内容

はじめに 1
1. 方式設計 1
1.1. リスク分析 1
1.2. システム寿命と採用技術の将来性 2
1.3. 採用技術の費用対効果 2
1.4. 方式設計で決めること 3
2. 業務分析 4
2.1. 統一モデリング言語（UML） 4
2.2. モデリングとは（UML に書かれないもの） 6
2.2.1. UML によるモデリング 6
2.2.2. 状態はオブジェクトか属性か 6
2.3. 開発範囲・計画 7
3. 開発プロセス 8
3.1. アジャイルとは 8
3.2. 利点 8
3.3. 短所 8
3.4. その他の特徴 8
3.5. アジャイルはどこで使う 9

オブジェクト指向―分析

はじめに

分析工程では、当該プロジェクトでのシステム化の範囲を決める（業務分析）とともに基盤として採用するアーキテクチャを決めます（方式設計）。システム化の範囲とは経営層がシステム化投資の対象として認識している業務領域で、要件定義により提示されます。

1. 方式設計

方式設計では主に非機能要件について検討します。一般的に以下の要因について検討が必要です。

- システムのリスク（内在するリスク、許容可能なリスク）
- システム寿命と採用技術の将来性
- 採用技術の費用対効果

1.1. リスク分析

以下にシステム開発時に考えておくべきリスクについて、一般的な例を挙げます。

（1）セキュリティ

リスクの種類や重大度はシステムの種類により異なります。アクセス範囲が社内のネットワークだけであれば、リスクは少なそうに見えますが重要な情報を扱うのであれば、利用者がネットワークのブリッジをしてしまっていたりウィルスの持ち込みを防ぐために閉じた専用のネットワークが必要かもしれません。逆に、サーバ上で営業秘密は取扱わない、参照だけのシステムであればプロジェクト固有のセキュリティ施策は不要という判断も可能です。

（2）他システム接続等

複数のシステムが接続してデータのやり取りをしている場合、一つのシステム障害が連鎖して大規模障害に繋がることがあります。ある企業では、運用部門に新規・改修システムの運用を依頼する際には他システムとの関係及び停止した場合の影響について連絡承認が義務付けられていましたが、現在この企業は企業統合／システム統合の後、大規模トラブルが続いています。

融合による複雑さが制御できるレベルを超えたら、再度 分界線を引き直してダウンが連鎖しないようにするべきです。例えば JR では宇都宮線、高崎線と東海道線に直通運用が開始されて以降、山の霧で海沿いの線が止まり、海浜に大風が吹いて内陸の線が遅れ、ローカル障害が首都圏全体にエスカレートする状況になりました。利用者の不便を上回る便益が得られたのか評価すべきでしょう。

（3）運用の継続性

重要なシステムでは、想定外のことが起こることも想定してフォールトトレラント、フェールセーフを考慮しておく必要があります。機器が停止する理由の全ては想像しきれなくても、停止したらどうするかを機器毎に考えておくことはできるでしょう。

また、ソフト障害に起因するものは待機系に切り替えても切替先で同一の障害が発生する可能性が高く、二重化したら安心とはなりません。物理的／ソフト的な攻撃への対処もシステムの重要度に応じた考慮が必要です。

1.2. システム寿命と採用技術の将来性

システムは投下費用を回収するために必要な期間が計画上の最短寿命になります。システムに採用するアーキテクチャに関しても同様に、実績のある技術を選んでも費用回収の前に技術の寿命が来てしまったとなったら投資としては失敗です。

Web を中心とした大規模なシステムでは Java が選ばれることが多いですが Java にはバージョン毎のサポート期限があります。ブラウザや AP サーバ、各種フレームワーク、ライブラリもセキュリティ対策や新機能の追加は継続し、時には新しい技術への置き換えが起こります(例えば、Flash 等)。互換性を保ちつつ関連資産の構成も置換していかざるをえません。

以上のことから、次の方針が考えられます。

- 業務ロジックはフレームワーク／ミドルウェアから独立して再利用を考慮した実装にする
  …例えば、所謂 POJO に業務ロジックを切り出しフレームワークからは呼び出すだけにする
- 極力自前で開発せず、BI/ETL ツールを利用する
- C や COBOL のような機能拡張が終わった言語環境を使う

