認します。

上記はあくまでも検討・調査時間（期間）だけについての話であり、仕事全体の時間（期間）配分ではありません。検討・調査時間は仕事の難易度や量によって異なるでしょう。

Check List ＃７５　時間の使い方

▶Check Timing：作業着手前

□ 事前に必要時間が許容時間内に収まる見込みをつけて仕事をしているか。

□ 問題の全体像の把握を先に行い、詳細な検討はその後にしているか。

□ 解けそうにもない問題にタイムリミットを設けて取り組んでいるか。

□ 解けない問題について、適時に有識者の支援を仰いでいるか。

Check List ＃７６　製品におけるパフォーマンス／レスポンス時間

▶Check Timing：設計／テスト着手前

□ ソフトウェアのパフォーマンス時間はシステム要件を満たしているか。

□ ソフトウェアのレスポンス時間はシステム要件を満たしているか。

＜コラム＃４７＞　パフォーマンス性能・レスポンス性能は重要なシステム機能要件

パフォーマンス・レスポンス性能未達は良くある重大なミスの一つです。「ロジックは正しく動きました、ただし処理時間内に終了できませんでした」という障害です。このような失敗を繰り返す開発者の頭の中には”時間”という感覚がないのかも知れません。この問題はリスクの問題ではありません。何が問題の本質であるのか、よく考えて見ればわかることです。ソフトウェアの基本はロジックが正しいのは当然として、適正時間内に応答が返ってくるという「レスポンス」、適正時間内に必要な処理が終了するという「パフォーマンス」の両方を達成する必要があります。ボタンを押して帳票が印刷されるまでに何分もかかるような端末は実用に耐えません。セットした時刻に鳴らない目覚まし時計など無用のガラクタです。 レスポンスとパフォーマンスは重要なシステム機能要件の一つであり、全ての処理に対してレスポンス・パフォーマンスを考慮した設計や製造を行うことは常識中の常識です。

＜コラム＃４８＞　時間不足の原因と必要な時間を確保する方法

時間不足の主な原因としては次の３点があります。

①そもそも実行に必要な時間が確保されていない。 ②失敗および不要な作業による無駄な時間の浪費。 ③割り込み作業による時間の消費。

多くの時間不足の原因は、上記原因の組み合わせによって発生しており、それぞれの原因が引き起こしている問題を解決しなければ時間不足は解決しません。それぞれに対する対策を以下に示します。

【必要な時間を確保する】

最も影響しているものは開発工程の始めに行われる見積りおよびスケジューリングです。見積りをできるだけ安くするために、多くの見積りにおいては失敗やリスクの考慮を省いた見積りが行われやすく、また発注者側からの強制的な金額・期間のカット要求がある場合もあります。この様な状況に対処し必要な工数・期間を確保するためには、妥当性のある見積りを提示し、それを相手に納得していただく技量がプロジェクトマネージャやリーダーに必要となります。その技量の基本となるものは、開発における生産性のデータです。日ごろから開発におけるＱＣＤのデータを収集していなければ、とうてい正確な妥当性のある見積りを提示することはできません。

【失敗および不要な作業による無駄な時間を排除する】

開発の各工程における失敗の原因を常に明らかにし、その改善対策を継続的に実施しなければ何度も同じ失敗を繰り返します。この問題に対処するためには失敗に学び、その対策をチーム内外に広げていくような活動、すなわち改善活動を続ける必要があります。また失敗に至らない場合でも、より効率的な業務の実施方法を考え、それを実行することも必要となります。

【割り込み作業による時間の消費を削減する】

割り込み作業が常に優先的な仕事とは限りません。複数の仕事が重なった場合は、必ず依頼者や上司とも相談の上、優先順位を決めて取り組む必要があります。また避けられない緊急の割り込み作業は必ずあるという前提で、各仕事は可能な限り前倒しを行うようにしておく必要があります。前倒しするためには当然のことに日常の改善活動が必須となります。

Last Point　：時間制御のポイント

個人およびチームにおける時間制御のポイントは次の通りです。

□　必要時間が許容時間内に収まることを確認してから行動を開始すること。
□　見積り回答における妥当な開発期間の獲得に全力を尽くすこと。
□　早期の仕様凍結に全力を尽くし、大規模な時間ロスを無くすこと。
□　事前着手などの想定仕様開発をなくし、大きな手戻りをなくすこと。
□　改善活動などを通じて、技術力・思考力を鍛え失敗を減らすこと。

