オフショア開発向けＵＭＬ適用ガイドライン
Ver ２．０

2008 年 6 月

特定非営利活動法人　UMLモデリング推進協議会

オフショアソフトウェア開発部会

目次

１．目的

（１）（オフショア開発全体における）本ガイドラインの位置付け

オフショアでのシステム開発においても、オフショアではない国内開発と同様、要求/設計/構築/テストの開発サイクル各工程に関するテクノロジー視点での開発方法論、プロセスやコストに関するプロジェクト管理手法、成果物の品質管理手法などが利用される。

本ガイドラインは、その中でも”UML を用いたオフショアでのソフトウェア開発”に焦点をあて、オフショア開発特有の、かつ、UML を用いた開発に共通に見られる課題点への Hints & Tips を、実践経験に基づきまとめたものである。

オフショア開発全体における本ガイドラインの位置付けを次に説明する。

オフショア開発全体には、以下の要素が必要である。

①開発方法論（開発プロセス、開発手順、開発技法）

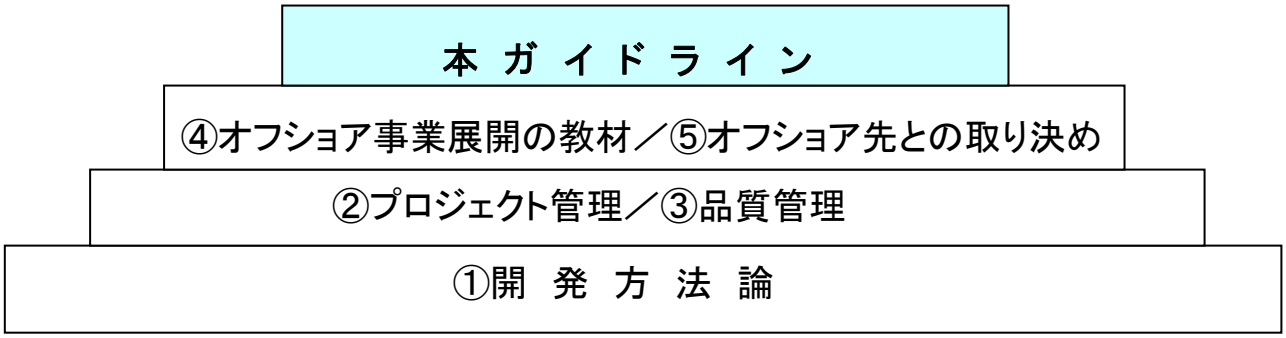
②プロジェクト管理（プロジェクトマネジメント計画書などに基づいた管理）

③品質管理（品質マネジメント計画書などに基づいた管理）

④オフショア事業展開の教材（オフショア事業展開事例集（市販本）、オフショア開発メルマガ、各社ノウハウなど）

⑤オフショア先との取り決め（オフショア先との作業範囲、スケジュール、納入物、価格などを決めたもの）

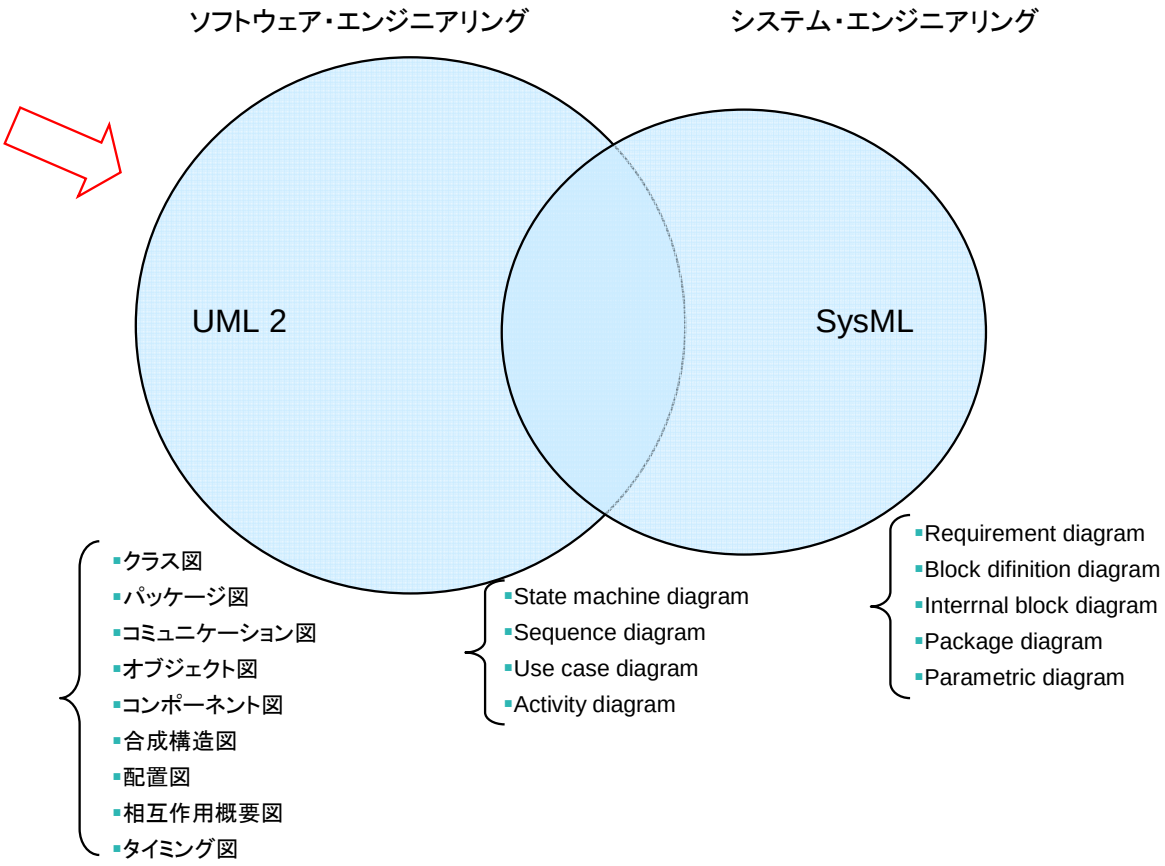
システム開発を行うには、上記の①～③を必要とする。オフショア開発では、さらに④⑤を必要とする。①～⑤は、オフショア開発に先立ち、各社が準備する必要がある。本ガイドラインは、モデリングを中心とした共通的な開発方法、開発手順、成果物を説明したものであり、①～⑤を補完するもの（ノウハウ）である。既に①～⑤が準備されていることを前提としており、それらに付加して使用することにより、オフショア開発の効率向上を目的としたものである。したがって、各社が決めるべき作業内容、作業手順などには言及していない。



（２）本ガイドラインでの対象

本ガイドラインでは、以下の範囲をガイド対象としている。

まず、「システム開発」という広義の開発を考えてみると、UML2 などを主に用いるソフトウェア・エンジニアリングと、SysML なども用いるシステム・エンジニアリングに分けられる。本ガイドラインでは、このうち、ソフトウェア・エンジニアリングに焦点をあてて検討した成果を記している。



次に、そのソフトウェア・エンジニアリングを詳細に見てみる。

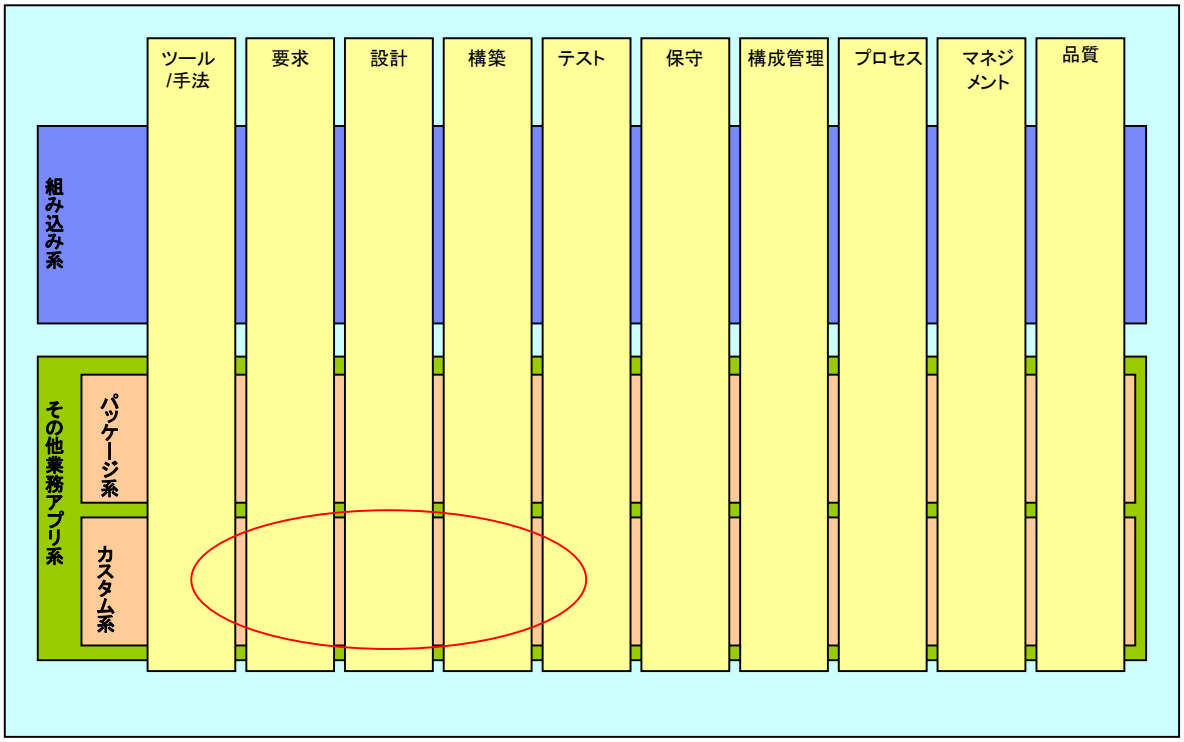
ソフトウェア・エンジニアリングは、大きく

- 要求～設計～構築～テスト～保守の、開発サイクルの各工程での開発技術の要素
- ツールや開発方法論、構成管理方法といった成果物作成支援の要素
- 開発のプロセスやコストに関するプロジェクト管理の要素
- 仕様書、プログラムなど成果物に関する品質管理の要素

に分けられる。

一方、そのエンジニアリング対象となるソフトウェアは、1) 組み込みを前提としたソフトウェアと、2)業務アプリケーション用ソフトウェアとに分けられる。 さらにその業務アプリケーションは、2a) ソフトウェア製品をパッケージとして利用し、その製品仕様のカスタマイズ範囲内で設計・構築を行うケースと、2b) 適用パッケージを考えにくく、自由度は大きいが、新規設計・構築部分が多くなるカスタム開発を行うケースとがある。

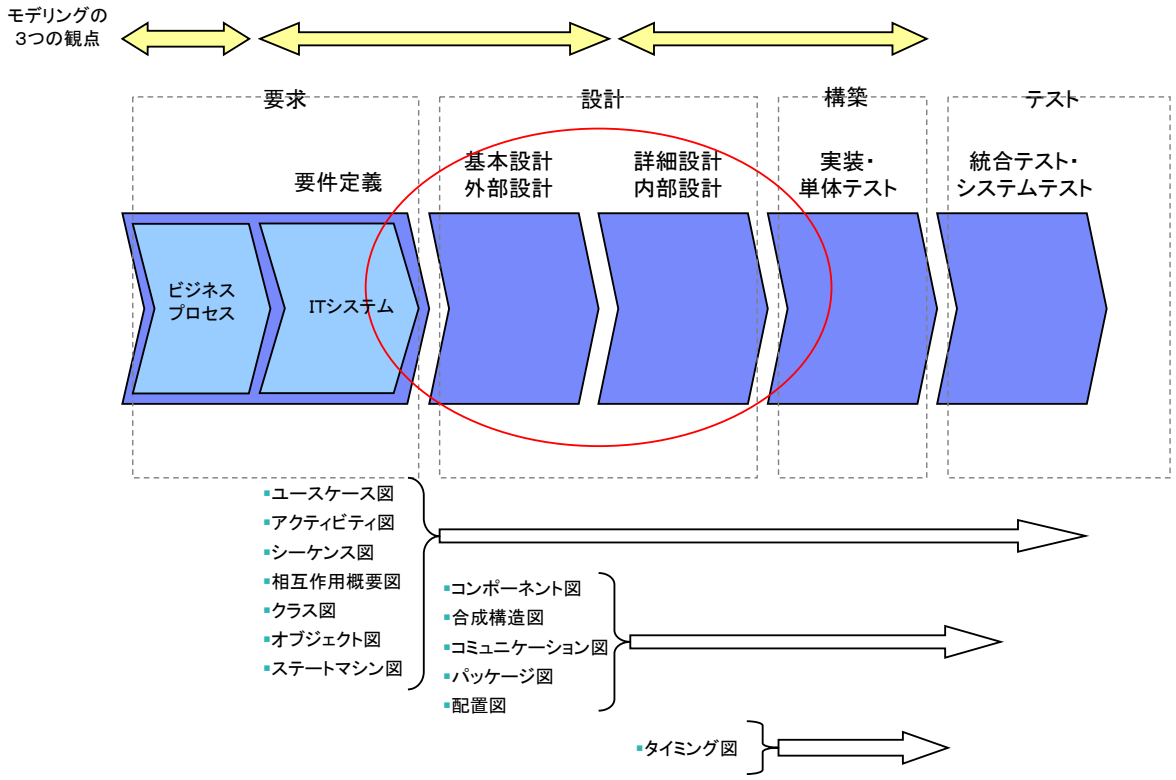
本ガイドラインは、この中で、開発サイクルの各工程での開発技術要素、それもパッケージではなく、カスタムの業務アプリケーション・ソフトウェア開発に焦点をあてて、Hints & tips とするものである。



さらに開発サイクルの各工程を、より詳細に見てみる。

開発サイクルは、要求工程の後半の要件定義から、設計を経て、構築、テストへと続いていく。 これを”UML”という切り口から見ると、ビジネスプロセスを実現する IT システムの要件定義を行う工程から、プログラミングを始める構築工程までが、UML を利用する局面の対象であると考えることができる。 具体的には、要件を UML により記述する、設計による仕様を UML により記述する、その仕様を UML によりオフショアメンバに伝え、UML に基づき実装を行う、といった場面になる。

本ガイドラインは、要件定義や外部、内部設計の各工程で UML により仕様記述を行う部分に着目することになるが、モデリングの観点からすると、仕様モデリングや実装モデリングが該当の範囲となる。



なお、ここでは要件定義、外部設計、内部設計、実装という名称を用いたものの、後述の本ガイドライン内では、

- 業務分析
- 要求分析
- システム分析
- アーキテクチャ設計
- 詳細設計
- 実装、単体テスト
- 結合テスト

の分け方・用語を採用しているので、注意されたい。

最後に、前述の UML 関連の工程において、オフショア開発の視点でどのように整理されるかについて触

れる。

本ガイドラインにおいて、オフショアは、外部設計（アーキテクチャ設計）完了後、内部設計（詳細設計）から実装を担当することを想定してまとめている。 開発者の視点では、

「オフショアのメンバに渡す仕様は、UML を用いて何をどう作成すればよいだろうか？」

「オフショア開発特有に気をつけるべき点は何になるのだろうか？」

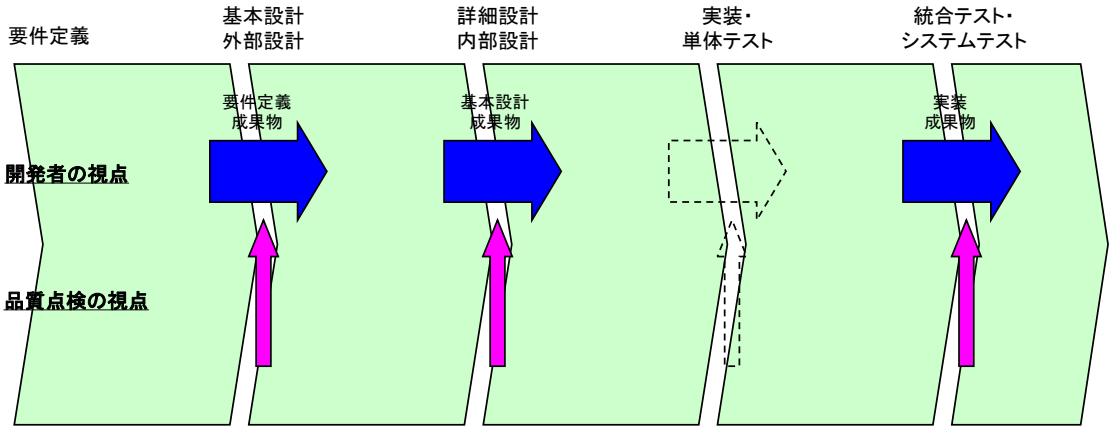
があるし、一方、そのオフショア成果の品質を点検する視点では、

「オフショア開発用に、どのような点に気をつけて点検するのがよいだろうか？」

「オフショア開発用には何が満たされていればよいと考えればよいだろうか？」

という関心事項が挙げられよう。

本ガイドラインは、特にこの点に着目したノウハウを示すものである。



		要件定義 → 基本設計	基本設計 → 詳細設計	詳細設計 → 実装	実装 → 統合テスト・システムテスト
開発者の視点	日本側	どのように利用・作成すればよいのか	どのようなものを渡せばよいのか	ー	
	オフショア側	ー	どのようなものを受取りたいのか	ー	
品質点検の視点	日本側	どのような内容を点検すればよいのか	どのような内容・レベルを達成すればよいのか	ー	
	オフショア側	ー	どのようなものを受取りたいのか	ー	