1.3. 採用技術の費用対効果

システムの重要度と予算からアーキテクチャを決めることになりますが、安心感のため多少割高になってもメーカーが出している製品を使いたいとういうユーザもいます。しかし、製品であっても中核部分でオープンソースをベースにしている（参照実装の商用利用等[1]）ことがあり、セキュリティ対策が海外での対策待ちになったり製品自体の入れ替えを強く推奨されることがあります。メーカーの製品を調達すれば安心というわけにはいきません。また、ある程度普及した技術でないと開発技術者を集めることが難しく費用の増大要因になります。

[1] Java の場合 仕様の動作基準として参照実装が作られます。AP サーバの参照実装で有名な Tomcat/Catalina をはじめ各種のオープンソースが使われています。
https://www.google.com/search?q=OSSRA

1.4. 方式設計で決めること

方式設計では、前項までの内容を前提条件としてシステムの実現方法を決めます。

環境として、ネットワーク構成を含む可用性の要求水準に見合うのはクラウドか自前のサーバか使える言語は Java か Python か Ruby か PHP か COBOL...か、適合するフレームワーク、ミドルウェアの候補はなにか。これらは要求仕様としてユーザから提示される場合もあります。

フレームワークは生産性、保守性に影響します。業務内容に依存しないワンパターンな AP 構造の方が要員の習熟とともに生産性や品質が上がります。また、現状の最適を求めすぎると将来の業務ルール変更への対応が困難になることもあるので、先々の保守を見越した選定が必要です。

業務開発を行う要員としては、使いやすい開発ツールやソース・環境定義の自動生成ツール等も欲しいでしょう。運用を行う要員は運用が楽で安全な方式を望みます。まず、知見を持つ類似プロジェクトを探してみてください。

最後に、業務を検討する側に対して以下の情報を提示します。

・フレームワーク、ミドルウェアと業務実装とのインタフェース（使用 API）や環境定義内容
・業務実装でやるべきこと、やってはいけないこと
・エラー処理、ログ出力等の方式・業務共通機能の使い方

※ここまでの内容は業務設計に着手するまでに実施すべき内容ですが、全てを方式設計の中で行うわけではありません。予算額の大きなシステムでは、分析工程着手の前に投資案件として成立するかある程度の具体的な構成を想定したうえで実現可否の検討とシステム化の計画を行います。

2. 業務分析

オブジェクト指向による分析は統一モデリング言語（UML）を使った対象領域のモデリングと反復型の開発プロセスが特徴で、データフローダイアグラム（DFD）＋ウォーターフォール型開発の構造化分析と対比されます。

2.1. 統一モデリング言語（UML）

UML は図式を使って仕様を表現する手法で、UML2 では 13 種類の図式があります。分析工程では数種類の図式を使い、要求事項の確認やシステム化対象のモデリングを行います。

（1）クラス図（概念クラス）

対象業務を説明する言葉の関係を大雑把にクラス図にしたものを概念クラス図といいます。

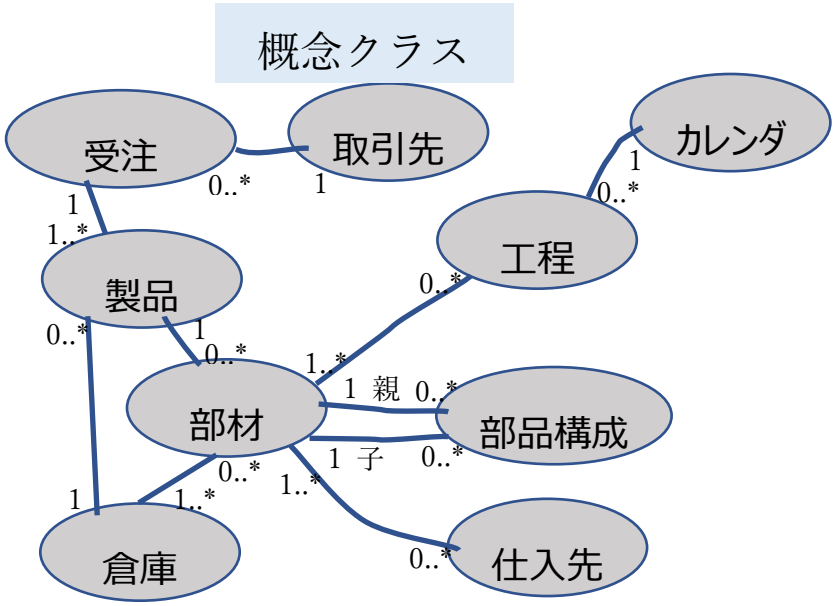
業務を説明する言葉は更に別の言葉で細かく説明できるか[2]もしれませんが、概念クラスは深堀せず広く漏れなく洗い出します。クラス図は実装からツールを使って作成できるので、特に保存を求められない限りは手書きで十分です。クラスをつなぐ線（関連）に書かれている数字は多重度で、これが業務ルールを表します。

