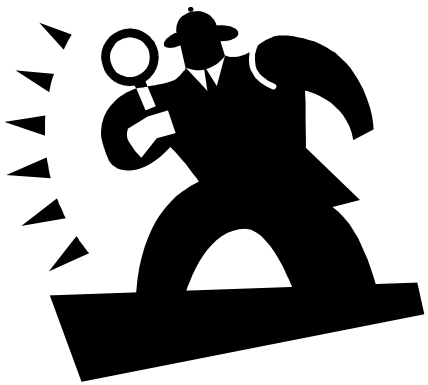
アジャイル開発の実践と評価
～何故周囲で利用がされていないのか～

平成25年度　OISA技術研究会
アジャイル部会　研究成果発表

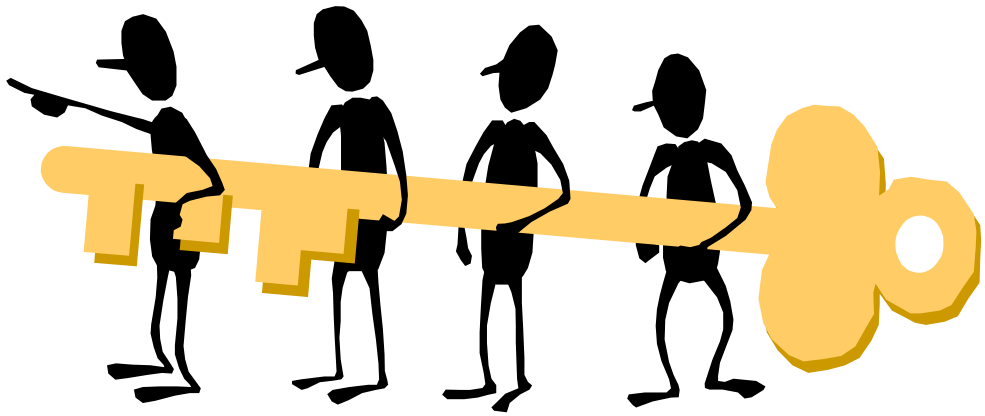
部会員紹介

- **部会員 （順不同）**
  - 榮倉　健　　九州東芝エンジニアリング株式会社
  - 岩男　奈々　　株式会社オーイーシー
  - 松吉　宏剛　　株式会社オーイーシー
  - 兵頭　勇輝　　三井造船システム技研株式会社



目次

- 第1章
  - アジャイル開発とは
- 第2章
  - アジャイル開発実践及び感想
- 第3章
  - アジャイル開発を通しての評価
- 第4章
  - まとめ