１７．　ノウハウ継承・人材育成問題のチェックリスト

First Point　：　ノウハウの譲りが人と組織の成長を生む

自分だけの成長を願っても、それでは余り効果がでないことでしょう。自分は、その仲間と共に成長したいという思いや行動が人の間に相互作用を起し、皆の成長をもたらすことになります。部下が成長できない問題の本当の原因はリーダー自身にあるのかも知れません。

一連の組織活動の最後の仕事は、その活動において獲得されたすべての有用なものを次に続く者たちに継承していくことです。このノウハウの継承行為は、良いものの好循環を生み出し、個人および組織の永続的な成長・発展には不可欠のものとなります。

ノウハウ継承・人材育成問題に関するリスクは次の通りです。

- □ 個人のスキル向上の疎外
- □ 組織能力向上の疎外
- □ プロジェクトＱＣＤの低下

■ ノウハウの継承

Check List ＃７７　ノウハウの継承はドキュメントによって

▶Check Timing：ノウハウ継承前

【見積り、要求仕様】

- □ 要求仕様書品質の評価ガイドラインはあるか。
- □ 要求仕様変更管理マニュアルはあるか。
- □ 見積回答書作成マニュアルはあるか。

【開発管理】

- □ プロジェクト管理マニュアルはあるか。
- □ 品質管理ガイドラインはあるか。
- □ コスト／プロフィット管理ガイドラインはあるか。
- □ 進捗管理ガイドラインはあるか。
- □ 生産性管理ガイドラインはあるか。
- □ 開発／評価プロセス手順書はあるか。

【技術情報】

- □ 統一専門用語集はあるか。
- □ 顧客システム情報集はあるか。
- □ 顧客システム運用情報集はあるか。
- □ 技術情報／ノウハウ集はあるか。

□ アプリケーション仕様集はあるか。
□ 仕様変更影響度表作成マニュアルはあるか。
□ 基本設計ガイドラインはあるか。
□ 詳細設計ガイドラインはあるか。
□ プログラミングガイドラインはあるか。
□ 評価設計ガイドラインはあるか。
□ テスト仕様書作成マニュアルはあるか。
□ 開発／評価ツール操作マニュアルはあるか。
□ 機器操作手順書はあるか。
□ 構成管理マニュアルはあるか。
□ 障害対応マニュアルはあるか。
□ 重要障害対応記録集はあるか。

■ 人材育成

Check List ＃７８ 仕事を任せる場合のポイント

▶Check Timing：作業委任前

□ 上司は、任せる仕事の全体像を完全に把握しておくこと（丸投げしてはいけない）。
□ 仕事の難易度とメンバーの能力レベルを見極めておくこと。
□ 作業の目的・意味・背景を説明すること。
□ 仕事の指示内容は口頭のみならずドキュメント資料により、漏れや誤解を防止すること。
□ 説明資料は、細かさも大切だが、分かりやすさに重点をおくこと。
□ 作業指示時に、相手の反応を確かめ、理解度をチェックすること。
□ 日次情報共有会議等で問題・疑問点のヒアリングや助言等を行うこと。
□ 部下の能力や気質を常に把握し、徐々に負荷を増していくことでその成長を図ること。
□ 本当の失敗を防ぐために、事前に適時のレスキューを計画しておくこと。

＜コラム＃４９＞ 仕事の任せ方について

自分でやった方が早いと思い人に仕事を任せられない。その結果多くの仕事を一人で抱え込んでいるリーダーが多く、自分の仕事も回らなくなり、さらに部下の成長もうまくいかないという悪循環に陥いることになります。どの仕事を誰に任せるかという問題は、難易度や量に対応して初級・中級・上級者のどのレベルの人が適任かという判断に従えば答えは明白でしょう。上級者が、仕事が速いのは当たり前のことで、いつも上級者であるあなたばかりが仕事を占有していては部下の成長が遅れるだけではなく、あなた自身も過負荷になり、あなたが更に上級のスキルを身につける機会をも失ってしまうことになります。自分がやった方が早いからという理由は適切ではありません。経済効率上で言えば、あらゆる仕事において上級者が下級者の仕事を行うことは全く不適切で合理性が認められません。高い賃金の人が安い賃金の人の仕事をやるのは経済的に非合理的なだけではなく、下級者の育成の機会を奪い、自分がなすべきもっと