以上、本ガイドラインで、どの範囲を対象としているのかを示した。

オフショア開発において典型的なパターンである「オフショアでの内部設計～構築」を想定し、日本側での要件・設計のまとめと、それらのオフショアメンバへの伝達、オフショア側・日本側双方でのオフショア開発成果の点検、に関してが本ガイドラインの対象である。

補足として、やや異なる視点での対象範囲を示してみる。

要求を分析する、それに対応するシステムを設計する、仕様をオフショアに伝える、伝達結果に基づいて作成し、その成果の品質を点検する、という開発過程において、

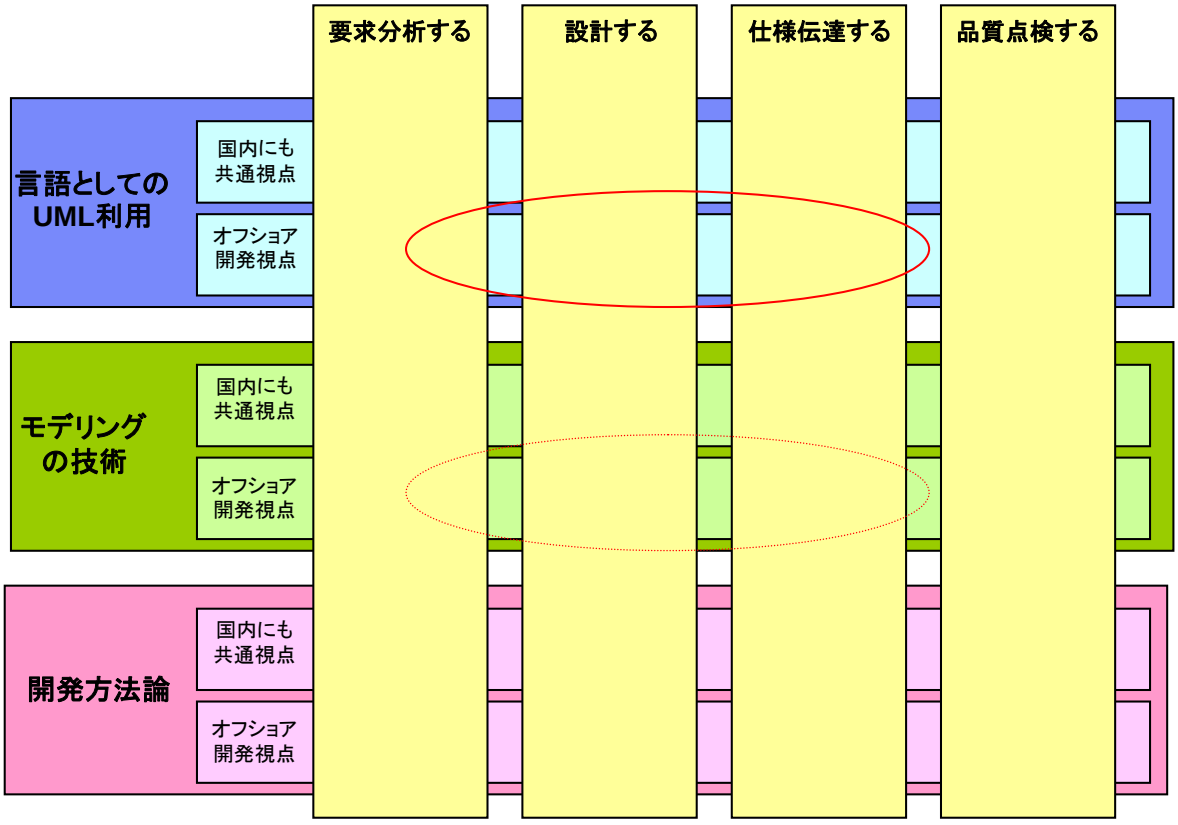
・　国内開発にも共通の点と、オフショア開発特有の点

があり、そこでオフショア開発特有の点に着目しても、開発に利用する要素として、

・　言語として UML 利用する際の技術とノウハウ
・　モデリングを行うにあたっての技術とノウハウ
・　経験に基づく知的資産となった開発方法論の技術とノウハウ

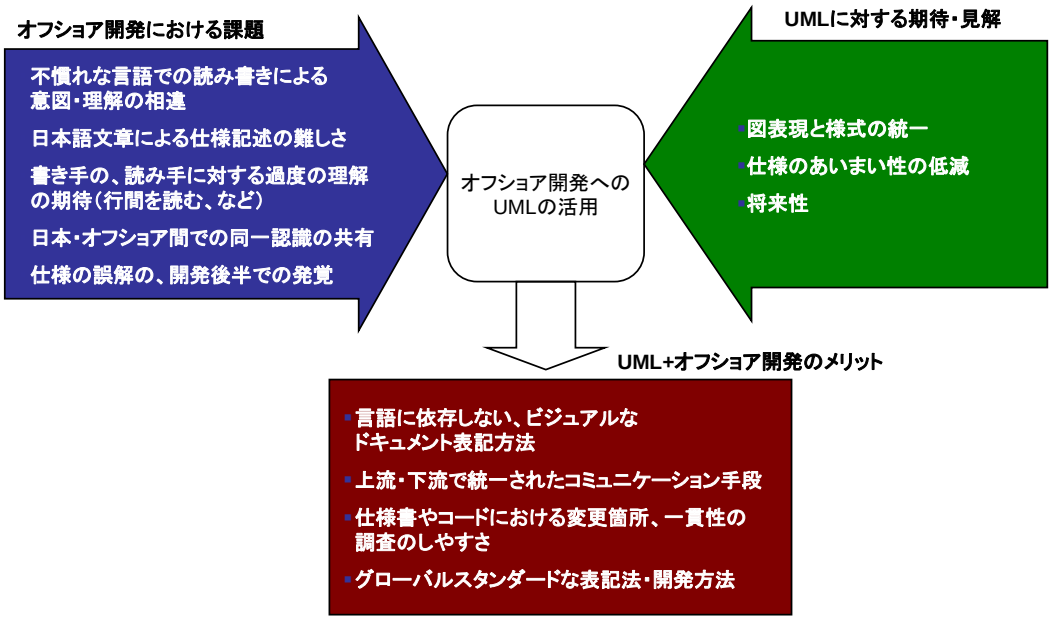
が関連事項として挙げられる。
しかし、本ガイドラインでは、「UML を利用するオフショア開発で、要求や仕様を記述し、仕様を伝えて、成果を点検する」という点に絞って記述している。すなわち、UML そのものの解説や使い方、モデリングの解説や具体例、開発方法論の解説や使い方については、本ガイドラインで示してはいない。

以上のことから、本ガイドラインの読者には、UML、モデリング、開発方法論、ソフトウェア開発技術要素それぞれに知識があることを前提とし、本ガイドラインをそれらの知識をベースに前提知識に紐付けながら読み進められることを期待したい。（本ガイドラインは、「オフショア開発に固有でない UML、モデリング、開発方法論、品質管理、プロジェクト管理のすべても解説し、かつ、オフショアソフトウェア開発のノウハウも合わせて詳述する」ことを目的としていない。）



２．オフショア開発における現状の問題点と対策

オフショア開発の発注側（日本側）・受注側（オフショア側）それぞれに対して、オフショア開発における UML 活用状況の現状を把握するべくアンケートおよびヒアリングを実施した。アンケート結果のポイントをまとめると下図のようになる。発注側・受注側共に、UMLに対する期待が大きく、UMLの活用で、課題がかなり解決されると認識している。



（１）アンケート調査概要

＜発注側（日本側）アンケート＞

調査対象： UMTP 参加企業および関連企業

調査時期： 2006年11月

有効回答数： 70 （主にPM/PLの回答　平均開発経験年数 16.2 年　オフショア開発経験年数　2.2 年）

＜受注側（オフショア側）アンケート＞

調査対象： UMTP 参加企業および関連企業のオフショア発注先の中国企業

調査時期： 2007年3月

有効回答数： 91 （主にPM/PLの回答　平均開発経験年数 4.4 年）

※今回は、日本からのオフショア先として現状では最も多いと考えられる中国を調査対象とした。

（２）アンケート結果内容（一部抜粋）

アンケート結果より、オフショア開発における課題、UMLに対する期待・見解とオフショア開発にUMLを活用した時のメリットについて下記に抜粋する。

オフショア開発における課題

表2.1.1 オフショア開発における課題と深刻度 － 発注側（日本側）

		極めて深刻	かなり深刻	深刻	やや深刻	全く深刻ではない	深刻度が高い割合(%)
1	言語や文化の差が原因で誤解が生じる	4	19	14	13	10	38
2	オフショア企業側に伝達内容を正確に伝えるのに時間がかかりすぎる	1	15	16	18	10	27
3	仕様書の曖昧さが原因で誤解が生じる	1	11	14	22	13	20
4	仕様書とオフショア企業側から納品されたソースコードに乖離が生じる	4	17	21	14	4	35
5	仕様書に漏れ・抜けが起こる	2	13	19	16	10	25
6	オフショア企業からくる大量の質問への対応で業務が圧迫される	6	19	19	13	2	**42**
7	テスト段階以降になって品質の問題が多く露呈する	3	14	11	26	6	28
8	仕様書を詳細に記述するための負担が大きい	3	14	22	14	8	28
9	オフショア企業側でのテスト内容に不備、不足がある	5	13	14	22	6	30
10	オフショア企業の担当者が離職することで、業務の継続に支障がある	12	22	13	9	3	**58**
11	オフショア企業が保守・メンテナンスを担当する場合、バグの改修におけるリードタイムが大きい	11	23	16	7	2	**58**
12	オフショア企業と同じイメージで効果的な情報共有ができない	7	27	17	6	2	**58**

表2.1.2 オフショア開発における課題と深刻度 － 受注側（オフショア側）

（注： 一部質問が異なるため、通し番号が飛んでいる）

		極めて深刻	かなり深刻	深刻	やや深刻	全く深刻ではない	深刻度が高い割合(%)
1	言語や文化の差が原因で誤解が生じる	6	7	11	31	36	14
2	発注元（日本）側に伝達内容を正確に伝えるのに時間がかかりすぎる	2	6	14	32	37	9
3	仕様書の曖昧さが原因で誤解が生じる	9	28	24	28	2	**41**
4	仕様書とオフショア企業側から納品されたソースコードに乖離が生じる	13	12	10	27	28	28
5	仕様書に漏れ・抜けが起こる	12	23	33	19	4	38
12	発注元（日本）側と同じイメージで効果的な情報共有ができない	7	27	17	6	2	**58**
13	プロジェクト内メンバ間のコミュニケーションができていない	4	15	16	23	32	21

発注側（日本側）・受注側（オフショア側）共に相手と「12. 同じイメージで効果的な情報共有ができない」点を深刻な問題点と捉えている。受注者側は、「3. 仕様書の曖昧さが原因で誤解が生じている」ことを問題視しているのに対し、仕様書の書き手である発注側はあまり問題としていない。しかし、実際には仕様書で伝えきれないために発注側は「6. オフショア企業からくる大量の質問への対応で業務が圧迫される」という構図が推測できる。

オフショア開発における課題に対する解決策

前記の「12. 同じイメージで効果的な情報共有ができない」という課題に対して、仕様書やレビューでUMLに限らず統一的に図表を利用している人は表 2.2 の通り全体の74%であり、大多数の人が図表の利用を実践し重要性は認識している。

表2.2 仕様書やレビューで図表やUMLを使う割合 – 発注側（日本側）

	ケース	平均割合(%)	
(a)	UML （そのダイアグラム1つでも） を利用する	19%	
(b)	UMLではないが、組織内で図表の描き方を統一して利用する	38%	
(c)	(a)(b)には該当しないが、個人的に図表の描き方を統一して利用する	17%	74%
(d)	統一されていない描き方で図表を使用する	14%	
(e)	文字は用いるが、その中に図表は使わない	5%	
(f)	口頭で行う	3%	
(g)	その他の方法	4%	

さらに、UML に対しての認識は下記の調査の通りであり、単なる図表に比べてより明確な効果が期待されていることが伺える。

表2.3.1 UML に対する期待・見解 – 発注側（日本側）

	意見	全く反対	どちらか言うと反対	どちらとも言えない	ある程度賛成	非常に賛成	賛成の割合(%)
1	UMLは上流工程に関わる人だけが知っていればよい	29	28	8	2	0	3
2	UMLを使っても目に見えるほどのコストは下がらない	3	9	28	21	6	40
3	周囲にUMLを使っている事例がほとんどない	10	17	9	22	9	46
4	UMLは教育が難しい、教育にコストがかかる	4	10	17	31	5	**54**
5	UMLによって生産性が向上する	1	4	38	19	5	36
6	UMLによって下流工程の手戻りが少なくなる	2	5	30	24	6	45
7	UMLによって品質が向上する	2	4	27	29	5	**51**
8	UMLは専門的すぎて難しい	9	22	24	10	2	18
9	UMLを使うと見積が難しくなる	5	20	32	8	2	15
10	UMLを使うと進捗管理が難しくなる	6	20	37	4	0	6
11	仕様にUMLを使うと曖昧性を低減することができる	1	2	18	37	9	**69**
12	仕様にUMLを使うと仕様書変更の影響範囲が把握しやすくなる	2	6	17	32	10	**63**
13	レビューでUMLを使うと、説明内容を効果的に伝えることができる	2	4	15	35	11	**69**
14	UMLを使うと開発に混乱が生じる	12	29	20	6	0	9
15	上流工程ではUMLよりも、それに特化したツールの方が優れている	5	15	36	9	2	16
16	UMLを使うと発注元や発注先との間のコミュニケーションが促進される	2	3	30	23	9	**48**
17	UMLを使うと維持管理のトータルコストを下げることができる	1	13	36	12	5	25
18	UMLは開発を自動化するのに有効である	3	15	26	18	5	34
19	UMLは国内開発よりオフショア開発でより沢山利用されている	3	6	42	12	3	23
20	UMLは国内開発よりオフショア開発の方が向いている	4	6	29	24	3	41

表2.3.2 UML に対する期待・見解 － 受注側（オフショア側）

（注： 一部質問が異なるため、通し番号が飛んでいる）

	意見	全く反対	どちらか言うと反対	どちらとも言えない	ある程度賛成	非常に賛成	賛成の割合(%)
1	UMLは発注元（日本）側だけが知っていればよい	51	22	4	4	1	6
2	UMLを使っても目に見えるほどのコストは下がらない	7	23	35	15	2	21
3	周囲にUMLを使っている事例がほとんどない	27	19	23	9	4	16
4	UMLは習得するのにコストがかかる	4	18	26	27	6	41
5	UMLによって生産性が向上する	0	0	14	47	21	**83**
6	UMLによって発注元（日本）からの手戻りが少なくなる	1	6	21	33	18	**65**
7	UMLによって品質が向上する	0	1	26	36	19	**67**
8	UMLは専門的すぎて難しい	10	21	20	23	8	38
9	UMLを使うと見積が難しくなる	4	32	39	7	0	9
10	UMLを使うと進捗管理が難しくなる	6	45	27	3	1	5
11	仕様にUMLを使うと曖昧性を低減することができる	1	3	12	45	21	**80**
12	仕様にUMLを使うと仕様書変更の影響範囲が把握しやすくなる	0	3	18	37	24	**74**
13	レビューでUMLを使うと、説明内容を効果的に伝えることができる	1	4	19	35	23	**71**
14	UMLを使うと開発に混乱が生じる	24	37	15	5	1	7
16	UMLを使うと発注元（日本）との間のコミュニケーションが促進される	0	3	21	40	17	**70**
18	UMLは開発を自動化するのに有効である	1	3	23	43	12	**67**
19	UMLは他の開発よりオフショア開発でより沢山利用されている	2	12	44	20	4	29
20	UMLはオフショア開発に向いている	4	4	43	27	6	39
21	UMLに関する本は沢山ある	0	13	16	30	23	**65**
22	UMLは知っていると昇進や転職のときに有利になる	1	4	25	28	24	**63**

発注側（日本側）・受注側（オフショア側）共に UML の「11. 仕様の曖昧性を低減」、「12. 仕様変更の影響範囲が把握しやすい」、「13. レビューで UML を使うと説明内容を効果的に伝えることができる」といった効果を認識し、双方間での「16. コミュニケーションが促進」される結果「7. 品質が向上する」と考えている。特に、受注側の UML に対する期待が高い。一方で、発注側の多くは「4. 教育が難しい、教育コストがかかる」といった課題をあげている。