例えば、下図の多重度から以下の業務ルールが存在することが分かります。

① 製品は一つの倉庫とのみ関係付く … 製品毎に収納倉庫が決まっている
② 部材に関係付く倉庫が一つ以上 … 部材は複数の倉庫に分散している
③ 製品から見て部材が存在しないことがある … 製品購入がある⇒製品と仕入先に関連要？

※上記の内容に忠実に設計を行った後、例外に気が付いたり変更があった場合に影響範囲が大きくなります（しかも割と頻繁に発生する事態です）。必ず次の検証が必要です。

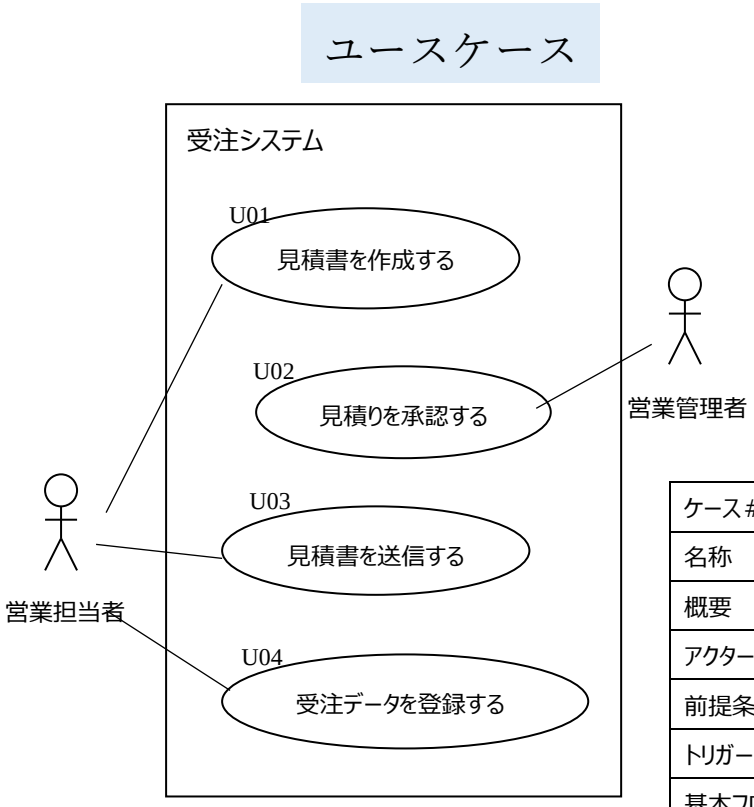
・ユーザからのヒアリング内容は本当に正しいか？矛盾はない？ユーザ・部門間の認識は一致？
・近い将来にこの業務ルールが変更される可能性は無いか？（可能性がある変更と対策は？）



[2]業務的に価値がありそうな言葉（名詞）を拾い集める。…オブジェクト指向でよく使われる“概念”になります。例えば“受注”は“見積”、“契約”、“製造中”...他、の概念を構成要素に含みます。

（2）ユースケース

ユースケースの粒度がユーザの認識する仕事の単位になっていると仮定し、この単位で実装内容を考えます。例のユースケース「見積書を作成する」では、納期／価格のシミュレーションと見積書作成に関連する一連の手続きを規定します。



＜ユースケース記述＞

ケース#	U01
名称	見積書を作成する
概要	顧客の依頼を受け、価格と納期を見積もる
アクター	営業担当者
前提条件	見積条件書の受領
トリガー	顧客からの見積依頼受領
基本フロー	①アクターは見積依頼画面より商品コードと数量、納品場所、その他条件を入力する ②システムは商品の部品展開を行い所要量計算と日程計画シミュレーションより納期、売価を算出し、画面に回答納期と価格を表示する ③アクターは値引き、ロット調整等を行い再度①を行うか、見積書確定を行う ④システムは提携様式の見積書を作成する
例外	①-E：商品コードの登録がない場合～ （以下略）