上級レベルの仕事を放棄することにつながります。

後輩の技量は信頼できないので、とりあえず自分で、何でもやってしまうというのでは後輩はいつまでたっても育ちません。何とかその後輩が頑張ればできそうなら、その人を指導し、フォローしつつ任せるべきでしょう。

仕事における信用・信頼の基準として”仕事を任せられること”だと皆さん異口同音に言っていますが、無条件に仕事をまかせられる人などそうそう居るものではありません。一言いえば何でもやってくれる人などどこにも居ません。リーダーにおいては”仕事の任せ方”が重要なポイントとなります。相手によっては細かい指示が必要になるでしょう。仕事において他者との信用・信頼関係を作りあげるために必要なことをもっと細かく具体的に考えてみる必要があります。対客先、対上司、対同僚、対部下、部下もそれぞれのレベルがあり、これらの対人関係において実行すべきことは色々と異なってくます。共通のキーワードは”良い仕事をするためには何をすべきか“ということに尽きます。

＜コラム＃５０＞　人のレベルに合わせた指導を行うこと

教育の基本は、それぞれの人のレベルに合わせて指導を行うことにあります。理解が足りない人には細かく、理解が進んでいる人には高度な指示が必要となります。一律に、作業の前に細かな指示を出さないことが、全ての部下にものごとを考えさせる機会になるとは限りません。分かっていない部下への指示はつい細かくなり勝ちですが、いきなり自分で考えろと言ってもなかなか難しいでしょう。このような人に対しては少し面倒ですが、指示した内容のポイントを復唱させ、逆にその指示ポイントの意味を答えさせるようにして見てはどうでしょうか。相手の持っている疑問点・不明点をどれだけ聞き出せるかが重要なポイントになります。

また、指示を与える場合においては最低限、その仕事の目的・意味・背景について明確に説明を行う義務があります。また部下に仕事を指示するにあたっては必ず資料に基づいて説明する必要があります。ドキュメント・ベースの仕事は、説明忘れを防止し、お互いの理解の一致を確認するのに有効な手段となります。

＜コラム＃５１＞　大いに失敗を恐れること

”失敗することを恐れる”感情は道理に適った当り前のことです。この恐怖感や不安感は否定すべき感情ではありません。困難な仕事に対しては大いに恐れる必要があります。人が恐れや不安に対してとる行動には二通りがあります。怖いから何もしないで逃げるか、怖いから万全の準備をしてから問題に取り組むかのどちらかです。失敗することを恐れるために部下に仕事を任せられないということではなく、失敗を恐れるために事前の準備や細心のフォローを行いながら部下に任せてみるということが必要でしょう。

部下に大きな失敗をさせてはいけません。プロジェクトは絶対に成功させなければいけません。難しい仕事を任せた人に対しては日次会議などを通して必ず抱えている問題点をはっきりさせ適切な支援を行い、その人の力で成功させるようにする必要があります。またメンバー全員の保有技術能力・経験システム等についてのキャリアシートを作成し、適材適所および各メンバーの育成を常に目に見える形にしておき、リーダー層で共有するような活動も必要です。

Check List ＃７９　社員教育の場

▶Check Timing：教育実施前

社員教育の場として以下を意識的に利用すること。

□ チーム内における日次情報共有会議
□ 開発着手前における仕様の説明時
□ 仕様調査時における不明点および疑問点に関するＱ＆Ａのやりとり時
□ レビュー時（設計、コードレビュー、単体テスト、結合テスト、総合テスト）
□ 不具合修正時
□ プロジェクト終了時の振り返り会議

＜コラム＃５２＞　開発プロセスの確実な実行が全て教育の場となる

メンバー教育の一番の障害になっていることは、人の性格ではなく、おそらく教育に必要な適切な教材がないこと及び時間不足にあると考えられます。別途教材を作ろうとしても現実的には時間がなくて作成できないでしょう。教育の基礎資料として一番の教材は仕事そのものにおいて作成された優れた要求仕様書、設計書、評価資料などです。そのためにはメンテナンスもされていないような資料はほとんど役に立ちません。また応用資料としては改善活動の中から生み出された種々のノウハウ集がそのまま利用できます。もしこれらのちゃんとした資料がないのであれば、当面は自分が保有する有効な資料および頭の中にあるノウハウの伝授を日常業務の中で地道に実行することから始めるのがよいでしょう。