ＵＭＬをオフショアに使った時のメリット

表2.4.1 UML をオフショア開発で使った時のメリット － 発注側（日本側）

	メリット	全く重要でない	ある程度重要	非常に重要	加重合計
1	グローバルスタンダードなドキュメント表記方法が使える	13	26	22	**131**
2	ビジュアルなドキュメント表記方法が使える	10	26	24	**134**
3	ユーザ企業側でも分かるようなドキュメント表記方法が使える	13	27	20	**127**
4	ソースコードを見なくても仕様書で修正箇所が調査できるので保守しやすい	14	31	15	**121**
5	開発の上流から下流までシームレスになってい	16	27	18	**124**
6	オフショア開発と国内開発で共通の開発支援ツールが使用できる	9	21	30	**141**
7	上流と下流で統一されたコミュニケーション手段がある	8	26	26	**138**
8	グローバルスタンダードな開発方法が使える	10	27	24	**136**
9	開発されたものが再利用できる	15	28	17	**122**
10	できるだけ自然言語に依存しない表記法が使え	13	30	17	**124**
11	仕様書とソースコードの一貫性が確認できる	8	31	21	**133**
12	発注先の国の言語（例：中国語）によるコミュニケーションできる	24	22	14	**110**
13	品質指標が確立されていて、品質に保障が取れる	13	21	26	**133**

表2.4.2 UML をオフショア開発で使った時のメリット － 受注側（オフショア側）

	メリット	全く重要でない	ある程度重要	非常に重要	加重合計
1	グローバルスタンダードなドキュメント表記方法が使える	7	41	43	**218**
2	ビジュアルなドキュメント表記方法が使える	5	44	42	**219**
3	ユーザ企業側でも分かるようなドキュメント表記方法が使える	3	43	45	**224**
4	ソースコードを見なくても仕様書で修正箇所が調査できるので保守しやすい	5	43	43	**220**
5	開発の上流から下流までシームレスになってい	2	48	41	**221**
6	オフショア開発と国内開発で共通の開発支援ツールが使用できる	17	42	32	**197**
7	上流と下流で統一されたコミュニケーション手段がある	7	39	58	**259**
8	グローバルスタンダードな開発方法が使える	5	46	40	**217**
9	開発されたものが再利用できる	5	36	50	**227**
10	できるだけ自然言語に依存しない表記法が使え	10	53	28	**200**
11	仕様書とソースコードの一貫性が確認できる	5	26	60	**237**
12	発注元の国の言語（日本語）によるコミュニケーションできる	5	35	51	**228**
13	品質指標が確立されていて、品質に保障が取れる	5	20	66	**243**

オフショア開発の場合、オフショア先とのコミュニケーションが重要になる。そのため、「12. 相手の国の言語によるコミュニケーション」というのは重要である。日中オフショアの場合には、中国側が日本語を理解してくれるケースが多いので、発注側の結果ではさほど UML 利用のメリットを感じていないが、受注側（オフショア側）はメリットと感じている。上流工程を発注側（日本側）で下流を受注側（オフショア側）が分担するケースが多いと思われるが、「7. 上流と下流で統一されたコミュニケーション手段がある」というメリットは、発注側・受注側共に認識している。

また、UMLの特徴である「2. ビジュアルな表記法」、「1. グローバルスタンダードな表記法」・「8. グローバルスタンダードな開発方法」などもオフショア開発においては重要なポイントと考える。

さらに保守性の観点では、「11. 仕様書とソースコードの一貫性が確認できる」などの支持も多数ある。

（3）ヒアリング

前記のアンケートによる調査結果を補完するために下記のヒアリングを実施し、本ガイドラインが受注側（オフショア側）からみても納得感のある内容となっていること、中国だけでなくインドにおいても適用可能な汎用性のあるものになっていることを確認している。

＜中国のPM／PLへのヒアリング（一部は記述式のアンケート）＞
目的： 定量的なアンケート調査では掴みきれない実態の把握のため
時期： 2007年11月

＜インドのアーキテクトへのヒアリング＞
目的： オフショア開発で日－中を先行するUS－インドにおいてのUML利用状況把握のため
時期： 2008年3月

３．UML モデリング

3.1 ＵＭＬの特徴

（１）ＵＭＬとは

UML(Unified Modeling Language)は、OMG(Object Management Group)により標準化された、ソフトウェアの成果物を仕様化、図式化するときに使用する表記法である。

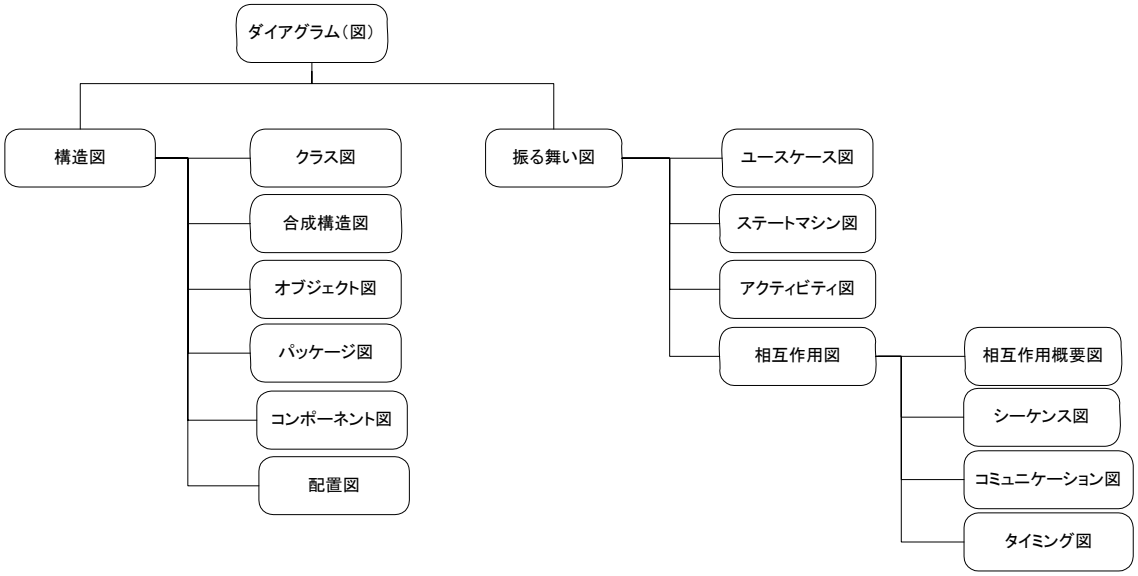
（２）ＵＭＬの特徴

UMLの特徴としては次のようなものが挙げられる。

- 世界標準である。
  - 世界中の異なる国間でも、異なるプロジェクト間においても、コミュニケーションが容易である。
- 表現力が高く、しかも理解が容易である。
  - UMLが提供する複数のダイアグラム（図）により、多様な視点で分析・設計対象を表現できる。しかし、どのダイアグラムも直感的で理解しやすい。
- 開発の全ての工程で一貫して使用できる。
  - 開発の各工程における成果物のトレーサビリティが確保されており、各工程間で開発者の間のコミュニケーションが容易である。

（３）ＵＭＬのダイアグラム（図）

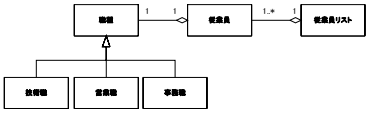
UMLのダイアグラム（図）には次のようなものがある。



・構造図

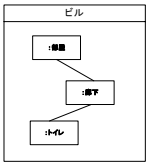
●クラス図

分析・設計領域の物や事を概念的に捉え、それをクラスおよびその関係により静的に表現する。



●合成構造図

クラスやコンポーネントの内部構造を階層的に表現する。

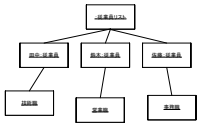


●パッケージ図

ＵＭＬ要素をまとめるパッケージ間の関係を表現する。

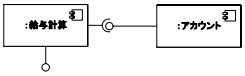
●オブジェクト図

システムのある瞬間の状態をオブジェクトおよびその関係により表現する。



●コンポーネント図

ソフトウェアコンポーネント（再利用可能な部品）の構成を表現する。



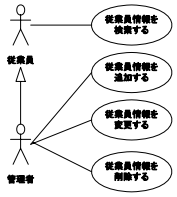
●配置図

システムの実行環境を表現する。

・振る舞い図

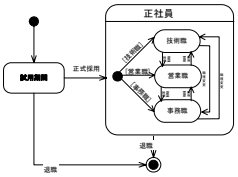
●ユースケース図

システムが提供する機能と外部との関係を表現する。



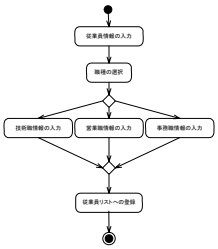
●ステートマシン図

一つのオブジェクトの時間経過に伴う状態の変化を表現する。



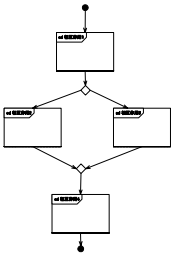
●アクティビティ図

業務フロー、イベントフロー、アルゴリズムの流れ（フロー）を表現する。



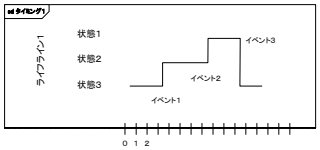
●相互作用概要図

複数の相互作用図の関係を概観し表現する。



●タイミング図

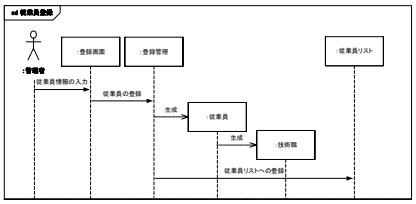
時間経過とともに変化するライフラインの状態の変化やメッセージのやり取りを表現する。



○

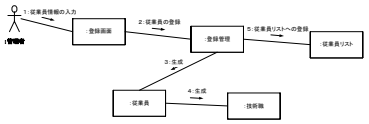
●シーケンス図

相互作用（分析・設計対象内に存在する“物”や“人”のやり取り）を時系列に表現する。



●コミュニケーション図

相互作用（分析・設計対象内に存在する“物”や“人”のやり取り）を“物”、“人”を中心にその接続関係に着目して表現する。



3.2 前提とするＵＭＬモデリングスキルレベル

モデルを利用したオフショア開発を効率的に行う為には、発注側、受注側ともある一定のUMLモデリングのスキルを保持している必要がある。これらのスキルを保持していない場合は、事前に学習、教育を行っておくべきである。また UML モデリングスキルレベルは、分析者、設計者、実装者、アーキテクト、プロジェクトリーダー、プロジェクトマネージャーなど立場によって、必要とされるレベルは異なることになる。

尚、ここで定義しているスキルレベルは、モデリングに関するもののみであり、分析者、設計者、実装者、アーキテクト、プロジェクトリーダー、プロジェクトマネージャーに一般的に必要とされるスキル、知識を前提としたものである。これら一般的に必要とされる、スキル、知識についてはここでは定義していない。

【目的】

オフショア開発に UML を使用した成果物のやりとりを、発注側と受注側で行うときに、双方が一定の UML モデリングスキルレベルを保持していれば、必要最小限の成果物のやり取りのみで、正解に意図が伝わる。開発要員の UML モデリングスキルが十分でないとき、成果物の意図が正確に伝わらないばかりか、UML やモデリングについての説明を開発担当者が行う必要が発生し、その分無駄な工数が発生してしまうことになる。

【詳細・補足】

〇レベルの測定（確認）方法

UMTP では、モデリングスキルレベルを次のように、L1～L4 まで定義している。

レベル	モデリングスキル	説明
L4	実践に基づいてモデリングを指導できる	L3のスキルを有し、開発プロジェクトでモデリングを一定数あるいは期間実践した経験を持つ
L3	実務でモデリングが実践できる	拡張性や変更容易性の点で高品質なモデルを定義できる
		ビジネスモデリング、分析、アーキテクチャ設計、組み込み開発を行うための専門的な知識を備えている
L2	UMLモデルの読み書きが普通にできる（モデリングリテラシーがある）	開発範囲の一部を担当し、モデリングができる
		他者のモデルの意味を理解できる
L1	簡単なUMLモデルの意味が分かる	UMLなどを使ってモデリングを行う最低限の知識を持っている

UMTPでは、L1～L2 まで、このスキルレベルに対応した試験を実施している。（L3 は 2008 年４月より実施）この試験に合格することで、スキルレベルの測定が可能である。

また L1 レベルについては、L1-T1 と L1-T2 の２つの試験を実施している。

L1-T1→UML の表記法を知っている。

L1-T2→UML で描かれたモデルを読み、良し悪しを判断できる。

※中国では、L1-T2 と L2 の試験について、実施中である。

実際の開発においては役割により、それぞれ次のようなレベルが最低限要求される。

実装者・・・UMLのモデルを見て、ソースコードを作成する。→L1-T2 レベル

分析者（一般）・・・分析者（上級）の指示のもとUMLの分析モデルを独力で作成する。→L2 レベル

設計者・・・アーキテクチャに基づきUMLの設計モデルを独力で作成する。→L2 レベル

分析者(上級)・・・初期分析モデルを作成する。→L3 レベル
アーキテクト・・・アーキテクチャモデルを作成する。→L3 レベル
プロジェクトリーダー・・・モデル分析・設計方針のレビュー、決定を行う。→L3 レベル
プロジェクトマネジャー・・・プロジェクトの最高責任者→L1～L2 レベル

	L1-T1	L1-T2	L2	L3
実装者	○	○	-	-
分析者(一般)	○	○	○	-
分析者(上級)	○	○	○	○
設計者	○	○	△	-
アーキテクト	○	○	○	○
プロジェクトリーダー	○	○	○	○
プロジェクトマネジャー	○	○	△	-

○スキルアップ
UMLの知識(L1-T1)を得たり、モデリング初級(L1-T2)のスキルを習得することは、市販の書籍を1冊読むことで可能である。しかし、UMLのモデルを独力でつくるスキル(L2)については、書籍だけでそのスキルを習得することは困難である。まずは、トレーニングやメンタリングによりモデリングのコツを掴み、その後、たくさんのモデルを見て、理解し、描いてみる必要がある。
さらに、上級のモデリングスキルレベル(L3)では、実務レベルで経験を積む必要がある。

○参考情報
・UMTP 基準準拠
市販の書籍には、UMTP基準準拠表示されているものがある。
これは、モデリング用語集　UML 編　第 2 版に適合しているかなどを審査し、許可するものである。
UMTP 基準準拠の書籍には以下のものがある。
http://www.umtp-japan.org/modules/data4/index.php?id=7

・トレーニングテキスト認定
トレーニングテキストに関しては、モデリング用語集　UML 編　第 2 版に適合しているかに加えて、
モデリングに必要な知識をがトレーニングテキスト中で解説されているかを審査し、認定を行っている。
認定トレーニングコースは次のものがある。
http://www.umtp-japan.org/modules/data4/index.php?id=8

【関連情報】
なし

４． UML の適用範囲と開発のノウハウ(Hints & Tips)

2章のアンケート結果から、多数の人が、オフショア開発に UML を適用した時のメリットを認識していることが分かる。しかし UML をオフショア開発に適用すれば、すぐに効果がでるのではなく、開発のどの局面に、どのように適用するかが重要である。本章では、オフショア開発を成功させるには、各工程で、どのような作業をすればよいかのノウハウを説明する。ノウハウは、UML を中心に説明するが、UML に特化しないものも含んでいる。

図4．1に開発の流れ、表4．1に各工程の目的、作業内容、ワーカー、入力成果物、出力成果物を示す。図4．1は主要な成果物のみを記載しており、成果物の詳細は表4．1を参照すること。開発の各工程で注意すべき事項（ノウハウ）をポイント番号で図4．1に示す。各工程のポイント番号ごとに、ノウハウを以下の形式で説明する。

・ポイント番号とタイトル
・内容
・目的
・詳細・補足
・関連情報

なお、条件が整い実施できる場合には効果が高いが、オフショア開発の初期段階では無理して実施する必要のないノウハウについては、Option（状況に応じて選択可能）の扱いとしている。