第１章
アジャイル開発とは

アジャイル開発とは

- 変更があることを前提としてイテレーション（反復）開発を行っていき、実装、テスト、リリースを繰り返し開発を進める手法

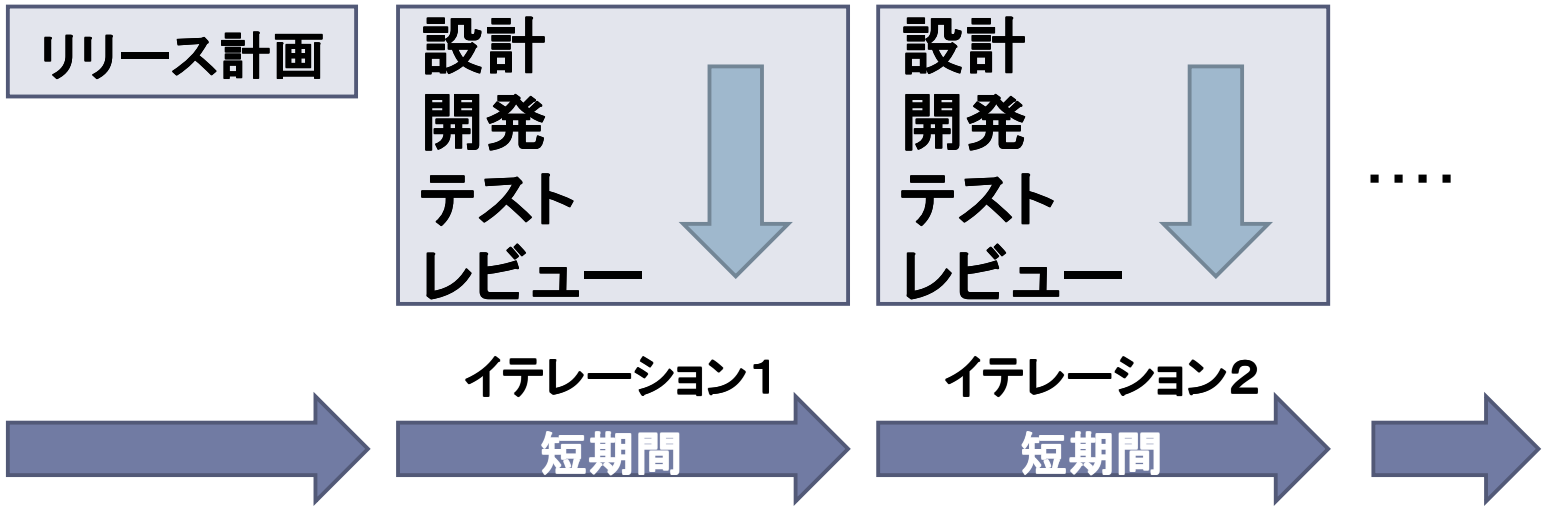


図1．アジャイル開発の流れ

アジャイル開発の特徴

- 仕様書等のドキュメント作成は必要最低限の作成のみ
  中間のドキュメントは作成せず、最終成果物に対する、
  ドキュメントの作成を行う

- 開発期間を短期間に区切り機能の作成、リリースを繰り返す
  ため、動作するものを早い段階でユーザーが確認できる

- 開発当初にリリース計画を設定し、計画に沿って開発を行う
  - **リリース計画**
    何番目のイテレーション（反復）で何を実装するか
    短期間のスケジュールを作成していくこと

ウォーターフォール開発とは

- 開発工程を要件定義、設計、開発、テストとし、各工程を順に行う開発手法のこと

図2. ウォーターフォール開発の流れ

ウォーターフォール開発の特徴

- 要件定義⇒基本設計⇒詳細設計⇒・・・のように工程ごとに開発サイクルを分割できるため、作業状況を明確に把握することが可能

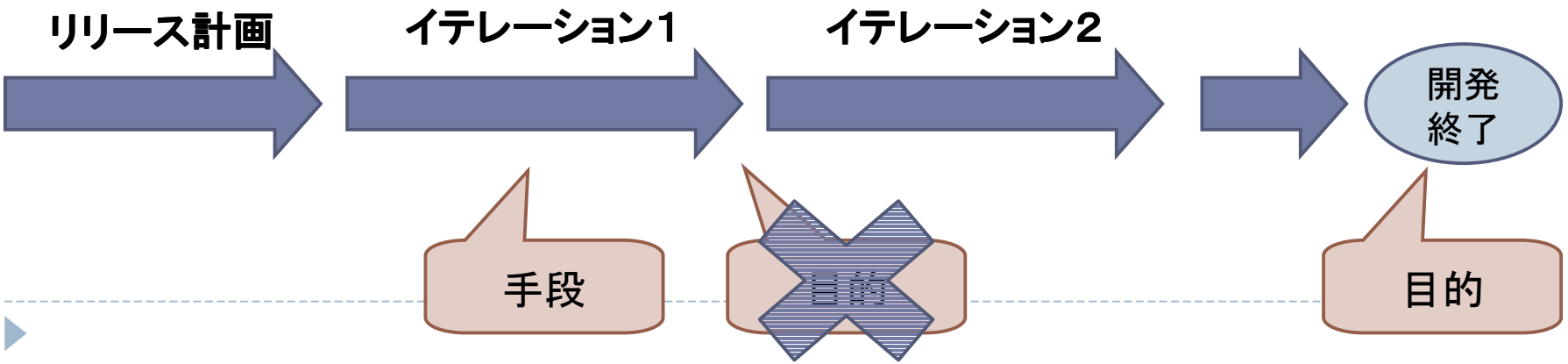
- プログラム開発前に要件を決めることによりスケジュールの遅延や要求するシステムとの差異を低減することが可能

各開発の比較

	ウォーターフォール開発	アジャイル開発
スケジュール	各工程が決まっているのでスケジュール作成や管理が容易	終了時期が明確でないため全体スケジュールの作成が困難
要求の変更があった場合	前工程に戻る場合が多く対応が困難	変更があることを考慮して開発を行うため要求の変更に対応しやすい
リリース回数	1回	複数回
設計書	詳細に表記 開発前に作成	必要最低限の表記 最終成果物に対し作成
ユーザーによる動作確認	開発終盤で確認	各イテレーションごとに確認