Check List ＃８０　スキルアップ

▶Check Timing：教育実施前

□ チーム内の日次情報共有会議等を通して問題点などの指導を行っているか。
□ ドキュメント・ベースの仕事を通してノウハウの蓄積ができているか。
□ ドキュメントは常に最新状態に更新されているか。
□ 失敗事例集の蓄積が行われているか。
□ 蓄積されたノウハウは、組織内でいつでも誰でも検索・参照可能になっているか。
□ 改善活動を通してスキルアップを図っているか。
□ 学習の目標および達成時期の計画を持っているか。
□ 学習計画を実行しているか。
□ 専門分野に限らず広い領域の書籍を読んで仕事に役立てているか。

＜コラム＃５３＞　自助努力による人材育成を

教育に当たってきちんとした時間を確保し、しっかりしたカリキュラムに従って、内外の講師による教育をイメージするのなら、現在は一部の大手企業を除いてほとんどの企業においてはそのような教育に割く時間も予算もないというのが現状でしょう。せいぜい実行されていることは外部の教育機関にて行われている有料講座の受講程度だけでしょう。

ただでさえ利益率の低い短納期リスク物件の開発に追われる状況の中では、教育の時間や予算を生み出すことは困難です。自分たちの努力で人材教育を行うためには下記のことを実行する必要があります。

１．まずリーダー自身のスキルアップを図ること。メンバーの育成には、優れたリーダーが必要となります。優れた人材は下記のことを率先して実行する中で養成されて行くでしょう。
２．開発行為の全てが教育の場であると考えること。日次情報共有会議、仕様検討時、仕様説明時、レビュー時、不具合修正時、振り返り会議時の優れた行為は全ての人材の最も良い教育の場となります。
３．開発工程すべてにわたる改善活動の実施を行うこと。現状の各工程における問題の改善活動は開発の効率化、利益の向上、時間の創出の効果を生み出すと同時に、全メンバーの最も効果的な実地教育の場となります。

＜コラム＃５４＞　アプリケーション仕様知識習得のポイント

短期間で体感的に学習するには実際の機器を操作して学習する方法が最も効果的でしょう。新人の開発者においては入社後の一定期間を評価チームにて機器に直接触れて学習する方法が良いでしょう。また新人でなくとも異なった業態・業界の仕様知識を習得するためには、ドキュメントを見ながら機器を操作して見るだけでも多くの知識を吸収することができます。

機器操作に関してすでに慣れている人で、ある特定の仕様について知りたい人は、ドキュメント・ベースの学習だけでも良いでしょう。この方式の必須要件は、学習に値するドキュメントが存在するということであり、正確なドキュメントが存在しない場合は正確な知識が得られない危険性があります。ドキュメント・ベースの開発を実現しているチームにおいては強力な学習方法となります。

ＯＪＴによる学習の長所は、一つのシステムについてじっくりと体系的に学習できることですが、短所は習得に時間がかかるところにあります。 この短所を補うためには、その他の学習方法も合わせて実行する必要があります。

自主学習会による方法は、基本的にドキュメントによる座学に拠りますが、各自の得意とする仕様分野について持ち回りで講義を担当することで多くの仕様について集団的教育としての効果が期待できます。この方式の長所は、各講義担当者において用意された仕様説明等の資料を集積すれば、りっぱな仕様マニュアルとなり得ることです。この方式の欠点は継続して実行することが難しいところにあります。

各チームや組織の実力状況に合わせて、上記の四つの方式を組み合わせた学習を行うことが望まれますが、いずれにしてもその継続は、各自の自学自習の情熱と行動にかかっています。

Last Point　：　自律性のある部下の育成とノウハウの継承

部下育成の目的は、最終的には自分で考え自分で行動できる自律性のある部下を育てることにあると言えます。すなわちものごとを良く見聞きし、良く判断し、適時有効な手段で効果的な行動がとれる人材を育てるということです。部下の育成は、部下の問題であると言うよりも、その部下を育成する上司の問題であると言えます。自律性のない上司には自律性のある部下の育成は難しいでしょう。人材育成の役割を担うリーダーや上司においては、自分自身における自律性の獲得を行うことが、直ちにメンバーの育成につながるという認識をもつ必要があります。