図 4.1 開発の流れ

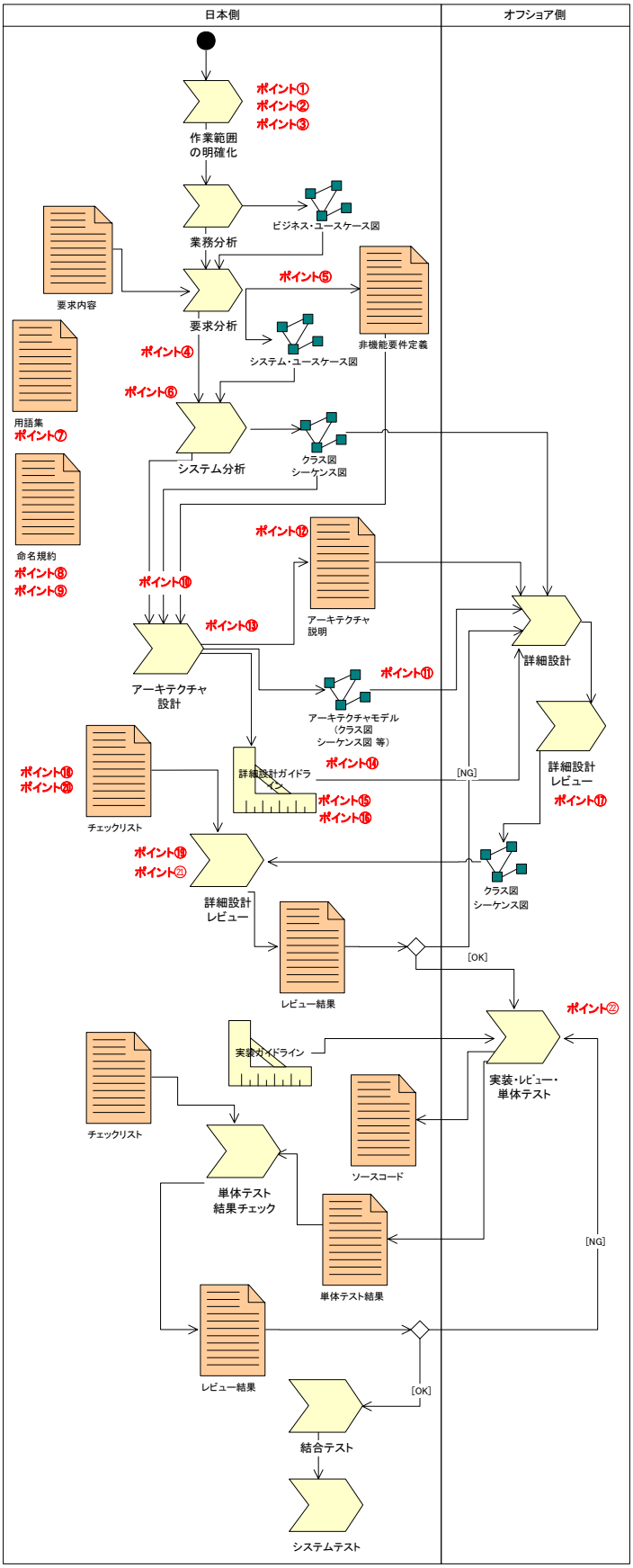

日本側
オフショア側
ポイント①
ポイント②
ポイント③
作業範囲の明確化
業務分析
ビジネス・ユースケース図
要求内容
要求分析
ポイント⑤
非機能要件定義
システム・ユースケース図
ポイント④
ポイント⑥
用語集
ポイント⑦
システム分析
クラス図
シーケンス図
命名規約
ポイント⑧
ポイント⑨
ポイント⑫
ポイント⑩
アーキテクチャ説明
ポイント⑬
詳細設計
アーキテクチャ設計
ポイント⑪
アーキテクチャモデル
（クラス図
シーケンス図 等）
詳細設計ガイドライン
ポイント⑭
[NG]
詳細設計レビュー
ポイント⑰
ポイント⑯
ポイント⑳
チェックリスト
ポイント⑮
ポイント⑱
ポイント⑲
ポイント㉑
詳細設計レビュー
クラス図
シーケンス図
[OK]
レビュー結果
ポイント㉒
実装ガイドライン
実装・レビュー・単体テスト
チェックリスト
ソースコード
単体テスト結果チェック
[NG]
単体テスト結果
レビュー結果
[OK]
結合テスト
システムテスト

表4．1各工程の説明

工程名：業務分析	
目的（日本側）： 今回のシステムの業務内容を理解し、明確にすることで、要求分析工程において、業務のどの部分をシステム化するかを明確にしやすくする。	目的（オフショア側）： 今回対象とするシステムの業務内容を理解することで、次工程以降の作業が円滑に進むように準備をする。
作業内容（日本側）： 業務内容を理解し、明確にする。必要に応じて業務の流れを整理する。	作業内容（オフショア側）： 今回対象とするシステムの業務内容を理解する。（上流に参加している場合）
ワーカー（日本側）： PM 分析者	ワーカー（オフショア側）： PM システム分析者（上流に参加している場合）
入力成果物： ・業務フロー	入力成果物： ・ビジネス・ユースケース図 ・業務フロー ・ビジネス・ユースケース記述 ・概念クラス図
出力成果物： ・ビジネス・ユースケース図 ・業務フロー ・ビジネス・ユースケース記述 ・概念クラス図	出力成果物：
参考情報： ・用語集 ・命名規約	参考情報 ・用語集 ・命名規約

工程名：要求分析	
目的（日本側）： システムの機能的要求および非機能的要求を明確にする。	目的（オフショア側）： 詳細設計以降を担当する場合は、直接的にこの工程の成果物を利用するわけではない。 この工程の成果物から、今回システム開発対象にしているシステムの機能を理解しておく。 システム機能を知っておくことで、なぜこのような設計をすべきかが理解できるようになり、詳細設計以降のコミュニケーションがスムースに行なわれる。
作業内容（日本側）： 顧客のシステムに対する要求を明確にする。	作業内容（オフショア側）： 出力成果物を参考にして、対象システムの機能を理解する。（上流に参加している場合）
ワーカー（日本側）： 要求分析者	ワーカー（オフショア側）： システム分析者（上流に参加している場合） 詳細設計者（上流に参加している場合）
入力成果物： ・ビジネス・ユースケース図 ・業務フロー ・ビジネス・ユースケース記述 ・要求内容	入力成果物： ・システム・ユースケース図 ・システムユースケース一覧 ・ビジネス・ルール定義 ・システム・シーケンス図 ・システム・ユースケース記述 ・画面遷移図 ・画面・帳票設計書 ・非機能要件定義
出力成果物： ・システム・ユースケース図 ・システムユースケース一覧 ・ビジネス・ルール定義 ・システム・シーケンス図 ・システム・ユースケース記述 ・画面遷移図 ・画面・帳票設計書 ・非機能要件定義	出力成果物：
参考情報： ・用語集 ・命名規約	参考情報 ・用語集 ・命名規約

工程名：システム分析	
目的（日本側）： システム開発対象の領域の知識を自然な形で分析する。 開発、運用環境に依存しないモデルを作成することで、再利用性、保守性を向上させる。	目的（オフショア側）： 詳細設計以降を担当する場合は、直接的にこの工程の成果物を利用するわけではない。 この工程の成果物から、今回システム開発対象にしている領域の知識の習得を目的する。 システム開発対象の知識を得ることで、なぜこのような設計をすべきかが理解できるようになり、詳細設計以降のコミュニケーションがスムースに行なわれる。
作業内容（日本側）： システムの概要レベルの構造・関連を示すシステムのコンテキストを定義する。その後、システムの地理的分散や運用の複雑さを理解するための概要を整理し、システムの振舞いを分析する。	作業内容（オフショア側）： 出力成果物を参考にして、システム開発領域を理解する。（上流に参加している場合）
ワーカー（日本側）： システム分析者	ワーカー（オフショア側）： システム分析者（上流に参加している場合） 詳細設計者（上流に参加している場合）
入力成果物 ・システム・ユースケース図 ・システムユースケース一覧 ・ビジネス・ルール定義 ・システム・シーケンス図 ・システム・ユースケース記述 ・画面遷移図 ・画面・帳票設計書	入力成果物 ・分析クラス図 ・分析シーケンス図 ・オブジェクト図 （・コミュニケーション図） （・ステートマシン図）
出力成果物 ・分析クラス図 ・分析シーケンス図 ・オブジェクト図 （・コミュニケーション図） （・ステートマシン図）	出力成果物
参考情報： ・用語集 ・命名規約	参考情報 ・用語集 ・命名規約

工程名：アーキテクチャ設計	
目的（日本側）： システムの基盤構造を明確にする。	目的（オフショア側）： 詳細設計のインプットとする。
作業内容（日本側）： 候補となるアーキテクチャを定義し、分析結果を元に、再利用可能なモデル要素を取り込みながら、定義されたアーキテクチャを洗練する。コンポーネント設計、データベース設計を行いながら、アーキテクチャの詳細検討、クラス/サブシステムの識別、サブシステム間インターフェースの識別を行っていく。	作業内容（オフショア側）： 出力成果物を参考にアーキテクチャを理解する。 （上流に参加している場合）
ワーカー（日本側）： アーキテクト	ワーカー（オフショア側）： アーキテクト（上流に参加している場合） 詳細設計者（上流に参加している場合）
入力成果物 ・分析クラス図 ・分析シーケンス図 ・オブジェクト図 （・コミュニケーション図） （・ステートマシン図） ・非機能要件定義	入力成果物 ・設計クラス図　・分析クラス図 ・設計シーケンス図　・分析シーケンス図 ・オブジェクト図 ・パッケージ図 ・コンポーネント図 ・配置図 ・アーキテクチャ説明 ・詳細設計ガイドライン ・詳細設計チェックリスト ・データ・モデル図（E-R図）
出力成果物 ・設計クラス図 ・設計シーケンス図 ・オブジェクト図 ・パッケージ図 ・コンポーネント図 ・配置図 ・アーキテクチャ説明 ・詳細設計ガイドライン ・詳細設計チェックリスト ・データ・モデル図(E-R図)	出力成果物
参考情報： ・用語集 ・命名規約	参考情報 ・用語集 ・命名規約

工程名：詳細設計	
目的（日本側）： 作業工数が多い部分について、中国側に作業を委託する。	目的（オフショア側）： 実装作業を行なえるほど、モデルを詳細化する。
作業内容（日本側）： 中国側の作業が、アーキテクチャから逸脱していないかをチェックする。	作業内容（オフショア側）： アーキテクチャモデルやアーキテクチャ説明書を入力として、サブシステム間で調整を行いながら、詳細化して実装モデルを作成する。
ワーカー（日本側）： レビューアー	ワーカー（オフショア側）： アーキテクト（上流に参加している場合） 詳細設計者
入力成果物 ・詳細設計クラス図 ・詳細設計シーケンス図 （・オブジェクト図） ・詳細設計レビュー結果（オフショア側）	入力成果物 ・設計クラス図　・分析クラス図 ・設計シーケンス図　・分析シーケンス図 ・オブジェクト図 ・パッケージ図 ・コンポーネント図 ・配置図 ・アーキテクチャ説明 ・詳細設計ガイドライン ・詳細設計チェックリスト
出力成果物 ・詳細設計レビュー結果（日本側）	出力成果物 ・詳細設計クラス図 ・詳細設計シーケンス図 （・オブジェクト図） ・詳細設計レビュー結果（オフショア側）
参考情報： ・用語集 ・命名規約	参考情報 ・用語集 ・命名規約

工程名：実装・レビュー・単体テスト	
目的（日本側）： 引き続き、作業工数が多い部分について、中国側に作業を委託する。	目的（オフショア側）： システムを稼働させるソースコードを作成する。
作業内容（日本側）： 必要に応じてチェックを行なう。	作業内容（オフショア側）： 詳細設計の成果物をもとに、実装および単体テストを行なう。
ワーカー（日本側）： レビューアー	ワーカー（オフショア側）： 実装者
入力成果物 ・単体テスト結果	入力成果物 ・詳細設計クラス図 ・詳細設計シーケンス図 （・オブジェクト図）
出力成果物 ・チェック結果	出力成果物 ・ソースコード ・単体テスト結果
参考情報： ・用語集 ・命名規約 ・実装ガイドライン ・チェックリスト	参考情報 ・用語集 ・命名規約 ・実装ガイドライン ・チェックリスト

工程名：単体テスト結果チェック	
目的（日本側）： 品質に問題が無いかをチェックする。	目的（オフショア側）： 品質の最終確認をする。
作業内容（日本側）： 単体テストの結果のチェックを行う。	作業内容（オフショア側）： 必要に応じて、単体テスト結果を見直す。
ワーカー（日本側）： レビューアー	ワーカー（オフショア側）： 実装者
入力成果物 ・単体テスト結果	入力成果物 レビュー結果
出力成果物 レビュー結果	出力成果物 単体テスト結果
参考情報： ・用語集 ・命名規約 ・実装ガイドライン ・チェックリスト	参考情報 ・用語集 ・命名規約 ・実装ガイドライン ・チェックリスト

工程名：結合テスト	
目的（日本側）： 単体テストを行なったソースコードを、結合しテストを行なう。	目的（オフショア側）： ソースコードの品質をさらに高める。
作業内容（日本側）： 結合テストを行なう	作業内容（オフショア側）： 詳細設計、実装に関するバグがある場合は、修正する。
ワーカー（日本側）： テスター	ワーカー（オフショア側）： 詳細設計者 実装者
入力成果物 ・ソースコード ・単体テスト結果	入力成果物 ・詳細設計クラス図 ・詳細設計シーケンス図 ・ソースコード ・バグ票
出力成果物 ・結合テスト結果	出力成果物 ・詳細設計クラス図 ・詳細設計シーケンス図 ・ソースコード ・バグ票 ・チェックリスト
参考情報： ・用語集 ・命名規約	参考情報 ・用語集 ・命名規約

表 4.2 工程ごとの主要な成果物

工程	成果物セット	成果物	UML の図
業務分析	ビジネス・プロセス・モデル	ビジネス・ユースケース図	ユースケース図
		業務フロー	アクティビティ図
		ビジネス・ユースケース記述	
	ドメイン・モデル	概念クラス図	クラス図
			オブジェクト図
要求分析	ユース・ケース・モデル	システム・ユースケース図	ユースケース図
		システム・ユースケース一覧	
		ビジネス・ルール定義	
		システム・シーケンス図	シーケンス図
			相互作用概要図
		システム・ユースケース記述	アクティビティ図で作成する場合あり
	ユーザ・インターフェース・モデル	画面遷移図	ステートマシン図
		画面・帳票設計書	
	非機能要件	非機能要件定義	
	用語集	用語集	
システム分析	分析モデル	ロバストネス図	
		分析クラス図	クラス図
			オブジェクト図
		データ・モデル図（E-R 図）	
		相互作用図	シーケンス図
			コミュニケーション図
		状態図	ステートマシン図
アーキテクチャ設計/詳細設計	アーキテクチャ・モデル	アーキテクチャ説明	
		ソフトウェア構成	パッケージ図
			コンポーネント図
			合成構造図
		ハードウェア構成	配置図
		基本処理方式	シーケンス図
	設計モデル	設計クラス図/詳細設計クラス図	クラス図
			オブジェクト図
		データ・モデル図（E-R 図）	
		相互作用図	シーケンス図
			コミュニケーション図
		状態図	ステートマシン図
			タイミング図
		クラス仕様書	
実装	実装セット	実装コード	
単体テスト/結合テスト工程	テスト・セット成果物セット	テスト計画書	
		テスト仕様書	
		成果物	UML の図

表 4.3 開発のノウハウ(Hints & Tips) 一覧

工程	ﾎﾟｲﾝﾄ No	ノウハウ
作業範囲の明確化	01	作業範囲／作業分担の明確化
	02	利用する UML 図の確定
	03	必ずUMLである必要はない
業務分析/要求分析	04	上流工程への参画 *(Option)*
	05	非機能要件定義
	06	分析モデルで業務を理解
	07	用語辞書を作成する
	08	命名規約を作成する
	09	モデルの作成規約を作成する
システム分析/アーキテクチャ設計	10	共通機能の明確化
	11	アーキテクチャ・モデル
	12	アーキテクチャ説明成果物の作成
	13	パターンの活用
	14	仕様書の記述レベル、書式の指定
	15	詳細設計ガイドライン作成
	16	仕様未決定部分は明確に
詳細設計/実装	17	オフショア側での成果物のレビュー実施
	18	日本側はチェックを繰り返し行なう
	19	Validation と Verification
	20	モデルで詳細設計のレビュー
	21	UML 図間の整合性
	22	実装はツールのコード生成機能を使用する *(Option)*
ー	補足	日本におけるシステム開発の特徴

【ポイント 01 作業範囲／作業分担の明確化】

各工程の作業分担を明確にする。各工程の成果物（UML 図、及び UML 図以外）を決定する。オフショアでの開発担当領域を明確にしてモデリング範囲／テスト範囲を決める。

【目的】

システム開発においては、複数の企業（国内、海外）にてプロジェクトを推進する。
企業により、各作業工程の作業内容、成果物が異なる為、プロジェクト開始時に作業工程別の作業内容を明確にして作業工程にずれが生じないようにする。
発注側／受託側での成果物を明確にして納品時のトラブルの発生を削減する。

【詳細・補足】

各工程でのオフショア担当作業対象を明確にし、作業効率を向上させる。

例えば、開発環境面から運用要件、障害対策要件、セキュリティ要件に関する内容はオフショアでは対応困難な面があり、予め日本側で担当する領域／オフショア側で担当する領域に分けておくと良い。

〈全般〉

- ・工程単位に使用する成果物を記述する。
- ・各工程のインプット、アウトプットを明確にする。
- ・成果物の、書式・内容の粒度のガイドラインを明記する。
- ・成果物間に関連性があるものは、成果物間の不整合を防止するため、確認方法も明記する。

〈アーキテクチャ設計、詳細設計 （モデリング）〉

- ・基盤・他開発部分の提供箇所の明確化

〈実装・単体テスト （製造、作成）〉

- ・開発環境（サーバ、ソフトウェア種類・バージョン etc）の状態
- ・提供部品スケジューリング （提供可能時期）
- ・提供部品（一覧、外部仕様）
- ・使用する開発ツール

〈テスト〉

- ・開発環境の状態の考慮
- ・テスト・データの提供有無 （サンプル・データを配布した方が理解度が高い）
  テスト・データを提供する場合は、できるだけ早めに提供すること。

特に、オフショアでの開発には、以下の点をプロジェクト開始当初から考慮しておく必要がある。

- ・オフショアに準備できる開発環境・テスト環境の制約

例: 日本語/中国語の文字の使用可能性、大型機の準備の可能性、ファイルの共有可能性。

・オフショアで設計・実装しにくい非機能要件部分に関する制約

例: 統合認証機能が必要になるテスト、準備可能なテスト機のパフォーマンスに依存するテスト、実環境でないことによる制約。

・オフショアで準備しにくいテスト・データの制約

例: セキュリティに基づく顧客データの提供。日本語/中国語の文字に依存するテスト・データ。

また、オフショア開発では、コミュニケーションが重要であり、コミュニケーションプランを明確にしておく必要がある。例えば、誰と誰が、どのタイミングで(例えば週次)、どのような方法で行うかを決めておく。