調査課題

- 小さい単位で実装、テスト、レビューを繰り返すため、全体のスケジュールや進捗を把握しにくい
  また、最終リリースを厳密に定めることは厳しい
- 大規模なシステム開発では収拾をつけにくい
- 手段と目的が置き換わる場合がある



調査目的

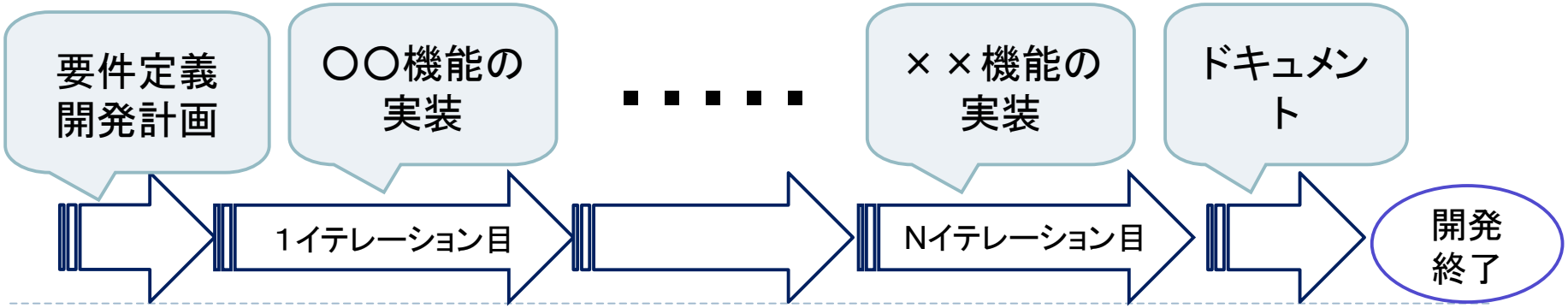
一般的に言われている問題を調査し、開発を通して
検討することでアジャイル開発について理解を深める

第２章
システム開発実践 及び感想

開発の流れ

- **以下の流れを意識して開発に着手。**
  1. **開発要件の確認**
  2. **開発計画の設定**
  3. **開発**
  4. **リリース**
  5. **開発終了**

ここのサイクルを繰り返す



開発要件

システム概要　：　美容室の予約システム

操作者　：　店内スタッフ

運用想定　：　お客からの電話が来た際に、
スタッフがその場で予約参照/登録を
行うことを想定。

想定運用

運用例

システムに必要な機能

- 担当者別、日別、月別等で予約が参照可能。
- 予約の登録（日時、メニュー等）
- 前回の情報を表示
- 担当者マスタ
  （画面は無くてよいが、複数担当者を扱えること。）