またノウハウ継承の最大の阻害要因はドキュメントによる継承が行われていないということにあります。このことはコミュニケーションにおける正確な情報の伝達における阻害要因でもあり、組織内で重要な情報を伝達するためには文書などのドキュメントが絶対的に必要であるという認識が欠如していることをも意味しています。さらにこの認識欠如の原因は、ドキュメントを作成する時間がないことおよび、他者との相互理解や相互関係をめんどうなことだと感じてしまう束縛感を嫌う姿勢がもたらしているのかも知れません。

１８．　学習能力欠如問題のチェックリスト

First Point　：　失敗に学ぶ気がなければ失敗を繰り返す

要求仕様が確定していなければ基本設計はできません。基本設計が確定していなければ詳細設計はできません。詳細設計が確定していなければコーディングはできません。コーディングが確定していなければ単体テストはできません。単体テストが確定していなければ結合テストはできません。全ての物事に関する簡単な道理についての話です。

例外的に何とかなるだろうと思うことは、窮地に陥った者たちにおける正常化バイアスがもたらす妄想と言ってよいでしょう。

学習能力の欠如に関する主なリスクには次の二つがあります。

- □　失敗に学ばないこと。
- □　振り返り・ラップアップを実施しないこと。
- □　改善活動を実行しないこと。

＜コラム＃５５＞　振り返り（ラップアップ）の意味

ラップアップはプロジェクトを振り返る重要な最後のけじめであると同時に、次なる仕事へのノウハウの継承という重要な役割を担う行為です。この仕事はリーダーの重要な仕事の一つであり、メンバーの自主性に任せておくような性質のものではありません。リーダー主導で必ず実行する必要があります。またラップアップは業務遂行中に採取し分析を済ませておいたＱＣＤの指標数値に基づいた振り返りの実行および次なるプロジェクトへの申し渡し事項をまとめるようにする必要があります。単に、○○の失敗をしないように“がんばろー”と言うような情緒的な会議にすることだけは避けたいものです。

■ 失敗に学ぶ

Check List ＃８１　失敗に学ぶポイント

▶Check Timing：振り返り（ラップアップ）前

□ 振り返り（ラップアップ）を行っているか。
- □ 不具合の発生率や重要度のＡＢＣ分析の実行
- □ 業務モジュール別の不具合発生率の把握
- □ 工程別の不具合発生率・重要度の把握
- □ 個人別、組織別弱点の把握
- □ 再発防止対策の整理・分類および蓄積
- □ 振り返り内容の他チームへの継承

□ 振り返りは、日次・週次・月次・工程毎に行われているか。

□ 失敗の真因および対策についてドキュメント化しているか。
後戻りや追加対応において新たに獲得した、仕様知識や実装理解について、他人でも理解できる形として全員でドキュメントに残しておくこと。

□ 仕様理解に全力を集中しているか。
スケジューリングミスの大元は開発仕様の理解不足による工数見積りミスであり、次回開発においては開発初期の短期間において主要メンバーにての仕様理解と妥当な見積りに全力を挙げること。

□ 顧客やベンダーとの密接なコミュニケーションを行っているか。
顧客やベンダーとの密接なコミュニケーションを通して、仕様ノウハウ、実装ノウハウを継続的に入手するようにすること。これらのノウハウに関しては個人レベルで保有せずに、必ずドキュメントに残してチームの財産として蓄積・利用すること。

□ メンバーの育成を図っているか。
今回の開発で成長が見られたメンバーに関しては継続的に育成を図ること。

□ 損失の回復計画を持っているか。
次回以降の開発にて今回の超過分を取り戻すように改善活動を実行すること。

＜コラム＃５６＞　喉元過ぎれば熱さを忘れる

客先にて大障害が発生した。開発チームは総がかりで原因の究明を徹夜で行い，幸いにも問題は終息した。翌日から何事もなかったように淡々と元の仕事に戻る。原因分析を記録したホワイトボードはすでに消されてしまい，皆でやりとりした情報もすでに散逸してしまっている。障害報告書には，××機能のプログラムミスについての簡単な説明および修正・評価済みの記述があるだけ。どのようにして問題を特定したのか，何が真因だったのか，再発防止策はどうするのかの記述もない。「喉元過ぎれば暑さを忘れる」とはこのようなことを言うのでしょう。