【関連情報】

図 4.1 開発の流れ

表 4.1 各工程の説明

【ポイント 02 利用するUML図の確定】

各工程で使用するUML図、補助資料の種類を明確にする。

【目的】

開発の各工程においては、使用するUML図が予め決まっているわけではなく、内容に応じて取捨選択し利用する。プロジェクト開始時、工程ごとに、利用するUML図を決め、作業効率を向上させる。

【詳細・補足】

各工程で使用するUML図を明確にすることにより、各工程の作業開始時のロスを削減する。必須・任意に応じて作成（モデルの根拠の説明のため必要）などUML図のランク分けを行う。業務分析～詳細設計の各工程では、次に示すUML図(UML2．x)の利用を考えてみると良い。

〈業務分析〉

・ユースケース図	△（状況に応じて）
・アクティビティ図	○
・クラス図	△（状況に応じて）
・オブジェクト図	△（状況に応じて）

〈要求分析〉

・ユースケース図	○
・アクティビティ図	△（状況に応じて）
・シーケンス図	△（状況に応じて）
・相互作用概要図	△（状況に応じて）
・ステートマシン図	△（状況に応じて）

〈システム分析〉

・クラス図	○
・オブジェクト図	○
・シーケンス図	○
・コミュニケーション図	△（状況に応じて）
・ステートマシン図	△（状況に応じて）

〈アーキテクチャ設計、詳細設計〉
（上記の、前工程での作成図の見直しや詳細化に加

・コンポーネント図	△（状況に応じて）
・合成構造図	△（状況に応じて）
・クラス図	○
・コミュニケーション図	△（状況に応じて）
・ステートマシン図	△（状況に応じて）
・シーケンス図	○
・オブジェクト図	○
・パッケージ図	○
・配置図	△（状況に応じて）
・タイミング図	△（状況に応じて）

【関連情報】

表 4.2 工程ごとの主要な成果物

【ポイント 03 必ずUMLである必要はない】

UMLで、システム開発の全成果物を作成できない。UML として用意されている図以外も必要に応じて利用する。

【目的】

開発にあたって、UML として定義されている図だけを用いて設計書を記述しようと考える必要はない。開発対象の内容を、よりわかりやすく、より正しく伝えることが設計書の本来の目的であり、この点を見失わないことが目的である。

【詳細・補足】

UMLの図の特徴を捉えて、情報をまとめる上で最も適切な図を用いる。
UMLを適材適所で使用し、UMLで支援していな成果物は、UML以外の図を使用すべきである。たとえば、まとめる情報によっては、表形式にまとめたほうが、あきらかに分かりやすいものがある。これを、無理やりUMLのモデルで表現しても分かりにくいだけである。
システム開発のあらゆる面をUMLで作成することにこだわり、ドキュメント地獄に陥らないよう注意すべきである。

UML以外の図の利用には以下のようなものが考えられる。

- ・画面遷移図
- ・画面・帳票設計書
- ・データ・モデル図（E-R 図）
- ・データフロー図（DFD）
- ・ロバストネス図（UML の拡張）
- ・ネットワーク図
- ・一覧(表)-EXCEL 等

【関連情報】

表 4.2 工程ごとの主要な成果物

【ポイント 04 上流工程への参画】

オフショア開発を行なう場合、日本語力が高く設計経験が豊富なSEがいれば、国内で行なう上流工程にもオフショア側のSEが参画することが望ましい。

【目的】

詳細設計以降の工程をオフショア開発する場合、詳細設計工程以降を効率的に推進する。

【詳細・補足】

上流工程をすべて日本企業で行い、詳細設計工程以降をオフショア開発とした場合、十分な成果物を作成したとしても、オフショア先で、完全にシステムを理解してもらうことは困難である。

したがって、上流工程にオフショア先のメンバが参画することにより、開発するシステムの目的、内容を理解でき、詳細設計工程以降の設計ミスを削減できる。また、システムの全体像を理解しているメンバがオフショア側にいることから、一部の結合テストについてもオフショア先へ委託できる可能性も出てくる。参加人数は、システムの規模にもよるが、数人程度でよい。いずれにしても、上流工程を任せられるようなSEがオフショア側にいるかどうかを見極めたうえで実施すべきで、徐々にオフショア先の担当領域を広げるのがよい。

【関連情報】

ポイント 01 作業範囲/作業分担の明確化
ポイント 06 分析モデルで業務を理解
ポイント11 アーキテクチャ・モデル

【ポイント 05 非機能要件定義】

非機能要件をまとめておく。 非機能要件とは、機能をどのように実現すべきか（システムの性質）を決める要件である。

【目的】

非機能要件は、アーキテクチャ設計時の重要な要件であり、アーキテクチャ設計前に明確にしておく必要がある。 非機能要件は、アーキテクチャ設計時の入力になる。

【詳細・補足】

システム要件には、機能要件と非機能要件がある。

ユースケース図で機能的な要求を明確にすると共に、非機能要件を定義する。 非機能要件の例を次に示す。

- ・開発／実行環境（ハード、ソフト）
- ・性能要件
- ・信頼性要件
- ・コスト（短期、長期）
- ・セキュリティ
- ・保守性要件
- ・運用要件

ここでは、オフショアの開発領域に限らず、システムに要求される非機能要件を指している。オフショア開発の場合には、“「常識」が常識でない”ために発生する誤解・問題が多いので、非機能要件も含めて成果目標を明確に定義しておくことは重要である。オフショア側も、非機能要件が不明確であると考える場合には、発注側に早目に確認することが肝要である。

【関連情報】

ポイント１１ アーキテクチャ・モデル

【ポイント 06 分析モデルで業務を理解】

分析モデルを作成することで、業務内容を視覚的に理解できるようにする。

【目的】

システム分析にモデルを使用することにより、業務内容の大枠を視覚的に理解できる。要求に精通している日本側で作成する。モデルの記述内容は日本語を使用するが、あまり長文を使用しないことが好ましい。モデルを使用することで、オフショア側プロジェクト・メンバの日本語のレベルがあまり高くない場合でも、情報の伝達が確実である。

【詳細・補足】

分析モデルはクラス図を中心に作成し、業務の内容を表現する。分析モデルは、実現方法を記述しない。矛盾があってはならないが、必ずしも詳細な内容を網羅する必要はない。共通理解ができるような情報をモデリングする。この時点で曖昧な点をなくすことは、それ以降の工程での手戻り削減につながる。逆にオフショア側で分析モデルの内容が理解できない場合、疑問がある場合には、早目に質問をすること。

【関連情報】

付録. 成果物サンプル （１）各工程の成果物 ③工程：システム分析

【ポイント 07 用語辞書を作成する】

用語辞書を作成する。用語辞書は、業務用語と共通用語を分けた方が効果的である。
用語辞書は、日本語だけでなく、オフショア先の言語(中国の場合は、中国語又は英語)を入れた方がよい。
特に、業務用語は、オフショア先の言語を入れた方が、オフショア側プロジェクト・チームに正しく伝わる。

【目的】

業務内容の理解度向上と意識の統一を目的とする。
オフショア側プロジェクト・チームに対しては、ネーミング・ルールを明確にしておくことで、下記をねらう。

- ・日本側プロジェクト・チームから受け渡した設計書の理解度向上
- ・オフショア側プロジェクト・チームでの仕様書作成の向上
- ・受け入れ時のチェックに利用

【詳細・補足】

用語と内容説明を記述し、業務の理解度をあげるために使用する。
用語辞書のバージョン変更の連絡は明確にする。
なるべく日本側プロジェクト・チーム、オフショア側プロジェクト・チームでの辞書のバージョンをリアルタイムに近い形で統一すること。

【関連情報】

ポイント 08 命名規約を作成する
UMTP モデリング用語集
http://www.umtp-japan.org/modules/data4/index.php?id=2

【ポイント 08 命名規約を作成する】

命名規約(ネーミング・ルール)を作成する。少なくともシステム分析工程で、命名規約(ネーミング・ルール)の標準化を行う。

【目的】

開発で使用する各種名称の命名規則を標準化し、要求分析/システム分析工程の成果物の仕様理解と、以降の工程の成果物の効率的な作成が目的である。

これは、オフショア側プロジェクト・メンバにとって不慣れな言語で記述された設計書の読解と、以降の設計書の作成時に有用である。

さらに、オフショア側プロジェクト・メンバに対して、命名規約(ネーミング・ルール)を明確にしておくことで、下記をねらう。

・日本側プロジェクト・チームから受け渡した設計書の理解度向上

・オフショア側プロジェクト・チームでの仕様書作成の品質向上

・受け入れ時のチェックに利用

日本側プロジェクト・チームにとっても、日本語の誤用や説明不足がある設計書を理解し、レビューする上での助けになる。

【詳細・補足】

英語表記なのか日本語表記なのか、何桁なのか、各桁はどのように使い分けられるのかなど、事前に統一しておくとよい。

命名規約(ネーミング・ルール)を決定しておく項目の候補としては

・サブシステム名/コンポーネント名、

・パッケージ名

・ユースケース名

・クラス名

・属性名/プロパティ名

・操作名/メソッド名

・イベント名

・メッセージ名

・状態名

・画面 ID、画面遷移 ID

など

【関連情報】

ポイント 07 用語辞書を作成する

【ポイント 09 モデルの作成規約を作成する】

ソースコードのコーディング規約などと同様に、モデル作成における規約（ルール）を作成する。

【目的】

モデル作成時、作成の規約（ルール）を作成しておくことで、プロジェクト・メンバの作成したモデルが同一基準で作成される。そのため保守性が向上する。

【詳細・補足】

UMLの要素（クラス、パッケージ、メッセージなど）を独自のルールを設けることにより、分類し、そのUML要素の意味をより明確にすることができる。

たとえばクラスを、BCE（Boundary、Control、Entity）に分けて、システム分析、アーキテクチャ設計、詳細設計を行なうことは、一般的に広く行われている。Boundary は画面や他システムを仲介するクラス、Control は流れを制御するクラス、Entity は現実にあるクラスを示している。

それ以外にも、「データの受け渡しに使用するクラス」、「データベースとのやり取りをするクラス」、「時間制約があるメッセージ」など、開発プロジェクトや企業単位の独自のルールを設けることで、プロジェクト・メンバ間の意思疎通を図りやすくする。

また、モデルに対する変更権限についても明確に定めておくと良い。例えば、オフショア側では private な操作は追加しても構わないが、public な操作を追加する時には日本側の承認が必要など。

必要に応じてこれらモデルの作成規約を設けるときに、UMLの拡張機能を利用することも考えられる。UMLでは、UMLを拡張する仕組みとしてUMLプロファイルを用意している。UMLプロファイルによるUMLの拡張は具体的には、ステレオタイプ、制約、メタ属性を用いる。

【関連情報】

ポイント 07 用語辞書を作成する
ポイント 08 命名規約を作成する

【ポイント 10 共通機能の明確化】

UML モデリング工程での共通機能化の検討を行う。
設計／実装時のオフショア側プロジェクト・チーム独自の共通化を事前に日本側プロジェクト・チームがアーキテクチャとして確立しておく。

【目的】

品質向上、開発効率の向上が目的である。
具体的には、オフショア側プロジェクト・メンバには、次の傾向があり、それを防止する必要がある。

・提供されたサンプルや最初に作成した開発物を繰り返しコピー＆ペーストし、 それを書き直して成果とする。
・コピー＆ペーストにより設計や実装がなされる結果、共通コンポーネント・共通機能・汎用モジュールとして考えられる類似の機能がいくつも作成されることになる。

【詳細・補足】

オフショア側プロジェクト・チームは内部での開発効率を向上させるために、サンプルを元に独自に共通化し、手順が確立後にコピーして作成していく場合が多い。
事前にアーキテクチャおよびガイドラインを日本側が提供し、日本側で早い段階で分析レビューを行う。
コンポーネントが適切に分割され、共通機能化が図られていることを確認するとよい。
ユーティリティと見なせる同じロジックが、複数のコンポーネントに散在することのないようにする。これには、

・作業工程の計画時に、共通機能化検討の作業項目を含めさせる
・レビュー内容に、共通機能化が行われているかのチェック項目を含めさせる

が考えられる。

【関連情報】

なし

【ポイント 11 アーキテクチャ・モデル】

「アーキテクチャ・モデル」とは、ハードウェアやソフトウェアの基本的な構造の設計のこと。アーキテクチャ・モデルをしっかりと作成し、内容を共有すること。

【目的】

ハードウェアとソフトウェア・アーキテクチャが顧客の業務的な要求を満足しているかどうか判断するためには、対象となる業務内容を十分理解しておく必要がある。このため、アーキテクチャ・モデルは、日本側で作成する。また、このアーキテクチャの変更は、システムに対して大きな設計変更・仕様変更を生じることになるため、長期的な視点に立ち、将来的なシステム拡張や変更を考慮してアーキテクチャを設計しておく必要がある。

アーキテクチャの設計をオフショア側に任せた場合には、将来のメンテナンスやシステム拡張を行う際に、発注側（日本側）のアーキテクチャへの理解が弱く、適切な変更・拡張の設計を行えないことにもつながる。（この点から、メンテナンスも含め、長期的にオフショアのある特定の企業に任せ続けるという選択肢は考えられる。この場合でも、内容について十分レビューし、日本側も把握しておく必要がある。）

【詳細・補足】

開発対象となるシステムのハードウェア、ソフトウェア・アーキテクチャは、要求に精通した日本側で作成すべきである。この時に考慮すべき事項としては、

・そのアーキテクチャは、機能面／非機能面で妥当であるか
・求められるパフォーマンスやキャパシティを、どう実現しようとしているか
・本番稼働環境／テスト実施環境からくる制約事項は、どう解決しようとしているか
・障害対策を含む運用上の要件を、どう実現しようとしているか
・セキュリティに関する要件を、どう実現しようとしているか
・開発に参加するメンバのスキルに、どう対応しているか

などが挙げられる。

そして、アーキテクチャ・モデルを日本側で検討、設計してから、オフショア側プロジェクト・メンバに成果としてそれを伝える。必要に応じて、「ハードウェア、ソフトウェア・アーキテクチャに従って開発を行う手順」などを記したガイドを作成するのもよい。

【関連情報】

ポイント 12 アーキテクチャ説明成果物の作成
付録. 成果物サンプル （1）各工程の成果物 ④工程：アーキテクチャ設計

【ポイント 12 アーキテクチャ説明成果物の作成】

システムのソフトウェア・アーキテクチャを説明する成果物を作成する。

【目的】

オフショア開発において、日本側での要求分析～アーキテクチャ設計からオフショア側での詳細設計へと開発の主体が移る際、開発対象システムのソフトウェア・アーキテクチャについて、日本とオフショア双方のプロジェクト・メンバが正しく同じように理解することが必要である。 開発対象システム全体がどのような考え方や構造に基づいて設計されているか、日本側プロジェクト・メンバからオフショア側プロジェクト・メンバへ効果的に伝える目的がある。

【詳細・補足】

プロセスへのインプットとして補足仕様書を用い、ソフトウェア・アーキテクチャを検討して、検討結果となるアーキテクチャを説明する成果物をアウトプットとして作成する。システム分析～アーキテクチャ設計レベルでのクラス図とシーケンス図、および、それらを補足して説明する資料群からなるのが、アーキテクチャ説明成果物である。

開発対象システムのためのハードウェア/ソフトウェア・アーキテクチャ、なぜそのように設計されたかの理由や判断ポイント、構造の狙いや目的を、主に機能要求面から各図に記述していく。 必要に応じて、コンポーネント図、パッケージ図、配置図も用いるとよい。 モデルを用いて図示することにより、日本語での説明記述であっても、より正確にオフショア側プロジェクト・メンバに伝えることができる。

すなわち、アーキテクチャ・モデル作成プロセスの関連成果物として、

・インプットとなる成果物: 非機能要件定義 （補足仕様書）

・アウトプットとなる成果物: アーキテクチャ説明

詳細設計ガイドライン

となる。

【関連情報】

ポイント 11 アーキテクチャ・モデル

付録. 成果物サンプル （2）アーキテクチャ説明書

【ポイント 13 パターンの活用】

アーキテクチャ設計において、あるいは、詳細設計において、アーキテクチャやデザイン上のよく知られた各種パターンを活用することが望ましい。

【目的】

アーキテクチャ設計におけるアーキテクチャの意図を伝える場合、あるいは、詳細設計におけるコンポーネントやクラスの責務や意図を伝える場合、プロジェクト・メンバの間によく知られるパターンを利用することにより、例え説明がなくても、その部分の目的・責務についての情報量が増えたことと同等の効果が得られ、双方の共通理解を促進する。