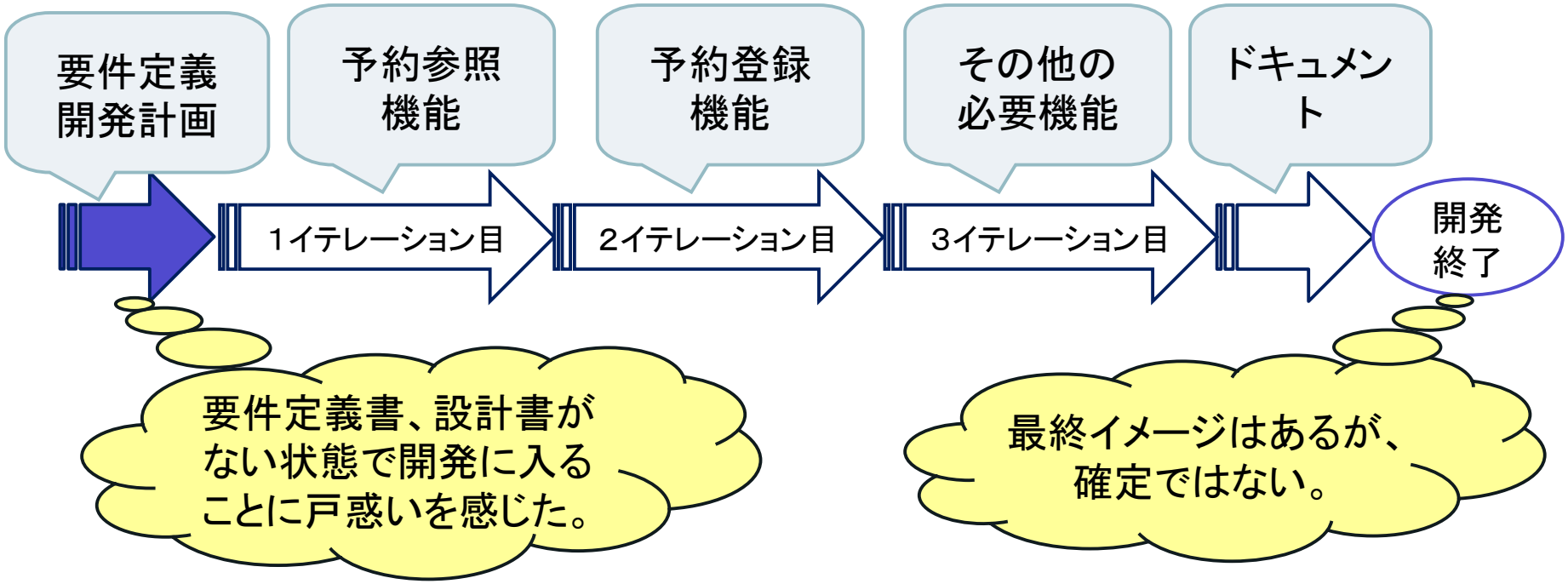
クライアントからの要求

- 予約の空いている日がすぐにわかるようなシステム
- 操作性がよいシステム（操作が簡単）
- お客様検索がすぐに出るようなシステム
  （電話番号＆氏名検索など）

