【Embedded Technology 2019 スペシャルセッション】

派生開発問題解決セミナー2019
～派生開発プロセスの基本と発展　守りから攻めの派生開発へ～

スペックアウトによる設計仕様の理解

AFFORDD 派生開発推進協議会
T19研究会
井貝 智行

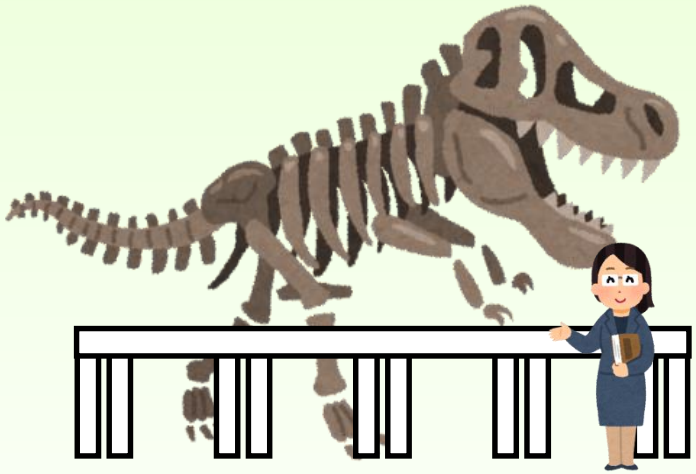
2019-11-22

目次

スペックアウトとは

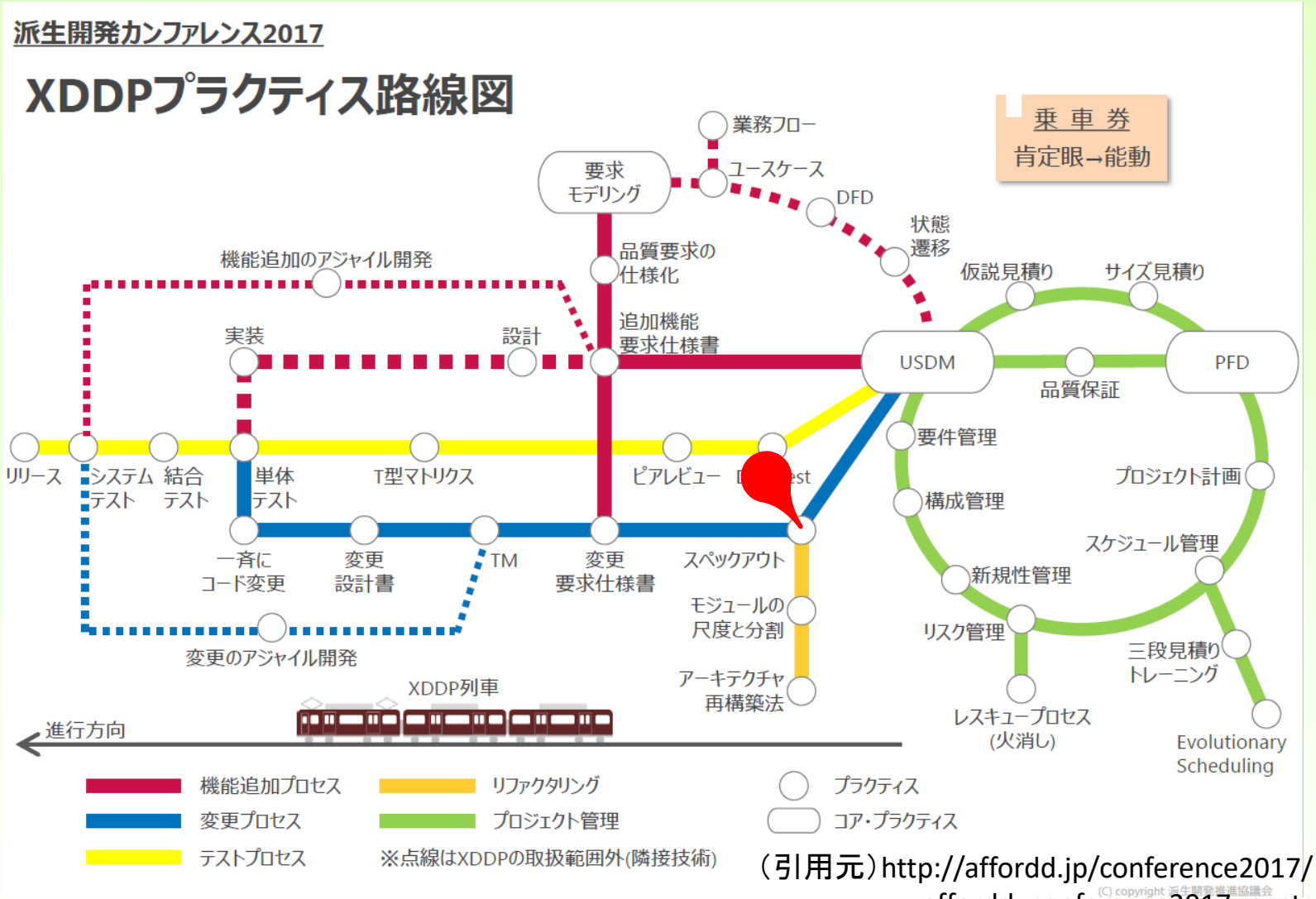
- XDDPでは
  - ソースコードから設計書の一部を生成する
  - ソースコードを読んで理解しながら
    変更箇所や影響箇所を特定する