Check List ＃８２　類似不具合の発生原因と防止策

▶Check Timing：振り返り（ラップアップ）前

□　不具合修正時に不具合発生の表面的な原因の修正のみしか実行しておらず、発生の真因の特定およびその恒久的な歯止めや解消を実行していないこと。

□　過去の不具合を検索する仕組みや手段を持っていないこと。

□　不具合情報をレビューに生かせないこと。

□　過去の不具合に対する組織的な取り組みを行っていないこと。

■　改善活動

Check List ＃８３　改善活動への取り組み

▶Check Timing：改善活動着手前

□　ＱＣＤ（品質、コスト、生産性）における問題点を把握しているか。

□　自分における製品品質および業務品質を数値で把握しているか。

□　組織における製品品質および業務品質を数値で把握しているか。

□　自分の問題点について改善活動を行っているか。

□　組織の問題点について改善活動を行っているか。

□　改善活動は、当然の中核的な業務だと言う認識を持っているか。

□　改善活動は、同じ失敗を繰り返さないために有効だと言う認識を持っているか。

□　改善活動は、品質向上、コスト削減、工期短縮に有効だという認識を持っているか。

□　それでも、「時間がないから」改善活動はできないと思ってはいないか。

＜コラム＃５７＞　現状維持の法則

何かに行き詰ったとしても人は今までやってきたことをなかなか変えることができません。例え，その行動が非効率だったとしても，それでよしとされる環境にいればいつまでもやり方を変える気にはなれません。しかし，その問題が限界点を越えたときにそれが新たな問題として立ち現れますが，もう後の祭りです。回復不可能な状態に陥らないようにするためには，地道な改善活動を継続する以外に方法はないでしょう。

Last Point　：　レビュー、振り返り会議の絶対励行を

レビューも振り返り会議も実行されなければ、失敗の経験に学ぶ機会が全くないということになります。そのようなプロジェクトは規模の大小を問わず、手抜きプロジェクトと呼んでもよいでしょう。日常的にこのようなやり方を続けていれば品質も利益も悪化し、人材も全く成長しないでしょう。自分が一担当者の立場であったとしても、プロジェクトリーダーおよびマネージャに問題提起をする必要があります。すべてのプロジェクトにおいて、必ず自己レビューや社内レビューおよび振り返り会議等を開発の規模にかかわらず必ず実行する必要があります。

「振り返り」を行っていない組織は「改善活動」も実行していないと思われ、それでは技術者個人のスキル向上も、組織力の向上も図れず、毎年バグに追い回されるだけの利益も上がらない集団のままとなるでしょう。それで良いわけもない。

１９．　最後のチェックリスト　～誇り高きプロフェッショナルのために

□　**自分を信じて開発を行っているか。**

私たちは、目の前で起きているさまざまな事柄について、他人の思惑を過度に忖度することなく、自分の目でしっかりと見、自分の頭で冷静に判断し、自分の意志により決定し、自分に自然に湧き起こる力で立ち向かっていきたい。