開発計画（全体）

WF型開発の繰り返し
と何が違うのか？

今回は、開発イテレーションは「3サイクル」とし、
3回のイテレーションにて、システムの完成を目指す。



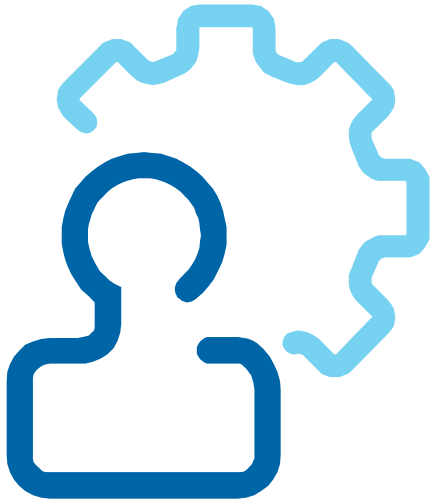
システム開発環境

・クライアント画面

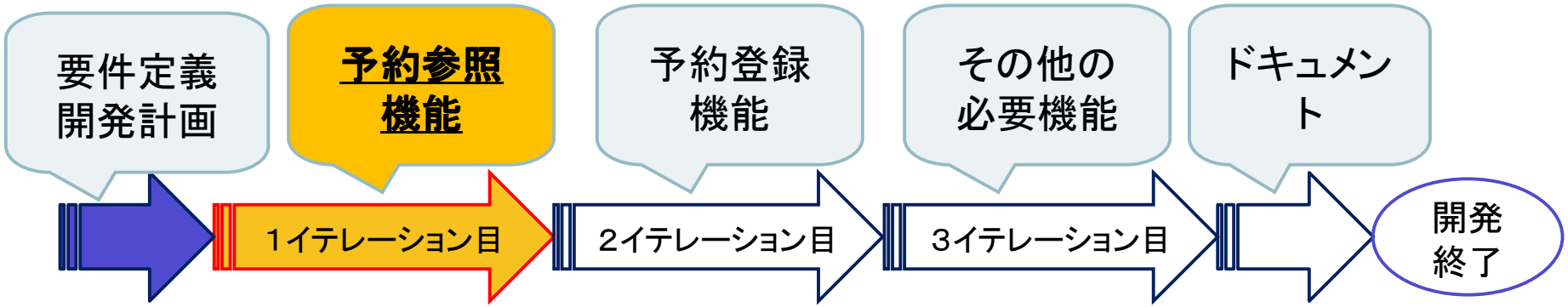
Microsoft Visual Studio 2010

・データベース

Microsoft SQL Server 2012



１イテレーション目の開発計画

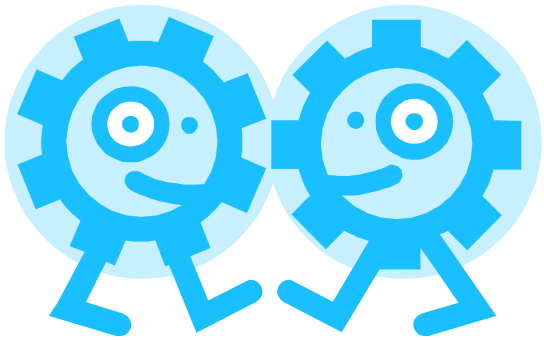


以下の機能を実装する。

- ●担当者別、日別、月別等で予約が参照可能。
- ●担当者マスタ

１イテレーション目の開発

開発中・・・・



１イテレーション目完了時のシステム画面

処理は、共通化や関数化を
意識する必要がある

一部分の機能に過ぎない為、
画面イメージも最終形を
意識する必要あり。

１イテレーション完了時のドキュメント

製造時に作成した資料

- DB設計書
- 画面イメージ

製造時に作成した、
認識合わせ用の内容を記載

DB設計書の内容

予約管理システム　データベース

スタッフマスタ[M_STAFF]

テーブル名	テーブル名(英字)	型	主キー	NOT NULL
スタッフID	staffId	CHAR(4)	○	○
スタッフ名	staffName	VARCHAR(8)		○
スタッフ名カナ	staffNameKana	VARCHAR(16)		
削除フラグ	deleteFlg	CHAR(1)		

顧客マスタ[M_CUSTOMER]

テーブル名	テーブル名(英字)	型	主キー	NOT NULL
顧客ID	customerId	CHAR(10)	○	○
顧客名	customerName	VARCHAR(8)		○
顧客カナ	customerNameKana	VARCHAR(16)		
性別	sex	CHAR(1)		
電話番号	tel	VARCHAR(11)		
郵便番号	zip	VARCHAR(7)		
住所	address	VARCHAR(50)		
特記事項	sspecialAffairs	VARCHAR(50)		
削除フラグ	deleteFlg	CHAR(1)		
更新日時	updateDate	DATE		
更新者ID	updateId	VARCHAR(10)		

時間マスタ[M_TIME]

テーブル名	テーブル名(英字)	型	主キー	NOT NULL
期間ID	periodId	CHAR(2)	○	○
時間ID	timeId	CHAR(2)	○	○
開始時間	startTime	CHAR(4)		○
終了時間	endTime	CHAR(4)		○

期間マスタ[M_PERIOD]

テーブル名	テーブル名(英字)	型	主キー	NOT NULL
期間ID	periodId	CHAR(2)	○	○
開始日	startDate	DATE		○
終了日	endDate	DATE		○

予約テーブル[T_RESERVE]

テーブル名	テーブル名(英字)	型	主キー	NOT NULL
日付	reservationDate	DATE	○	○
スタッフID	staffId	CHAR(4)	○	○
期間ID	periodId	CHAR(2)	○	○
時間ID	timeId	CHAR(2)	○	○
お客様ID	customerId	CHAR(10)		○
備考	remark	VARCHAR(50)		
予約時顧客名	reservationCustomerName	VARCHAR(8)		
予約時電話番号	reservationTel	VARCHAR(11)		
更新日時	updateDate	DATE		
更新者ID	updateId	VARCHAR(10)		

予約詳細テーブル[T_RESERVE_DETAIL]

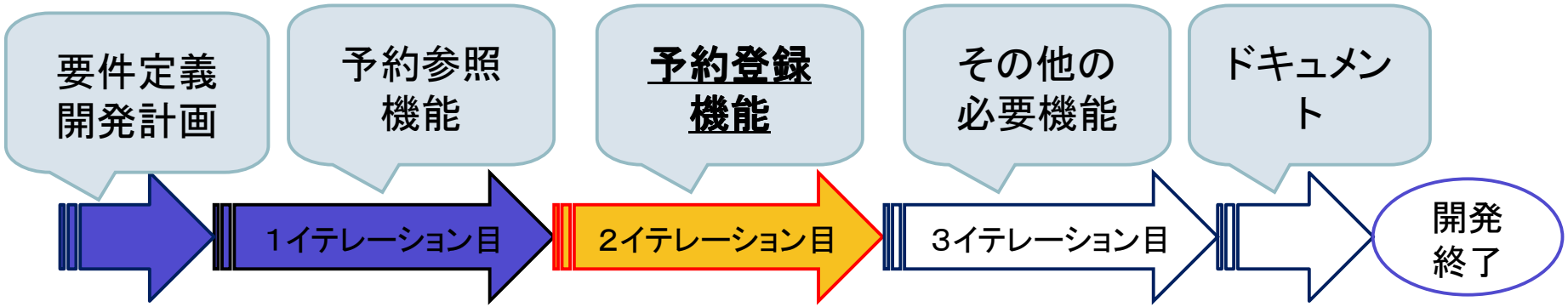
テーブル名	テーブル名(英字)	型	主キー	NOT NULL
日付	reservationDate	DATE	○	○
スタッフID	staffId	CHAR(4)	○	○
期間ID	periodId	CHAR(2)	○	○
時間ID	timeId	CHAR(2)	○	○
利用コード	workCode	CHAR(2)	○	○
更新日時	updateDate	DATE		
更新者ID	updateId	VARCHAR(10)		