ソースコード



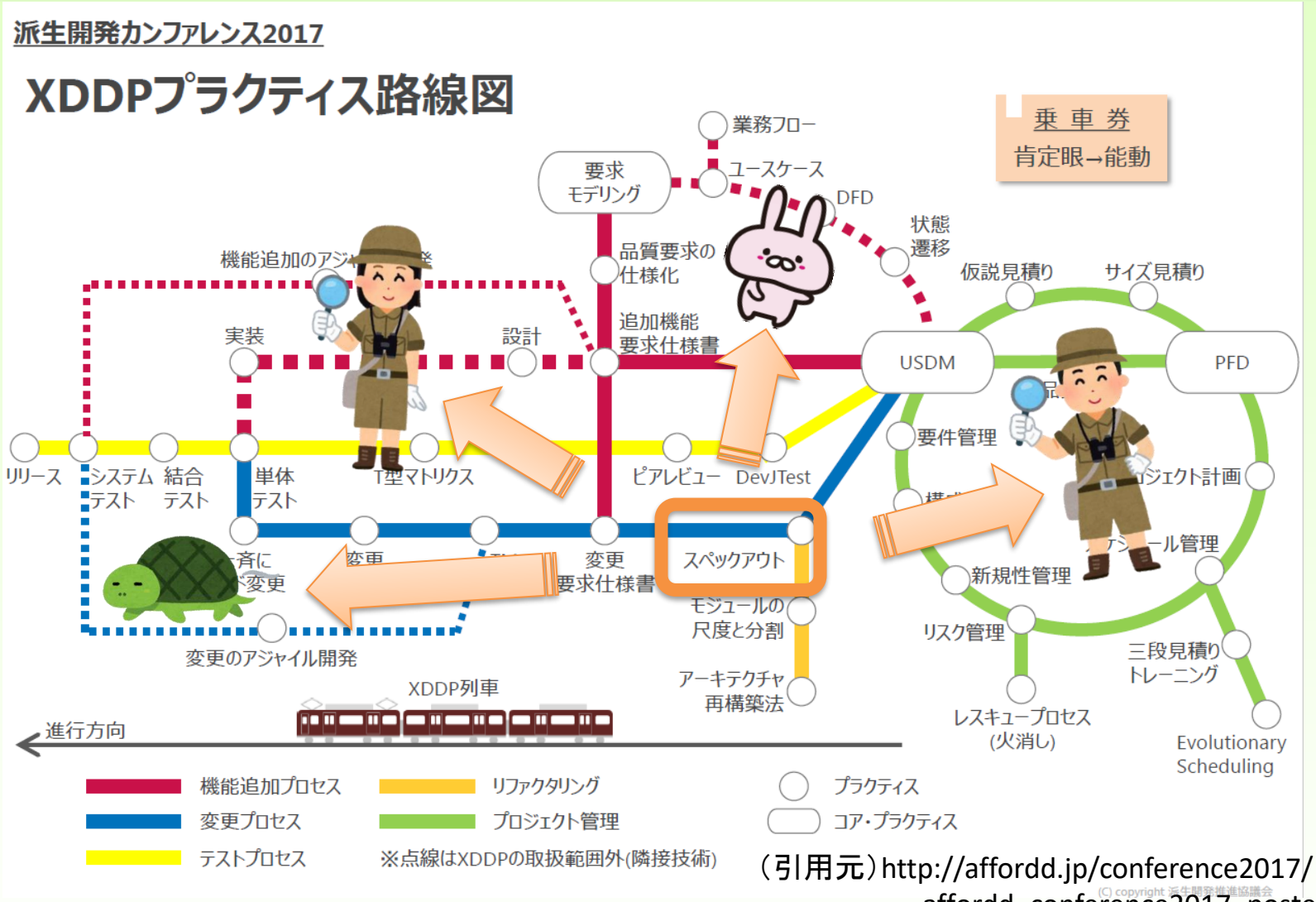
スペック(仕様)