【詳細・補足】

ソフトウェア開発におけるデザインパターンは、UML との組み合わせで説明されることが多い。また、設計のノウハウがそれら各パターンに集約され、開発者間に浸透することで、パターンを再利用・適用することによる品質確保ばかりでなく、設計やレビュー時のコミュニケーションにおいても、パターンによる共通理解を得られる機会が少なくない。 アーキテクチャ設計におけるアーキテクチャレベルでのパターンとして、例えば MVC(Model, View, Control) などを適用することにより、詳細設計担当のプロジェクト・メンバにその意図を誤解なく伝えやすくなる。 一方、詳細設計におけるデザインレベルでのパターンであれば、例えば GoF（Gang of Four） デザインパターンなどを適用することにより、レビュー時にも設計各部の責務などが想像しやすく、チェックすべき箇所が絞られることで点検を要領よく進められることにつながる。 このようにプロジェクト・メンバ間の共通理解を促進するためのパターン利用は有効である。

オフショア開発にパターンの習得と利用は必須ではないが、以上の効果のため、開発対象やプロジェクト・メンバのスキルによっては、コミュニケーションをより円滑にできる手法のひとつとして、習得するとよい候補のひとつと考えられる。

【関連情報】

ポイント 06 分析モデルで業務を理解
ポイント 11 アーキテクチャ・モデル
ポイント 12 アーキテクチャ説明成果物の作成

【ポイント 14 仕様書の記述レベル、書式の指定】

オフショア側の成果物となる各仕様書について、UML の記述レベルや書式を指定する。

【目的】

UML に限らず、オフショア開発においては成果物サンプルを求められることが多い。オフショア側プロジェクト・メンバによる記述に関し、日本側からの指示やガイドの不足は、個々のオフショア側プロジェクト・メンバ間に記述詳細のばらつきや、書式のばらつきの問題を生じることになる。オフショア側では、プロジェクト・メンバ全員で記述レベルや書式を統一しようとする動きがされにくい背景もここにある。

オフショア側成果物をお客様に提出するのか、日本側プロジェクト・メンバが見るにとどまるのかは、このポイントを適用する上での考慮点になる。

【詳細・補足】

成果物に含まれる UML については、実際の書式に従って、日本側で求める記述詳細やモデル粒度で、オフショア側プロジェクト・メンバの参考にできるサンプルを作成し、ガイドの一部として提供する。 この時、その UML で何を表して欲しいのか、何が示されていればよいか、モデルの目的を指示できるのが望ましい。これにより、UML の詳細さ／正確さ／粒度が揃いやすくなる。

例外処理／エラー処理についても、記述して欲しいレベルで適切にサンプルに含めておく必要がある（当然追記されるであろうと期待し、サンプルに含めないでおくと、最終的な成果物にも含められてこない結果となりがちである）。

その他に、成果物の記述時の注意点として「あいまいな表現ではなく具体的な表現で記述すること」、「使用する用語について統一して使用すること」も合わせて指示・ガイドできるとよい。

【関連情報】

なし

【ポイント 15 詳細設計ガイドライン作成】

アーキテクチャにそって、詳細設計を進めるガイドラインを作成する。

【目的】

アーキテクチャは、保守性、効率性、性能などを考慮して作成してある。
詳細設計では、そのアーキテクチャを踏襲することで、システム全体に対して、保守性、効率性、性能を維持することができる。
詳細設計の作業において、アーキテクチャから逸脱しないように、ガイドラインを作成することが重要である。

【詳細・補足】

アーキテクチャ説明においては、クラス図、シーケンス図などのモデルおよび、自然言語において、そのシステムのアーキテクチャの説明がなされている。
詳細設計では、そのアーキテクチャと同様のことを、システム（モデル）の他の部分(機能)に適用する必要がある。

アーキテクチャ説明に記述されているのは、アーキテクチャの考え方だけである。
ガイドラインでは、どのようにモデリング・ツールを使用し、クラス図、シーケンス図をどこに作成し、クラス、オブジェクトをどこに配置していくか、などを具体的に説明する。
アーキテクチャ説明を全体の方針とすると、ガイドラインは具体的にブレイクダウンされたものになる。

【関連情報】

付録. 成果物サンプル (3)詳細設計ガイドライン

【ポイント 16 仕様未決定部分は明確に】

成果物における仕様未決部分は、オフショア側に明確に提示する。オフショア側も仕様が不明確な場合は、勝手に判断せず早めに確認をする。

【目的】

開発期間不足などの関係で、要求分析～システム分析～アーキテクチャ設計工程の、日本側作業での仕様未確定部分が多いまま、オフショア側に開発依頼をする場合も少なくない。その仕様未決部分が明確に示されていない成果物は、内容の整合性に欠け、オフショア側プロジェクト・メンバに多くの疑問・質問を引き起こすことになる。

【詳細・補足】

オフショア側プロジェクト・メンバに渡す成果物には、仕様未決（未決・仕様変更の可能性大）部分を明記して渡すことで、仕様吸収漏れを抑えることができる。

仕様未決（未決・仕様変更の可能性大）部分については、成果物 UML 内に、

・どの範囲が仕様未決であるのか、影響が及ぶのはどの範囲か

・現仕様にどのような変更が予想されるのか（今後さらに詳細化される方向性なのか、内容として異なる内容に置き換わる方向性なのか）

を、備考や注釈等として明記する。

日本側プロジェクト・メンバがこれら各項目を管理することで、仕様決定時・仕様変更発生時のオフショア側開発分への組み込み漏れ防止、仕様変更工数の低減を図ることができる。オフショア側との進捗状況確認会議においても、それら仕様未決事項の一覧を元に、反映状況更新の確認、仕様決定事項の確認ができるとよい。

【関連情報】

なし

【ポイント 17 オフショア側での成果物のレビュー実施】

オフショア側プロジェクト・メンバによるレビューを確実に実施する。

【目的】

成果物の内容レビューを日本側でも実施することを伝えると、オフショア側レビューが全く実施されない状態での成果物が届くことが起こり得る。「日本側でレビューするのだから、オフショアでもレビューを行うと二重レビューになってしまい、作業の無駄になる」との考え方がその背景にある。

【詳細・補足】

依頼する成果物に関し、オフショア側でレビューする対象の一覧、レビューでのチェック対象となるポイント、レビューでの合格基準について協議し、合意しておくとよい。 その際、オフショア側でのレビューの目的、日本側でのレビューの目的の違いも明確に理解してもらい、オフショア側でのレビューが確実に実施されるようにしておく。オフショア側は、必ずレビュー報告書を作成し、日本側は必要に応じてレビュー報告書をチェックする。

基本的なレビュー・ポイントとして;
・開発予定機能一覧にある機能がすべて網羅されているか
・前工程の設計内容に対応する仕様がすべて記述されているか
・指定の記述レベル、書式に従っているか
・用語は正しく統一された語が用いられているか
・開発予定機能の静的な構造が正しく伝わり、設計、記述されているか
・開発予定機能の動的な振舞い・挙動が正しく伝わり、設計、記述されているか
・開発予定機能の物理的な配置が正しく伝わり、設計、記述されているか
・クラスの責務に混乱はないか
・図間の整合性が確保されているか
・仕様変更が反映されているか
などが挙げられる。

【関連情報】

ポイント 18 日本側はチェックを繰り返し行なう

【ポイント 18 日本側はチェックを繰り返し行なう】

日本側では、開発中はオフショア作業開始直後からチェックを行う。

※なお、本ガイドラインでは成果物の厳密な確認・承認を行なう「レビュー」に対して、サンプリング的な確認や成果物だけでなく意識レベル・理解度も含めての確認という意味で「チェック」という用語を用いる。

【目的】

オフショア開発におけるオフショア側プロジェクト・メンバにとっては、やはり慣れない言語でのコミュニケーションとなることから、

・（日本側からは）オフショア側プロジェクト・メンバの理解の度合いがわかりにくい

・（オフショア側では）誤った理解で思い込んでしまう。関連する内容をお互いで確認することなく進めてしまい、誤解が解消されないことへの対処が目的である。

オフショア側プロジェクトに任せたままにしていると、成果物が出来ていなかったり、品質が著しく悪かったりすることがある。それらのチェックを確実に行なっておく必要がある。日本側で適切にチェックすることで、品質向上・開発工数低減につながる。日本側にチェックのための工数と人材を確保しておくことが必須。

【詳細・補足】

作業開始直後から、ウォークスルーなどにより日本側からもチェックを頻繁に実施し、

a）成果物の内容記述は適切か、

b）成果物の形式は標準に従っているか、

c）要件や設計内容の理解に誤解がないか

をチェックするとよい。 オフショア側でのレビュー内容を、日本側からも、途中段階で早期から繰り返し行うべきとも言える。日本側は、すべての成果物をチェックするのではなく、適切なタイミングで適切な成果物をチェックするようにする。

さらに、開発期間中を通して、お互いの理解や開発成果内容を、オフショア側プロジェクト・メンバ間で確かめるよう習慣づけられれば、さらに好ましい。

【関連情報】

ポイント 17 オフショア側での成果物のレビュー実施

【ポイント 19 Validation と Verification】

日本側の目的達成のための Validation（妥当性確認）と、オフショア側の実装内容確認のための Verification（検証）のふたつの観点からレビューを行うことが望ましい。

【目的】

オフショア開発においては明確に作業分担されることから、日本側では業務分析～システム分析の、主にお客様の要求実現に関心が集まり、オフショア側では詳細設計～単体テストの、主に日本側からの仕様実現に関心が集中する。 この視点の違いは、テスト範囲の漏れや要求機能の実現漏れを生みだす。日本側からの仕様に合わせた実装結果の検証だけでなく、実装がお客様の要求を満たしているかどうかの妥当性確認も行うことにより、これらの漏れを防止することにつながる。

【詳細・補足】

オフショア側は単体テストや結合テストにおいて、詳細設計あるいはアーキテクチャ設計とプログラムの実行結果とをつき合わせ、発注された仕様通りに正しく作られているかどうかの検証（Verification）に終始することが多い。 一方、日本側は、お客様のビジネス要求を満たす正しい実装が作られているかどうかも含めて確認（Validation）したいことが多い。

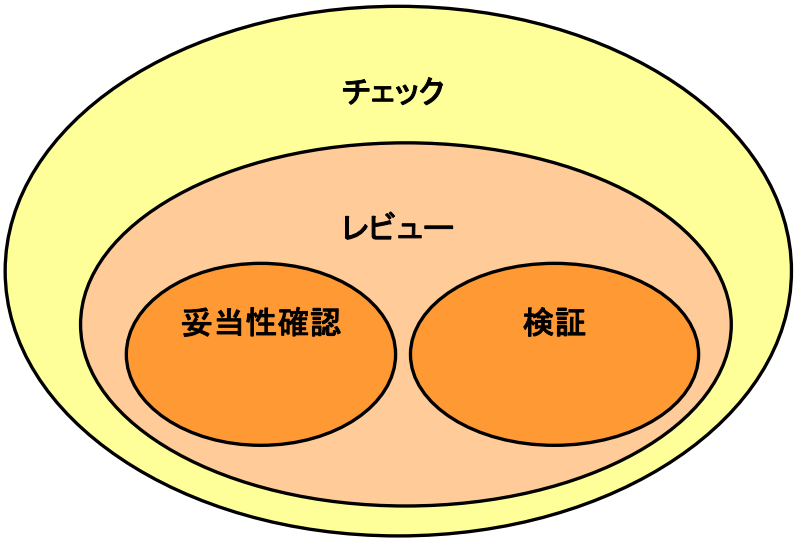
この視点の違いに注意し、オフショア側で行われがちな Verification 中心のレビューに加え、

- オフショア側にどのようなテストを実施して欲しいか、Validation 視点のテスト仕様として具体的に早期に明確に伝える
- オフショア側からの成果物受取り後、日本側で Validation 視点のレビューやテストを行えるように並行して準備する

という対応を行うことが望ましい。

例えばトレーサビリティ・マトリックスは、要求から実装までを結び付けることで、各要求はどの実装により満たされているかを確認することができる手法のひとつとして有効であるため、習得されているならば有効である。

なお、本ガイドラインにおいて、前出の「レビュー」「チェック」と本ポイントの「妥当性確認」「検証」とを、次の図のように位置づけて使用している。
成果物の承認を目的とするレビューは、妥当性確認と検証の2つの視点からレビューが望ましい、定義された成果物レビュー以外にも、離れた地点での開発において誤解なく円滑に進めるため各種の幅広いチェックが望ましい、と言い換えられる。



【関連情報】
ポイント 01　作業範囲／作業分担の明確化
ポイント 17　オフショア側での成果物のレビュー実施
ポイント 18　日本側はチェックを繰り返し行なう

【ポイント 20 モデルで詳細設計のレビュー】

詳細設計では、クラス図とシーケンス図、オブジェクト図、パッケージ図でレビューを行なう。

【目的】

実装を行なう前に、モデルによって方向性を確認する。ソースコードで、流れを追う必要はなく、モデル（シーケンス図など）で誰もが見やすい形でレビューを確実に行なうことが可能になる。
日本側でモデルにより詳細設計のレビューを行なうことで、実装の品質を向上させる。

【詳細・補足】

レビューは、通常の開発より、オフショア開発のほうがより確実に行なう必要がある。しかし、細かなチェック項目を作成しすぎて、日本側プロジェクト・メンバのレビュー工数が膨大になってしまいがちである。ここで、モデルの利用が有効である。

日本側プロジェクト・チームが作成した、アーキテクチャ説明、アーキテクチャのガイドラインに沿って、オフショア側プロジェクト・チームは、詳細設計を行い、成果物としてシーケンス図及びクラス図、オブジェクト図、パッケージ図を作成する。

日本側プロジェクト・チームは、オフショア側プロジェクト・チームが作成した、シーケンス図及びクラス図、オブジェクト図、パッケージ図のレビューを行なう。
レビューでは、アーキテクチャ説明を参照し、オフショア側プロジェクト・チームが作成したシーケンス図及びクラス図、オブジェクト図、パッケージ図がアーキテクチャから逸脱していないかのチェックを行なう。

【関連情報】

ポイント 09 モデルの作成規約の作成
ポイント 11 アーキテクチャ説明成果物の作成
ポイント 15 詳細設計ガイドラインの作成
ポイント 21 UML 図間の整合性

【ポイント 21 UML 図間の整合性】

オフショア側プロジェクト・メンバが設計した UML 図の間の整合性に注意する。

【目的】

UML に限らず、オフショア開発においては、成果物内容相互に整合性の欠落が見られることが多い。

日本側から成果物を渡す際の整合性の欠落は、オフショア側プロジェクト・メンバに多数の疑問・質問を引き起こす。日本側で成果物を受け取る際の整合性の欠落は、日本側プロジェクト・メンバが内容を理解するのに困難な状況となり、双方でレビューできない状況を引き起こす。読者に暗黙の了解を期待した成果物ではなく、必要な情報をすべて記載することも重要な点である。

【詳細・補足】

動的な振舞いの面で、

・クラス図/シーケンス図/コミュニケーション図の間

・シーケンス図/ステートマシン図/アクティビティ図の間

静的／物理的な構造の面で、

・クラス図/パッケージ図/オブジェクト図/コンポーネント図/配置図の間

など、関連する複数の図の間でお互いに対応する箇所があるはずの図の整合性は、日本側の詳細設計レビューにおいてチェックすべき対象となる。

また、それぞれの図において、同じものを表すのに異なる用語が使われていることがないかも、整合性の面からの点検すべき対象となる。

・あるシーケンスで動作すると記されたオブジェクトが、クラス図から読み取れない

・ある状態遷移を引き起こすトリガが、シーケンス図やアクティビティ図から読み取れない

・あるノードに配置されるはずのコンポーネントが、クラス図やコンポーネント図から読み取れない

といった整合性の欠落は、実装の誤りを生む。

【関連情報】

なし

【ポイント 22 実装はツールのコード生成機能を使用する】

利用できるモデリング・ツールがあれば、実装時はツールのコード生成機能を使用して、ソースコードのテンプレートを生成し、それをもとにコーディングを行なう。必ずしもツールを使用する必要はないが、ツールを利用した方が効率的に信頼性の高いコードを記述できる。

【目的】

詳細設計において、オフショア側プロジェクト・チームが作成したクラス図及びシーケンス図などの成果物を日本側プロジェクト・チームがレビューする。
モデルにおいて問題がないことを確認されたレビュー後の(信頼性のある)モデルから、モデリング・ツールのソースコード自動生成機能を利用することで、ソースコードの信頼性も高める。

【詳細・補足】

クラスについては、モデリング・ツールのコード生成機能を利用することにより、ソースコードのスケルトンを生成することが可能になる。
これにより、クラス名、属性名、操作名などは、ソースコードに出力されるため、ソースコード内の内容を、モデル(クラス図)で確認することが可能になる。

処理の流れ(主にシーケンス図)については、一般的なモデリング・ツールではソースコードの生成を行なわないため、手作業でのチェック作業が必要になる(モデリング・ツールによっては、ソースコードの実行したものをリバースし、シーケンス図を作成できるものもある)。
モデリング・ツールのコード生成機能を利用することで、見やすくわかりやすいモデルのチェックにより、ソースコードの信頼性も高めることができる。