□　**道理に沿った開発を行っているか。**

私たちは、一時の情緒や感情に流されることなく、また私利私怨に惑わされることなく、昔から連綿と続く信頼に値する道理に従い、人々の役に立つ考え方や行動をしたい。

□　**合理性に学んでいるか**

私たちは、天然自然の運行の真理をよく表す数値・数理に基づく科学的合理性にかなった考え方や行動をしたい。

□　**柔軟かつ自由な発想で新たな視点を見つけているか。**

私たちは、今までの自分たちの考え方に固執することなく、自分たちをとりまく変化に対し自由な心を保ち、新たな視点を見つけ、新たな取組みをしたい。

□　**ともに義務を果たし、共に助け合っているか。**

私たちは、自分たちを取り巻く人々と共に相互の義務を果たし、相互の助け合いを行いたい。

□　**価値あるものを譲り合い、分ち合っているか。**

私たちは、自分たちの保有する価値あるものを次の世代のものたちや劣位にある者たちに譲り合い、分ち合いたい。

□　**なすべきことをなし、後は運を天にまかすことができるか。**

それでも物事が良くならない場合でも、あせらず時が熟すのを待ちたい。

私たちは、自分を信じ、世の道理に従い、合理性に学ぶことによって、柔軟な発想で新たな視点を見つけ、仲間とともに相互の義務を果たし、ともに助け合い、持てるものを譲り分ち合うことによって成長発展していくことを本分としたいと思う。それとも、自分を信じず他人を憎み、道理に外れ合理を学ばず、自分のみに固執し社会的な義務を果たさず、他人との競争に明け暮れ、譲り合うことも分ち合うこともしないような人間になるのか。その選択肢は常に自分の手の中にあります。

最後に

本書で提供されたチェックリストは、リスクの見落としを防ぐ役割は果たすもののリスクそのものを解消する直接的な機能は持っていません。チェックリストは、どのような問題を解決すべきかを明らかにし、解決のヒントを提供するだけです。本当の解決策はあなた自身の頭の中にあり、解決行動を行うのはあなた自身なのです。

本書のチェックリストによって明らかになった問題を解決するためには、日々の継続した組織な改善活動の実行が必要となります。改善活動とは、個人の思考や行動の改善を起点とした組織的な思考および行動の改善に他ならず、単にトラブルや障害が減少し利益が出れば良いというような利益追求オンリー的な話ではありません。

本書において明らかになった個人的ないしは組織的な弱点の克服を目指し、喜びと希望に満ちた行動が開始されることを期待します。

問題のパターンを成功の物語として読み変えれば次のようになります。

■ 成功の物語

１．成功の序曲

問題の発端は、第１群の【不条理な顧客要求】および、それに対する【見積り問題】の二つにあり、それに対して開発の準備工程において第２群の適切な【相互義務の履行】および【リスク回避】を実行することで致命的な失敗の原因は排除されます。

２．第２次防衛線での対応

開発組織における最大の弱点である第３群の【コミュニケーション不良】を改善することで、第４群の【情緒的開発姿勢】【モチベーション問題】【本質の把握ミス】【優先順位問題】は確実に改善されていきます。

３．第３次防衛線での対応

第２次防衛線までの適切な行動は、開発現場における第５群の【ドキュメント・ベース開発】の実行を促し、【実務能力】を向上させ、さらに第６群の【リーダーシップの発揮】【チームプレー活動】【手抜き問題の解消】【時間認識問題の改善】を確かなものとするでしょう。

４．成功の終曲～成功体質の定着化

以上の問題対応は、第７群に示された【人材育成／ノウハウ継承】を自ずと実現し、プロジェクトの最終工程における【学習＝振り返り】の実行によって合理的かつ妥当な開発スタイルを良き習慣として定着させると同時に、将来にわたる好循環の連鎖を生み出していくでしょう。

チェックタイミング時系列順　Check　List　一覧

【開発工程ごとのチェックリスト】

▶Check Timing：開発準備時

▶Check Timing：開発全工程

▶Check Timing：各工程開始前

▶Check Timing：事前準備工程

▶Check Timing：見積り回答前

▶Check Timing：見積り時、要件定義時

▶Check Timing：要求仕様書発行・受領時

▶Check Timing：仕様凍結準備時

▶Check Timing：仕様調査時

▶Check Timing：開発着手前

▶Check Timing：設計・製造着手前

▶Check Timing：設計時

▶Check Timing：プログラミング着手前

▶Check Timing：テスト前

▶Check Timing：単体・結合テスト前

▶Check Timing：総合テスト前

▶Check Timing：製品リリース前

▶Check Timing：振り返り（ラップアップ）前

▶Check Timing：開発完了時

【問題対応に関するチェックリスト】

▶Check Timing：問題検討時

▶Check Timing：問題対応時

▶Check Timing：緊急対応前、プロジェクト危機時

▶Check Timing：改善活動着手前

【ドキュメントに関するチェックリスト】

▶Check Timing：変更･修正発生時、決定事項発生時、ドキュメント作成時

【教育、ノウハウ継承に関するチェックリスト】

▶Check Timing：教育実施前、学習実施前、ノウハウ継承前

引用

イラスト：　いらすとや　http://www.irasutoya.com/