XDDPにおける位置付け



（引用元）http://affordd.jp/conference2017/affordd_conference2017_poster1.pdf

XDDPにおける位置付け



（引用元）http://affordd.jp/conference2017/affordd_conference2017_poster1.pdf

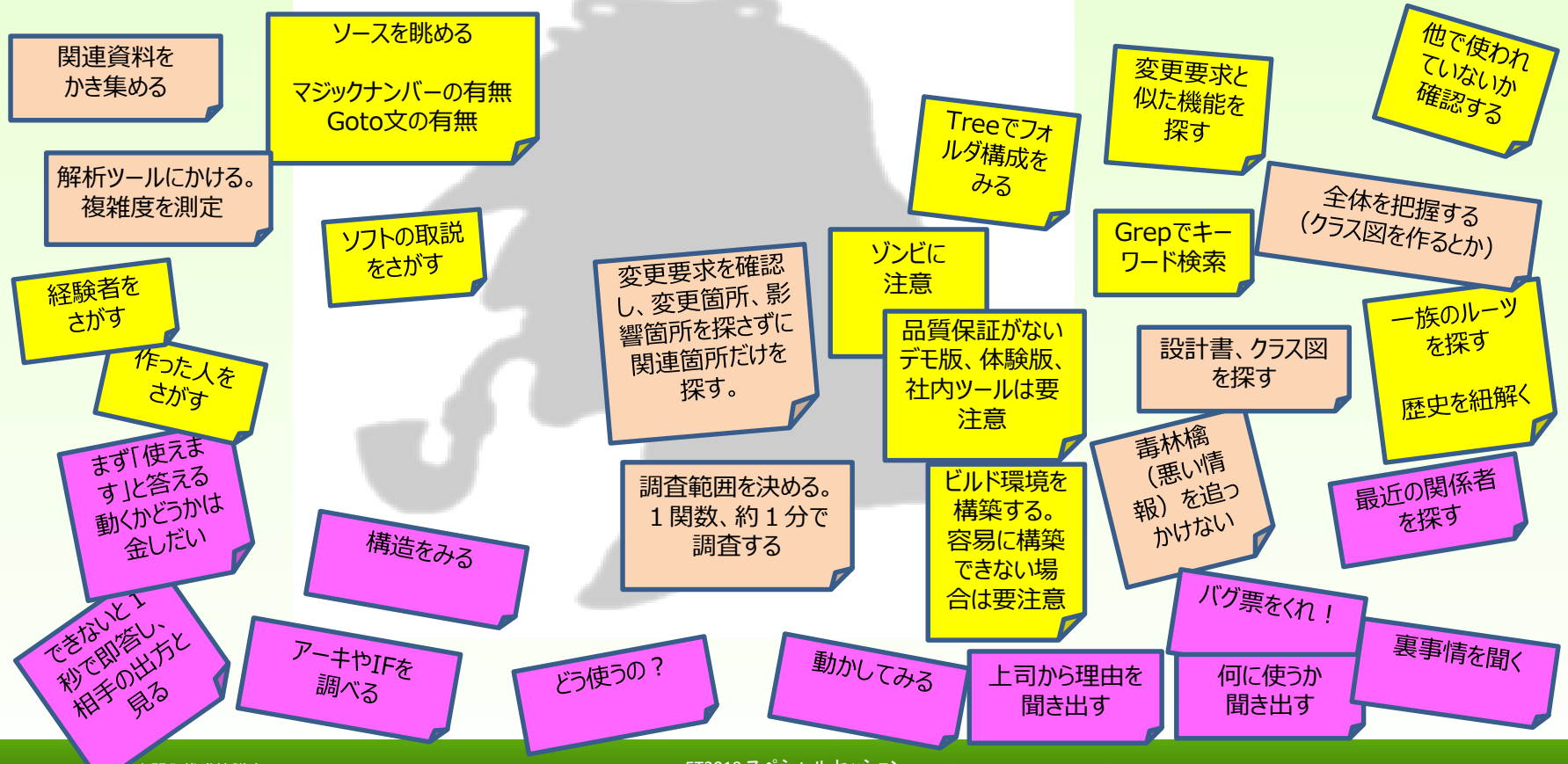
スペックアウトで何をやればいいの？

例えば、ドメイン知識が無い派生開発を担当することになったとき
まず、何をしますか？

聞いてみた

「既存のソースコード10万行を今度の製品に使おうと思っているが、使えるかどうか1週間で判断してほしい」

T19研究会メンバーの考え＋お越しいただいた方の考え



スペックアウトで何をやればいいの？

スペックアウトに関連した意見を抽出

関連資料をかき集める

全体を把握する（クラス図を作るとか）

設計書、クラス図を探す

構造をみる

アーキやIFを調べる

変更要求を確認し、変更箇所、影響箇所を探さずに関連箇所だけを探す。

設計仕様を理解したい

「設計仕様の理解」は難しい

設計仕様の理解が
重要なのはわかるけど

- ソースコードしかない
  - しかもコメントが無い、役に立たない
- ドキュメントが同期していない、信頼できない
  - 設計の意図がわからない
- 前任者が退職していて有識者がいない
  - 効果的なレビューができない

スペックアウトの実情

特別な作業ではなく、皆さん普段作業していると思いますが