【関連情報】

なし

【補足　日本におけるシステム開発の特徴】

発注側（日本側）のシステム開発における特徴を明確にし、受注側（オフショア側）とのギャップを解消する。

【目的】

日本のシステム開発は、以下のような傾向がある。

・仕様変更を前提（当たり前）として、プロジェクトが進められる

・テストの網羅度や管理方法について厳密さを求められる

日本側のシステム開発の特徴を明確にすることにより、発注側／受注側双方の認識を合わせプロジェクトのトラブルの発生を軽減する。

【設計編】

仕様変更はプロジェクトの品質悪化に最も重大な要素であることを顧客、開発側（日本側とオフショア側）で共有することが重要である。

【詳細・補足】

仕様変更が発生する要因について以下に示す。

① 顧客の業界では一般的（当たり前）な決まりごと、慣習、特殊な条件における動作などについては記述されていないことがあり、テスト完了後の顧客確認時に発覚する。

② 日本の顧客は、操作方法や画面のレイアウトについて細部の要望を出す傾向がある。

③ 日本の顧客は、一度確定（合意）した仕様でも変更要望を出すことがある。

④ 仕様が確定していない段階でもオフショア側担当作業（詳細設計）の開始を依頼することがある。

【製造・テスト編】

【詳細・補足】

テストに関する要求事項について以下に示す。

① 全てのパスについてテストを要求される

条件のバリエーションや組み合わせ、全ての画面遷移、全てのコード（網羅率　100%）など

② エビデンス（テスト結果）について基本的には要求される

・処理パターンが明確になるようにエビデンスを要求される

・開発途中で品質に問題がある場合は、エビデンスを要求される

③ テスト項目の基準値が設定される

・ステップ数、画面項目数、DBテーブル数でテスト項目数が設定される

・正常、異常、境界の各ケースの比率が設定される

④ テスト中のバグ件数が設定される

・バグ件数が正常系、異常系それぞれの標準に満たない場合、再テストを要求される

・バグ件数が多い場合、見直しを要求される

⑤ テスト中の多くの管理資料を要求される

・バグ票、問題課題表、バグ発生曲線、テスト工程品質管理図など

付録. 成果物サンプル

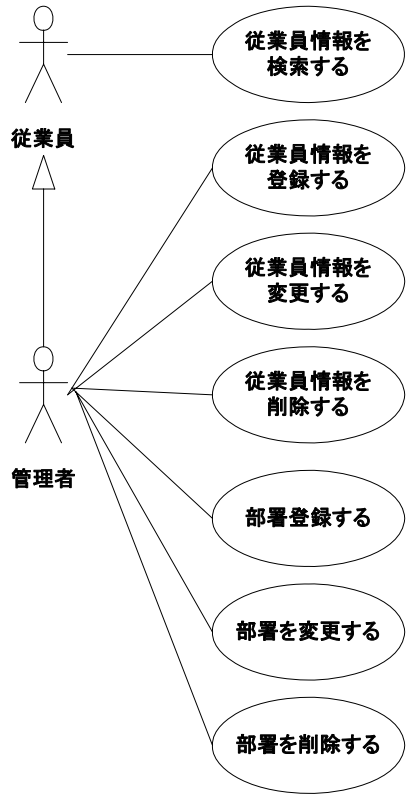
（１）各工程の成果物（一部分だけ）

①要求定義

A社では、従業員情報の検索システムの開発を検討しています。
従業員の部署、年齢、住所などのキーワードを入力して検索すると、それに適合する従業員情報を表示します。従業員ならば誰でもこの機能を使用することができます。
この会社の従業員は、技術職、営業職、事務職に分かれます。この会社には幾つかの部署があり、従業員はいずれかの部署に所属します。従業員の所属する部署は1つとは限らず、従業員は、複数の部署を兼務する場合もあります。
従業員情報は、あらかじめ管理者が登録しておきます。また、変更や削除をすることもできます。
従業員の所属する部署も管理者があらかじめ登録しておく必要があり、変更、削除ができます。

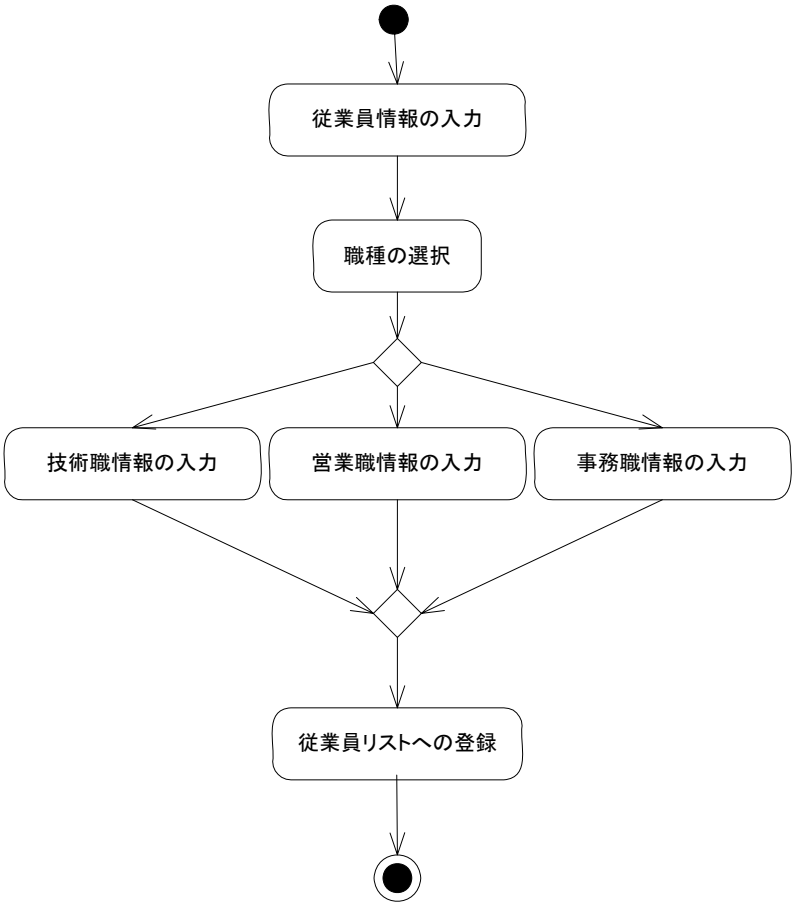
②工程：要求分析（日本側⇒オフショア側）

●ユースケース図

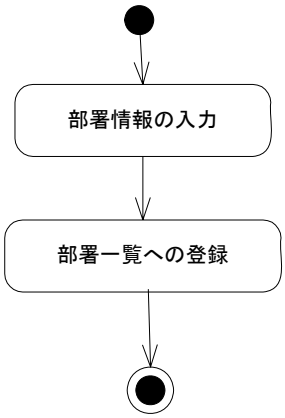


●ユースケースのアクテビティ図(イベントフロー)

ユースケース「従業員情報を登録する」のアクテビティ図(イベントフロー)



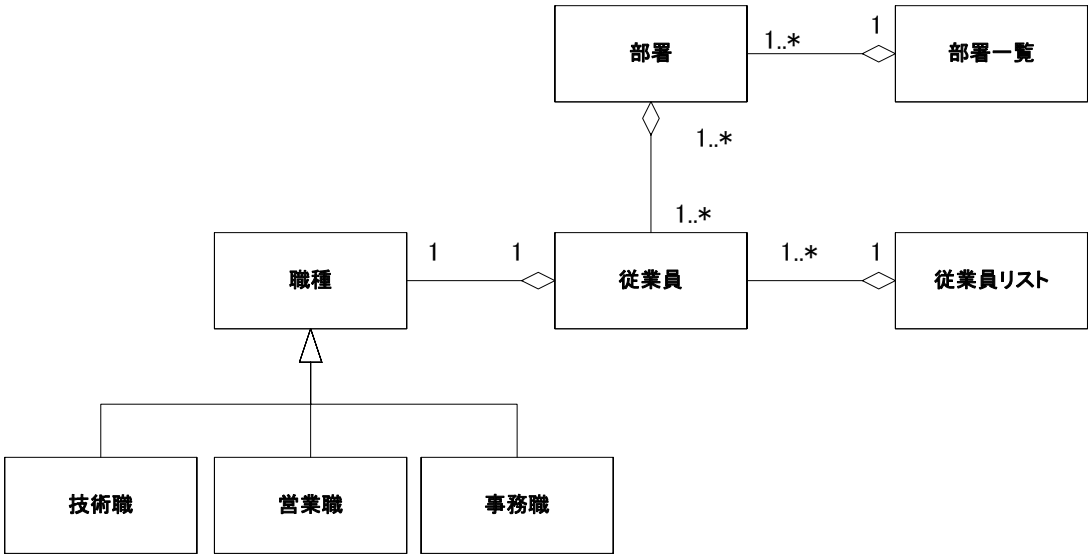
ユースケース「部署登録する」のアクテビティ図(イベントフロー)



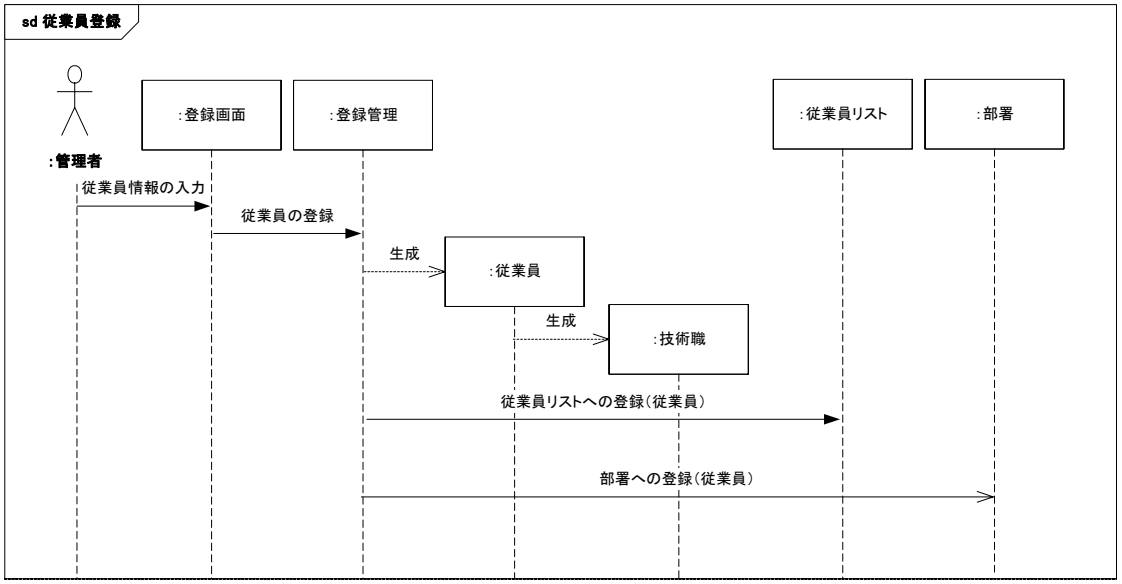
※その他、全てのユースケースに対して、このアクテビティ図(イベントフロー)を記述する。