利用マスタ[M_WORK]

テーブル名	テーブル名(英字)	型	主キー	NOT NULL
利用コード	workCode	CHAR(2)	○	○
利用名	workName	VARCHAR(20)		○
削除フラグ	deleteFlg	CHAR(1)		

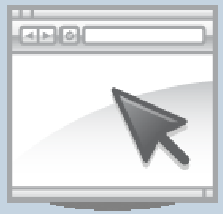
お客様マスタの性別は　0:男　1:女
時間マスタの時間は「○○：××～△△：□□」のイメージですが、もっといい案があれば変更してください
利用コード　1:カット　2:カラー　3:パーマ　など
削除フラグ　0:有効　1:削除

２イテレーション目の開発計画



以下の機能を実装する。

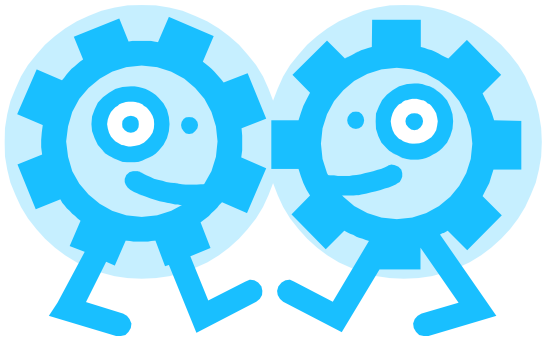
- ●予約の登録（日時、メニュー）
- ●前回の情報を表示
- ●DB接続情報の設定化 NEW