しかし・・・

- スペックアウト技術や行為が俗人的、人任せ
  - スペシャルエンジニア的な人がいれば高品質
  - 担当者によって品質のバラつきが発生
- ソースコードの調査(スペックアウト時に)に時間がかかっている
  - 一度調べた関数を何度も見た
  - どこから行き着いたかわからなくなった
  - 調査結果が、全然使えなかった だからまた調査した
  - 途中で調査の目的が変わった

スペックアウトの実情

特別な作業ではなく、皆さん普段作業していると思いますが、
しかし・・・

- スペックアウト技術や行為が俗人的、人任せ
  - スペシャルエンジニア的な人がいれば高品質
  - 担当者によって品質のバラつきが発生
- ソースコードの調査(スペックアウト時に)に時間がかかっている
  - 一度調べた関数を何度も見た
  - どこから行き着いたかわからなくなった
  - 調査結果が、全然使えなかった だからまた調査した
  - 途中で調査の目的が変わった
- そもそもスペックアウトという工程を意識していない
  - 設計仕様を理解するための有益な成果物が残らない

スペックアウトを行う上で大切なこと

理解したことを示す「成果物（スペックアウト資料）」が重要

- “理解した”とはどういう状態をいうのか？
  - 自分の言葉で説明できる状態なったとき「理解した」と言える

- 頭の中はレビューできない
  ⇒レビューするためにも成果物が必要
- レビューすることで関係者間で共有する
- 今回限りの資料にしない
  ⇒今後も役立つ資料にする