③工程：システム分析（日本側⇒オフショア側）

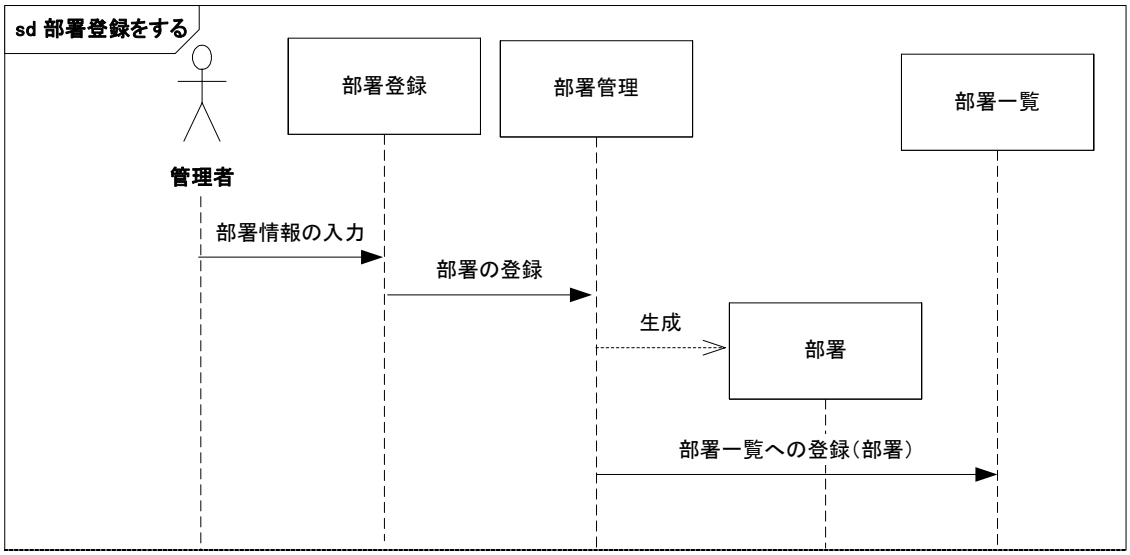
●クラス図



●「従業員情報を登録する」のシーケンス図

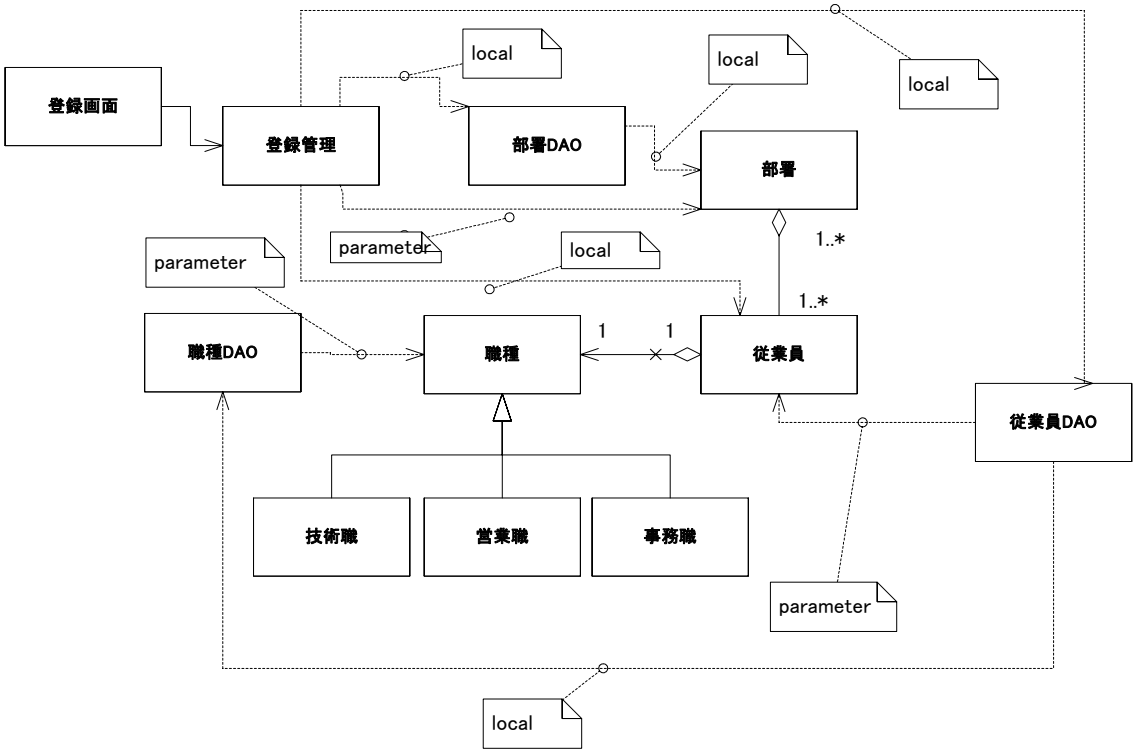


●「部署登録する」のシーケンス図

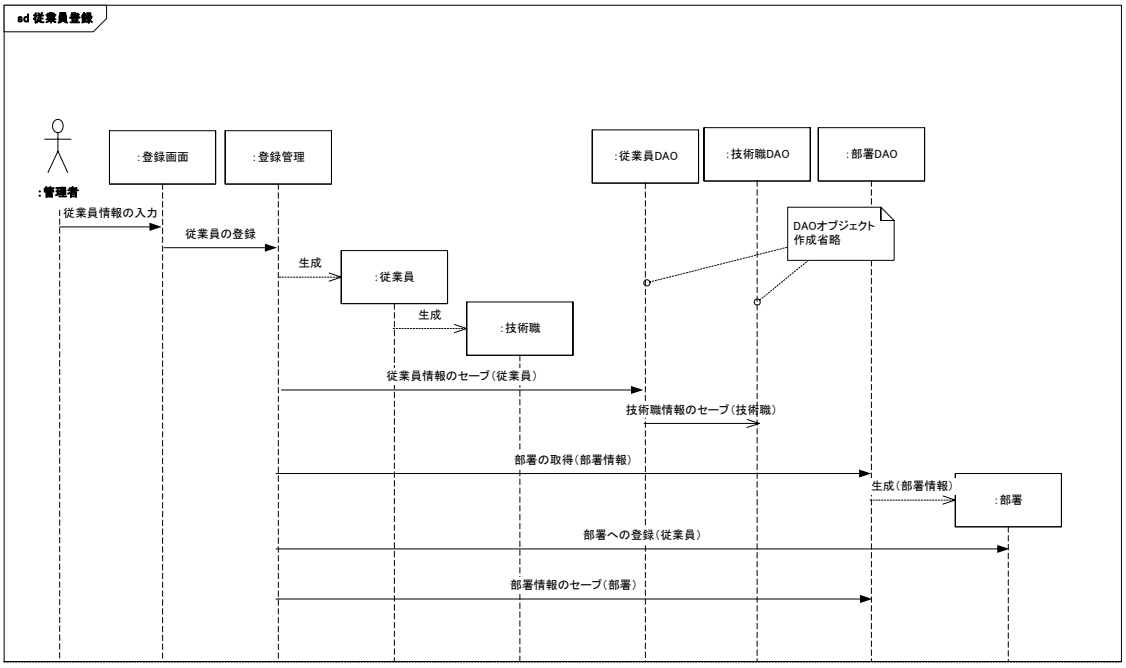


④工程：アーキテクチャ設計（日本側⇒オフショア側）

●クラス図

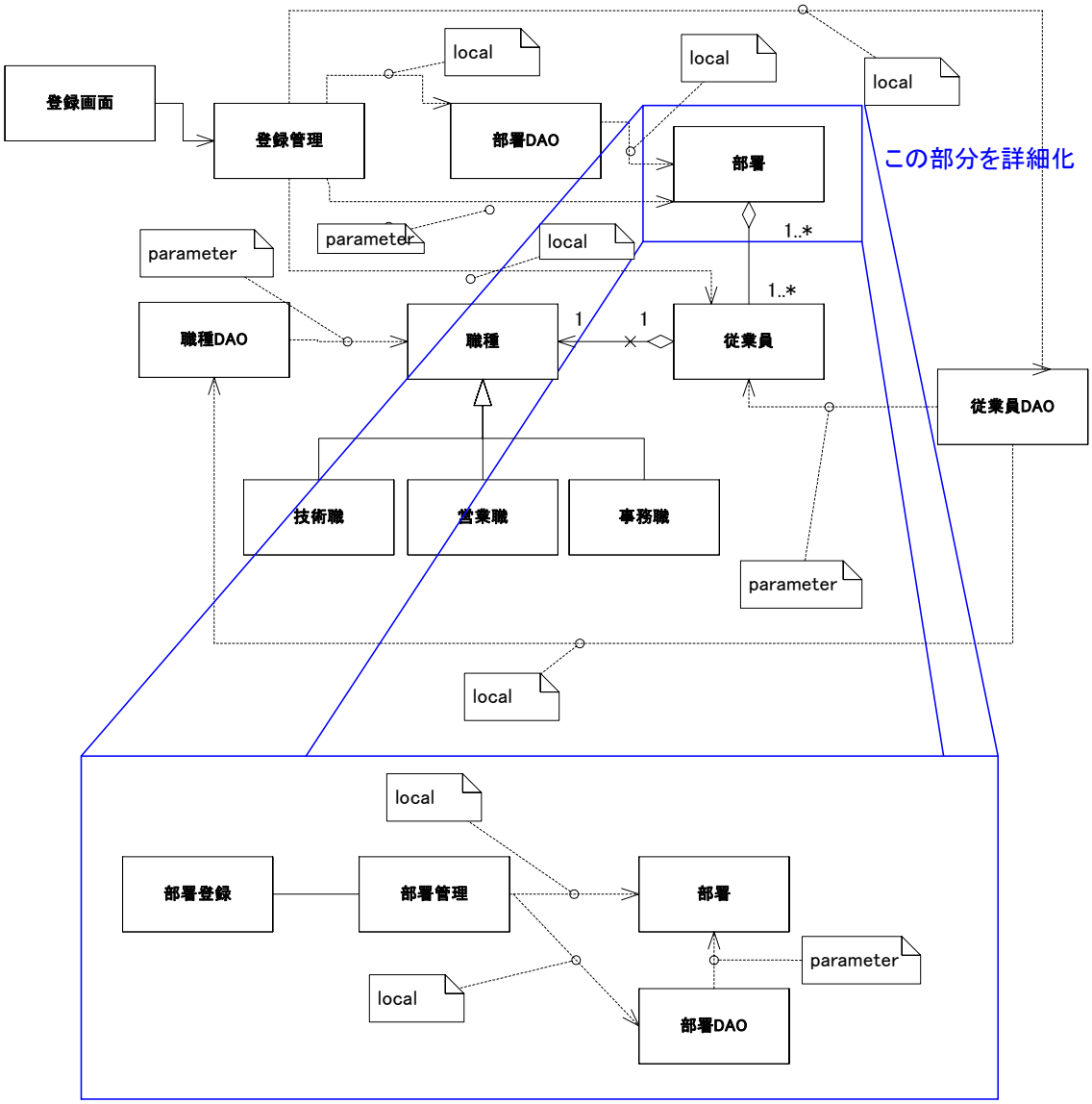


●「従業員情報登録」シーケンス図

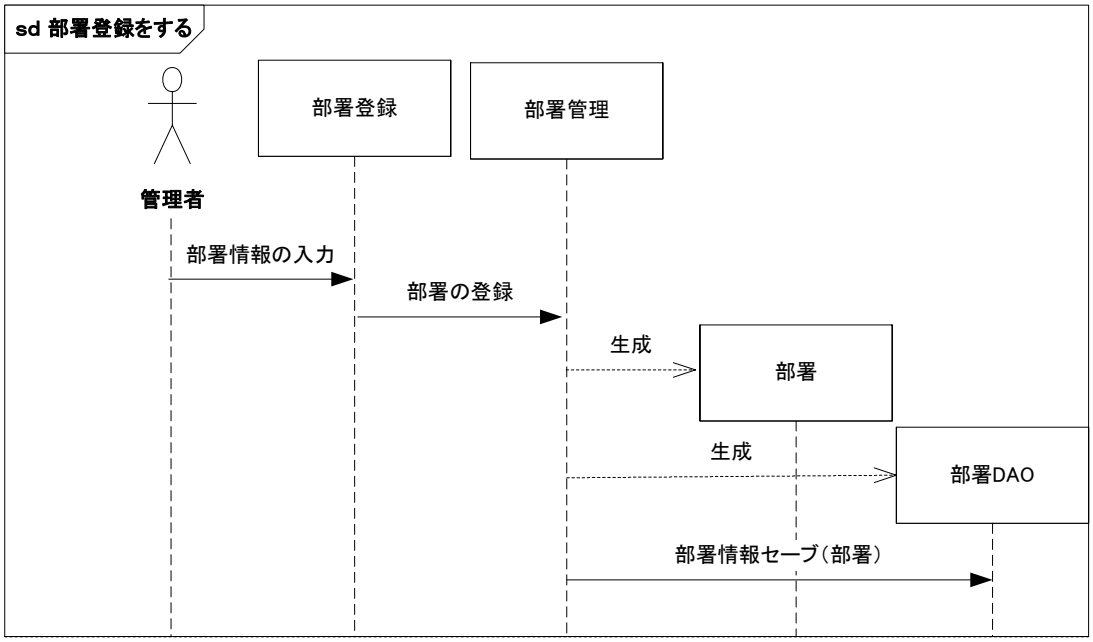


⑤工程：詳細設計（オフショア側⇒日本側）

●クラス図



●シーケンス図


sd 部署登録をする
部署登録
部署管理
管理者
部署情報の入力
部署の登録
生成
部署
生成
部署DAO
部署情報セーブ（部署）

（２）アーキテクチャ説明書

バージョン 0.1

目次

〇永続化について（ユースケースビュー及び論理ビューの一部）

・Data Access Object (DAO)パターンの利用。

目的:

ビジネスロジックと永続化の実装を分離して、永続化の方法、データベースの違いを吸収する。

DAO の役割:

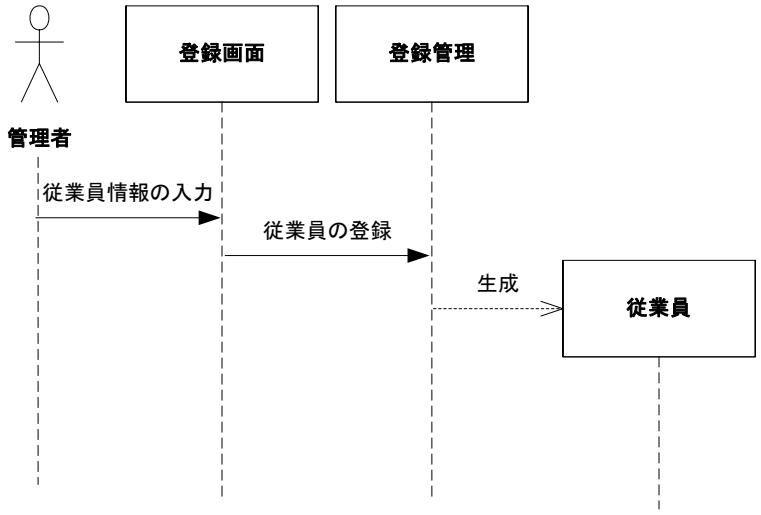
データの CRUD を DAO が受持つ

データベースのコネクションなどの処理を受持つ

・例　簡単な例（従業員登録）でシステム分析モデルに DAO パターンを導入する

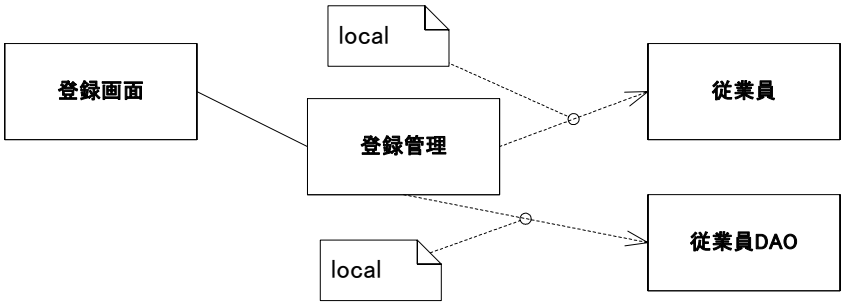
システム分析のクラス図

システム分析のシーケンス図



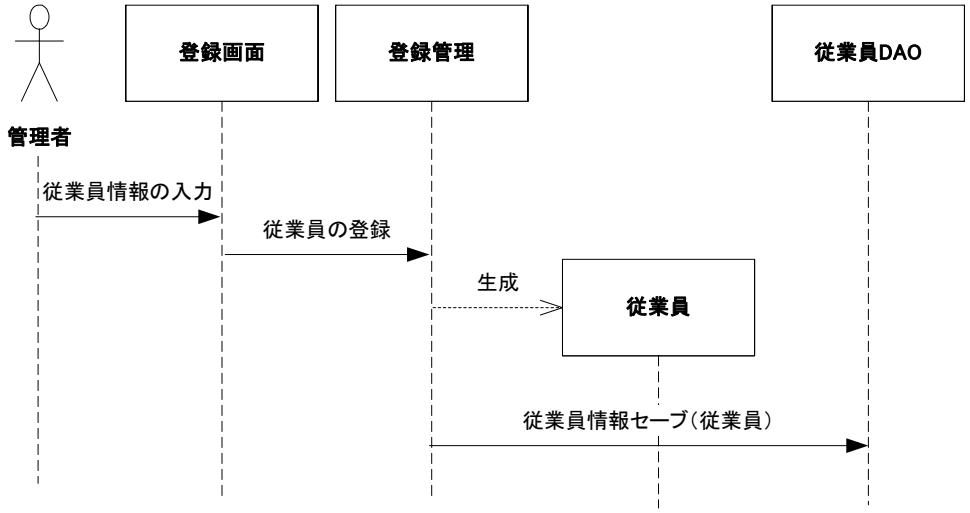
DAO パターンを導入した設計モデル

クラス図



※ local とは、依存関係元のクラスが持っており操作の中で、依存関係先のクラスのオブジェクトを生成することを示す。

シーケンス図



「従業員 DAO」クラスが、「従業員クラス」のデータベースへのデータのセーブ、ロードを受け持つ。

.
.
.

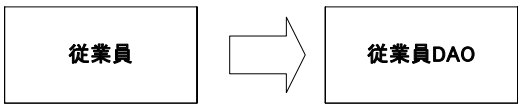
（３）詳細設計ガイドライン（一部）

設計クラスの追加

・永続化　Data Access Object (DAO)パターン

永続化を必要とする1つの entity クラスに対して、1つの DAO クラスを作成する。

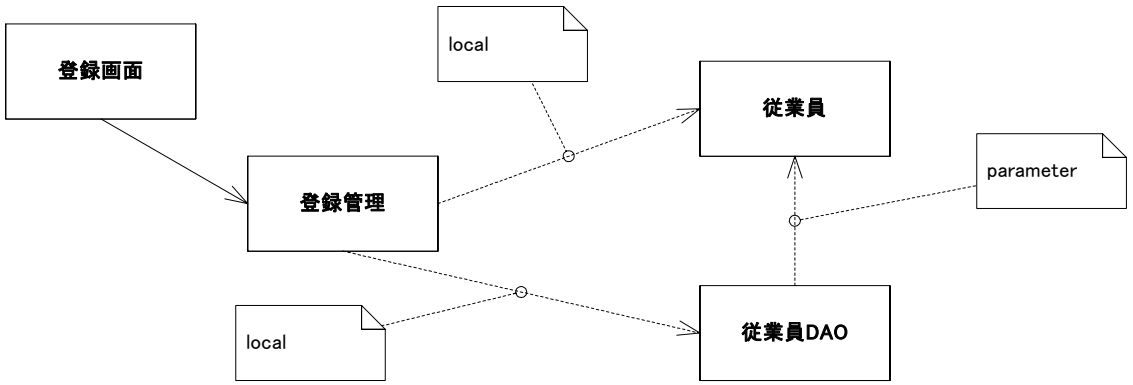
DAO クラス名は、entity クラス名 + ”DAO”　とする。



・クラス図

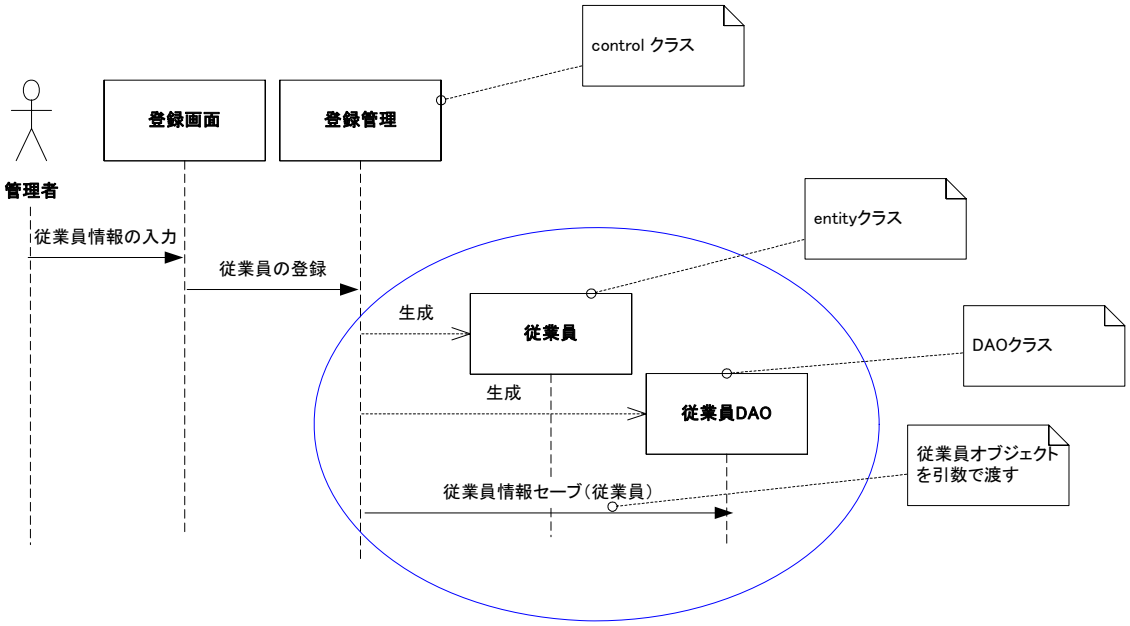
control クラスは、新規にローカルに、DAO クラスのオブジェクトを作成し、また entity クラスも新規にローカルに作成するので、依存関係を引き、「local」とコメントを入れる。

DAO　クラスは、entity　クラスを引数として渡されるので、依存関係を引いて、「parameter」とコメントを入れる。



シーケンス図

DAO クラスのオブジェクトは、control クラスがローカルに作成して、呼び出す。セーブする場合は、entity クラスを引数に渡す。



特定非営利活動法人 UML モデリング推進協議会
オフショアソフトウェア開発部会

ガイドライン作成メンバー

・王　春生	Oriental Standard Human Resources Co., Ltd. Beijing
・大下　美穂	バブ日立ソフト株式会社
・小倉　博	オープンワークス株式会社
・加瀬　直樹	株式会社東芝
・神岡　太郎	一橋大学商学研究科
・許　炎	通華科技（大連）有限公司
・今野　隆一	ジャパンシステム株式会社
・齋藤　肇	日本海隆株式会社
・正田　塁	株式会社オージス総研
・高橋　和司	アドソル日進株式会社
・竹政　昭利	株式会社オージス総研
・中原　俊政	バブ日立ソフト株式会社
・幅　洋平	バブ日立ソフト株式会社
・浜川　剛	株式会社テクノロジックアート
・日高　綱弘	日本インフォメーション株式会社
・藤野　博之	NEC ネクサソリューションズ株式会社
・吉田　亮	日本アイ・ビー・エム株式会社
・梁　暁冬	明治安田システム・テクノロジー株式会社
・若林　良二	NEC ネクサソリューションズ株式会社