2.2. モデリングとは（UML に書かれないもの）

モデリングは会社等の組織の見え方を図に写すという点で具象画を描くことに似ています。具象画は目の前の静物に対して、美術品として「必要な部分」以外を削ぎ落すことで価値がでます。モデリングも同様の方法で現実の切り取りを行います。

2.2.1. UML によるモデリング

モデリングでは、UML を使い以下の切り口で分析を行います。

（1）概念間の静的な関係

概念クラス図は概念間の静的な関係を表します。

（2）トランザクション

ユースケース図は処理を行うきっかけと、それに応じたシステムの振る舞いを記述します。

（3）処理の順番・内容

アクティビティ図で業務の手順等（業務フロー）を表します。アクティビティ図には実行する順番、内容、条件等を書きます。

（4）データの変化

状態遷移図でシステム稼働中のオブジェクトの状態の遷移を表現できます。

但し、この「状態」とはスィッチのオン／オフや装置の停止中／待機中／動作中等を扱うもので、見積中／受注／製造着手のような保存が必要な状態に関して使われることはありません。

2.2.2. 状態はオブジェクトか属性か

UML はシステムの静的な状態を分析するものが多いので、データ処理中心のシステムでは UML を記述して設計工程が終わったにもかかわらずシステムの肝心な部分が決まっていなかったということになるかもしれません。

例えば、オーダー品の見積依頼～納品・検収に関する以下の業務フロー・データをどう扱えばよいでしょうか。

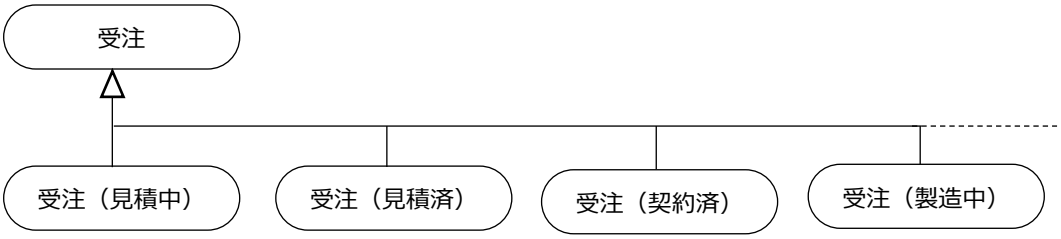
・見積中 …顧客依頼受け見積中
・見積済 …社内承認／顧客提示済
・契約済 …受注・契約
・製造中
（以下、略）

※前提の業務フローは、顧客より見積り依頼を受けた段階で必要な仕様を受け取り“受注データ（見積中）”として登録します。また、必要な製品登録等も同時に行っておきます。後は、見積書の送付や、受注等のイベントに合わせて情報を管理していきます。

（1）状態をオブジェクトとする場合

状態のオブジェクトは以下のクラスから作られ、この場合はクラス図に状態が表せます。

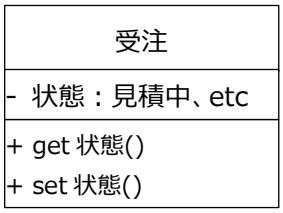
RDB へ保存する場合は各状態毎に異なるテーブルに格納します。



（2）属性として持つ場合

オブジェクト／クラス／テーブルは一つで、属性を更新して契約の進捗状態を管理します。

この場合、データのライフサイクル（状態遷移）は CRUD 図等により定義し破綻がないことを検証します。



（3）長短

サービスや物品の販売を行う企業のシステムでは「企業活動＝トランザクションの処理」であり、この例のようにトランザクション・データの状態が変化して（させて）いくことが重要な企業活動になります。したがって、状態遷移をオブジェクトで捉えことができる（1）が分析としては正しいようにみえます。

一方で、「状態」の数が増えたら（例えば、“見積二回目”、“検収戻り”等）その都度、大きな改修が入ります。また、顧客からの問い合わせがあった場合や顧客毎のバッチによる集計等が必要になったら「状態」を横断して検索が必要にり、属性で管理した場合に比べて運用が複雑です。メモリサイズや ORDB の性能や透過性がオブジェクト／クラスの設計の制約になります。