２イテレーション目の開発

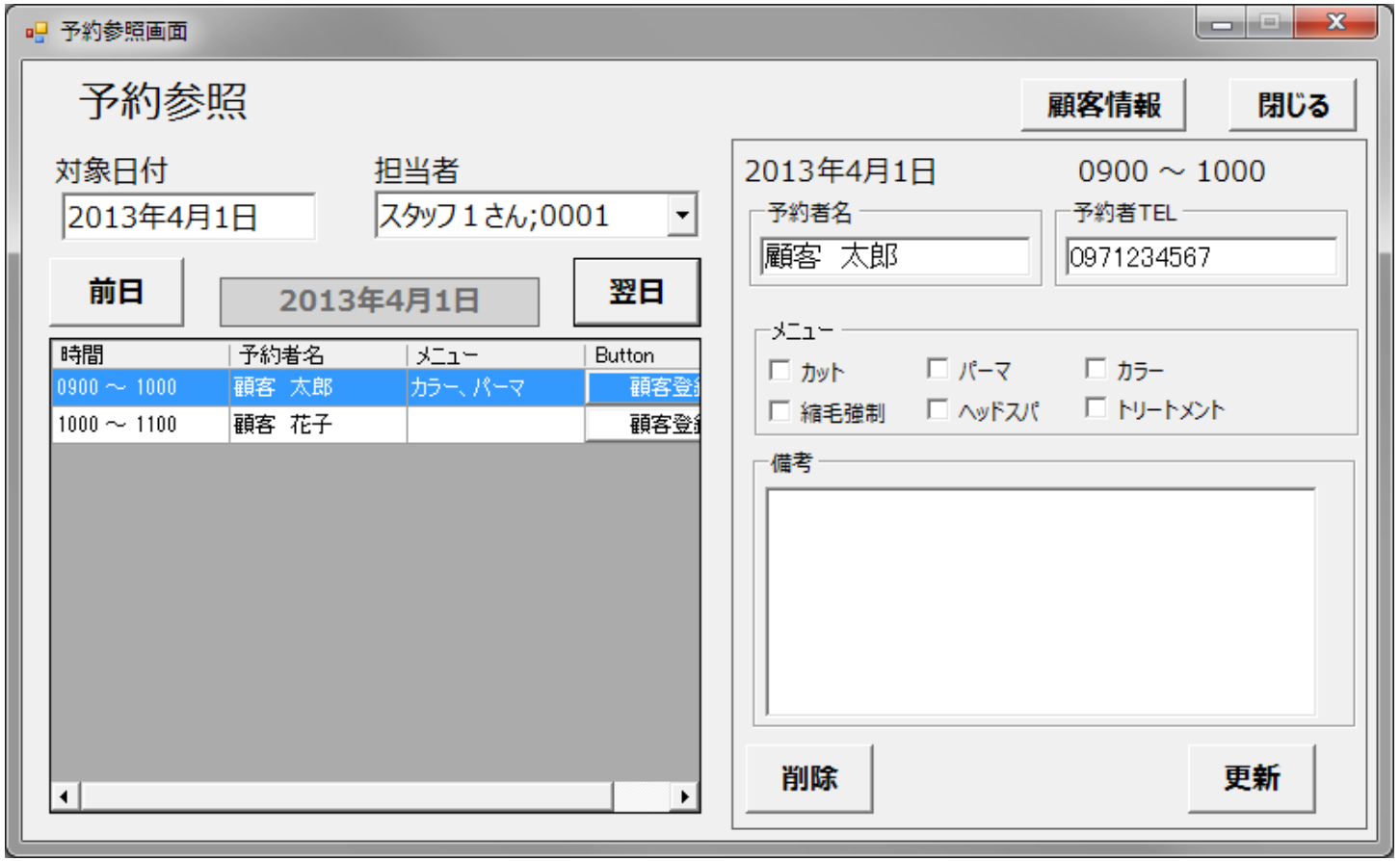
開発中・・・・

２イテレーション目の開発は
想定範囲内の為、特に問題発生せず



２イテレーション目完了時のシステム画面

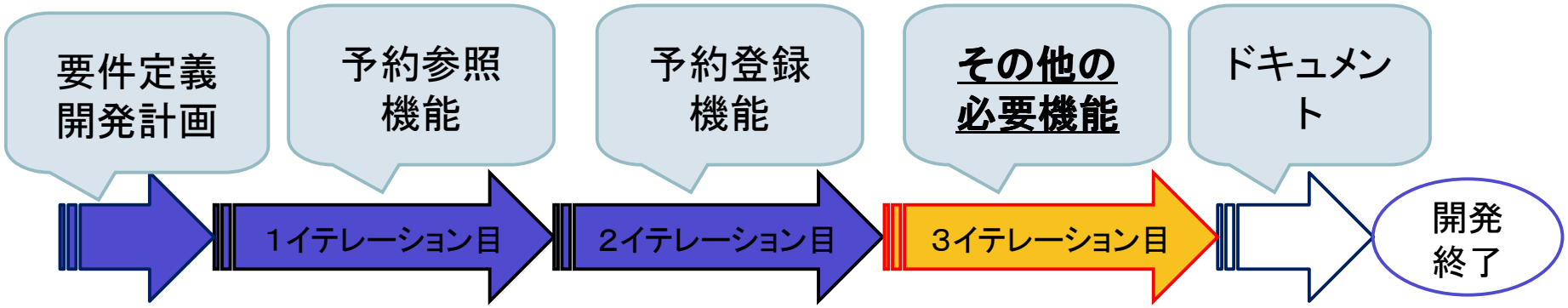
予約参照/登録画面



２イテレーション目完了時のシステム画面

顧客情報参照画面

3イテレーション目の開発



要望項目の下記の機能を実装する。

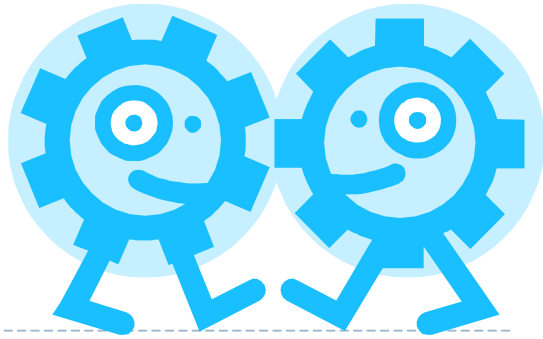
- 予約状況のカレンダー表示機能を追加
- 担当者の選択機能について機能強化
- 日付の入力機能について機能強化
- 予約の登録機能について機能強化