2.3. 開発範囲・計画

概念クラス、ユースケースから開発のサイズ（所要工数、費用、予定期間）を調整し、予算を超える場合は優先順をつけて着手する範囲を決め開発計画とし確定します。

3. 開発プロセス

1990 年代に Rational Software 社が UML とその関連ツール、開発プロセス(Rational Unified Process; **RUP**)を策定し、その後の J2EE の隆盛、2003 年の IBM による Rational 社買収と PR により“オブジェクト指向”、“ＵＭＬ”、“RUP 反復型開発プロセス”が三位一体で有名になりました。

オブジェクト指向と反復型開発プロセスは一体のものではありませんが、プロジェクトを成功させるための重要な要素として開発プロセスが注目されています。

RUP と並ぶ開発プロセスにアジャイルがあり、しばしばウォーターフォール型開発と比較して語られます。RUP とアジャイルは目的と重点を置く場所が若干異なるので、以下、より一般的なアジャイル開発プロセスを基に纏めます。

3.1. アジャイルとは

アジャイルに分類される中にも数多くの手法が唱えられています（IPA では「スクラム」推しのようです[3]）が、基本はシステム全体を平行して完成させるのではなく、小規模な「設計-実装-テスト」のサイクルを繰り返し[4]ます。スクラムでは、数週間の単位に作業を区切りスクラム＝チームを中心に作業を行います。

3.2. 利点

重要な利点の一つは、ウォーターフォールは長い時間を掛けないとユーザに分かる成果物がでないのに対して、反復型開発プロセスは結果がすぐでるのでユーザとの認識の齟齬が早く分かり、時間の無駄が無い（少ない）というものです。

もう一つ重要な点として、発注者と事業者の「協働」による成果の享受を謳ってい[5]ます。

3.3. 短所

長く時間が掛かる作業、例えば、先行調査や基盤の構築などはチケット化が難しい。１週間程度で作業を切って別チケットで作業を継続させることはできますが、スクラムマスターの力量次第で締め切りの無い無監視作業になりかねません。

3.4. その他の特徴

ウォーターフォールで必要になる次工程向けのドキュメントは作らないので、ドキュメントは実装になります（別途、マニュアルは必要です)。ドキュメントを作らない分、進捗は早くなりますが基幹系のような多数の業務ルールを詰め込むシステムでは「ナゼそうしたか」が残らず、実装者がいなくなってからの改修作業で問題が発生することがあります。背景の分からない処理を削除したり変えたりするには相当の勇気が必要になるためです。

---

[3] https://www.ipa.go.jp/search.html?q=スクラム

[4] RUP ではスパイラル（漸進的）に反復する

[5] アジャイル開発実践ガイドブック 2021 年（令和３年）3 月 30 日 内閣官房情報通信技術（IT）総合戦略室

3.5. アジャイルはどこで使う

プロジェクト推進の観点からは長時間掛かる作業をチケットで行うのは不向きです。また、基幹系の先行事例があるようなシステムを構築するプロジェクトではそれほど認識違いなものが出来上がる可能性は少ないため、先行事例と異なる機能だけをモックアップで作ったりアジャイル的にパイロットプロジェクトで進めることができます。原則として、大規模システムでは設計書でシステムの背景事情を残すウォーターフォールを、既存の方式に載る限定的な機能追加や改修であればスクラム／チケット管理によるプロジェクトが向いています。

もう一点、事業者の立場からは別の観点があります。基幹系の刷新プロジェクトで事業者が発注者に訴訟を起こされる事例が何度か起きています。

パッケージの改造による開発の場合、発注者はパッケージの具体的な内容を理解しておらず、事業者は発注者の文書化されていない要求に気づいていない状態で進めてしまうとこうなります。相互理解に自信が持てなかったら、機能毎の試験をチケット化して「協働」作業をした方がよいでしょう。費用はパッケージの単純導入よりも掛かりますが訴訟費用よりは安くなると推測します。

マイグレーション中心でシステムを構築する場合は、「現行通り」という言葉を通したら訴訟か大赤字への道に繋がっています。「現行通り」が起きるのは発注側の責任者が現行のブラックボックスに責任を持ちたくないからでしょうが、事業者が負うべきリスクでもありません。このような場合はスパイラル型の協働作業で推進するのが適切です。

以上