3イテレーション目の開発

短期間にリリースの
回数が多い。

時間の関係により製造できず。
・・・開発する想定で設計のみ考える

想定できていない要望は
内部ロジックによっては、
大きなリスクになる。



第３章
アジャイル開発を通しての評価

調査内容①

- 小さい単位で実装とテスト、レビューを繰り返すため、開発プロジェクト全体のスケジュールや進捗が把握しにくい
- 最終リリースを厳密に定められない

スケジュールを管理しづらい

実践結果①

調査：スケジュールを管理しづらい

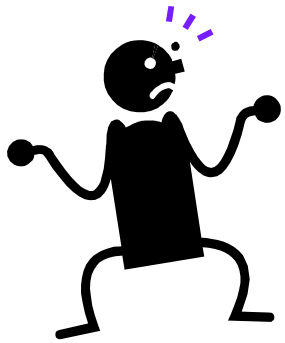
- 計画からのずれや変更、また、顧客要望の変更が発生した場合であっても、都度スケジュールの見直しを行い、正確な作業工数を算出することができれば、全体を把握することも可能。そのためには、コミュニケーションを積極的にとることが必要。

- リリース計画において、最終リリース日は決定している。

結果：見直しを都度行うことで管理可能

調査内容②

大規模なシステム開発では
収拾をつけにくい



実践結果②

調査：大規模なシステム開発では収拾をつけにくい

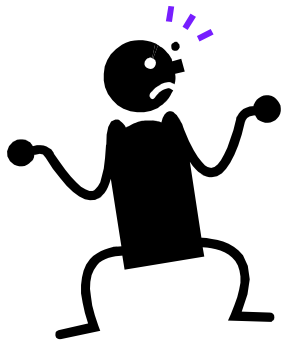
- 開発途中で顧客要望を受け入れた場合、今回のような小規模のシステム開発では、影響範囲が小さい。また、顧客の要望自体も比較的予想可能なものであった。

- 逆に考えると、規模が大きくなると影響範囲は大きく、長い開発期間の中で顧客要望が多様に変化することが予想される。

結果：大規模な開発への適用は困難

調査内容③

手段と目的が置き換わる場合がある



実践結果③

調査：手段と目的が置き換わる場合がある

- 「1つのイテレーションがゴールではない」ことを理解した上で開発を始めたが、振り返ると1つのイテレーションを終わらせることが目的となっていた。
- 最終成果物を見据えた開発（拡張性やコーディングルールの作成等）を行うことができなかった。

結果：目指すべき所を見失っていた

顧客から見たアジャイル開発

顧客から見たアジャイル開発

顧客から見たアジャイル開発

- 重要な機能からリリースされ、動くシステムに触れることができ、システム全体のイメージをつかみやすい
- 開発が始まっても、要望の変更を受け入れてもらいやすく、本当に使える、使いたいシステムに近づく
- 開発者任せではなく、プロジェクトの一員としてより良いシステムとなるよう協力する必要がある

第4章
まとめ

プロジェクト管理

- WFと同様もしくは、より細かい進捗管理が必要である
  - 今回の開発ではイテレーションの期限を守れなかった

細かい単位でのスケジュール管理が必要である

- 意思疎通が難しい
  - 開発時に設計書がそろっているわけでない為、
    開発者間で認識が異なり、作業が止まる場面が見られた。

開発者間の意思疎通のためのドキュメントが必須

開発機能の変化

- 全体スケジュールを把握することが難しい
  - 顧客ベースとなって仕様を決めるため、全体スケジュールの把握が難しい
- 正確な作業工数が把握しづらい
  - 要望機能の変化に伴い、作業工数が変化する

全体スケジュールにマージンをとる、
開発機能のトレードオフを行うなどの必要がある

機能的な品質

- 顧客満足度の高いシステムを作ることが出来る
  - 段階的に機能の開発/リリースを行うため、各タイミングで顧客の要望を反映することが可能となり、顧客が本当に要望するシステムを作ることができる。

顧客のニーズを吸い上げることが可能なため、
よりよいシステムを作ることが出来る

開発の進め方

- イテレーションごとの見積もりを早く出す必要がある
  - 短納期の開発が反復するため、イテレーションごとの工数の見積もりを、スピーディーに出さなければならない。
  - 見積もりを行いやすい環境（可読性の高いソース/熟知したPGが見積もる等）が必要である。

開発期間が短いため、拡張性のあるソースコードと理解しやすいプログラミングが必須である

まとめ

導入時の利点

・顧客満足度の向上

考慮すべき点

・PGのスキルは高いものを要求される。
・WF開発に慣れてしまっているため、
アジャイル開発に慣れる必要がある。

御清聴ありがとうございました。