スペックアウトを行う上で大切なこと

目的を明確にする

- 何のためにスペックアウトをするのか
  - 知りたい機能は？
  - 知りたいことは？
  - 必要な成果物は？
  - スペックアウトに投入できる時間は？

スペックアウトの一番の目的は
変更要求仕様におけるBeforeを明確にする

スペックアウトのやり方

Step 1

スペックアウトの準備
関連資料の整理
ソースファイルの整理

Step 2

スペックアウトと
スペックアウト資料の作成

処理構造　状態遷移　データ構造　…

Step 3

スペックアウト資料から仕様を作成

スペックアウトのやり方

Step 1

スペックアウトの準備
関連資料の整理
ソースファイルの整理

Step 2

スペック
スペックアウ

・関連資料はそろっているか？
・ソースコードと同期しているか？
・目的を明確にする

処理構造　状態遷移　データ構造　…

Step 3

スペックアウト資料から仕様を作成

スペックアウトのやり方

Step 1

スペックアウトの準備
関連資料の整理
ソースファイルの整理

Step 2

スペックアウトと
スペックアウト資料の作成

処理構造　状態遷移　データ構造　…

Step 3

スペックアウト資料から仕様を作成

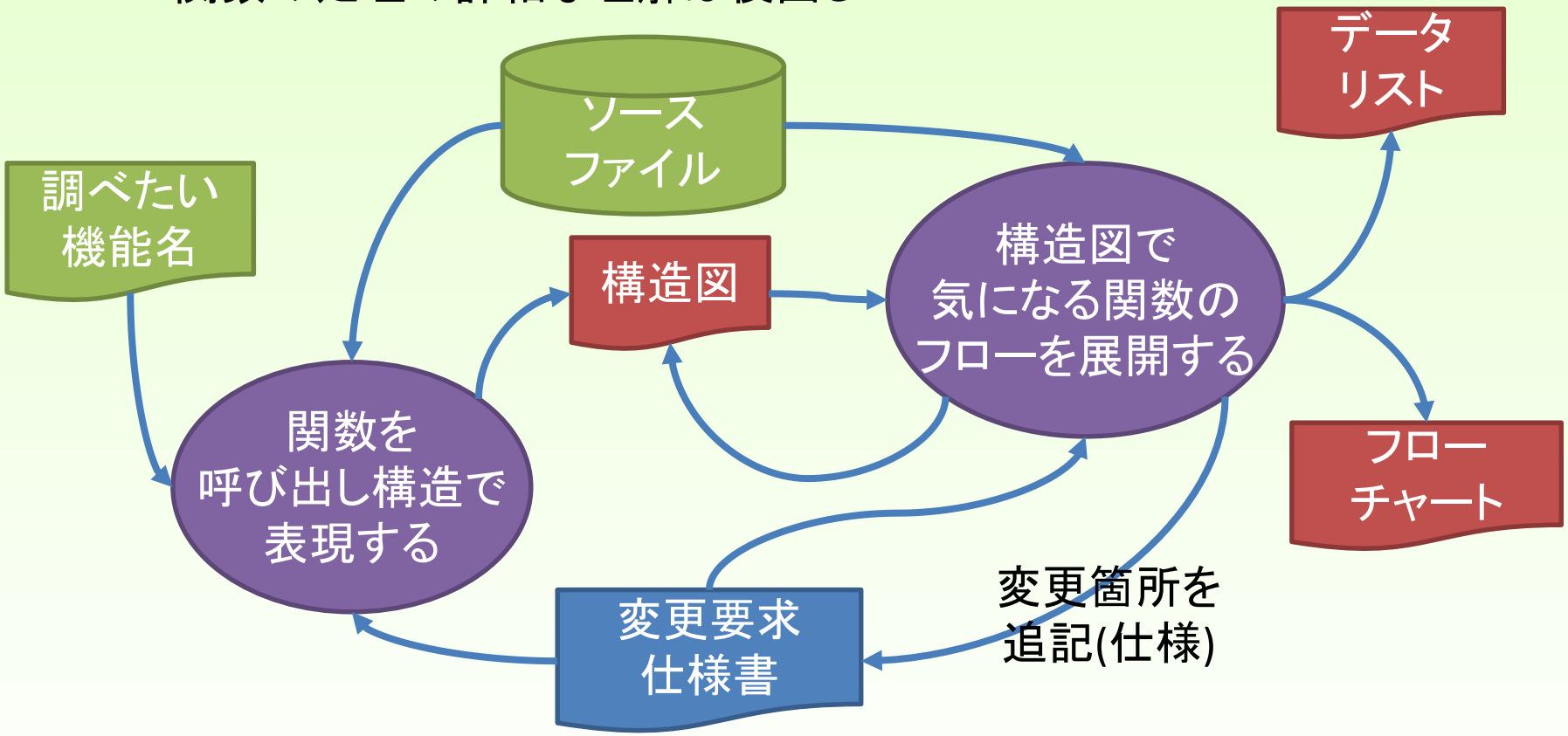
スペックアウトの種類

スペックアウトは、その「場面」（派生開発、リファクタリング、アーキテクチャー再設計など）によっていろいろな表現方法が考えられます。

- 派生開発でのスペックアウトの例
  - 処理構造のスペックアウト
    - 関数の呼び出し構造の表現（構造図、クラス図、シーケンス図など）
  - 状態遷移のスペックアウト
    - 状態遷移の表現（状態遷移マトリクス、状態遷移図）
  - データ構造のスペックアウト
    - データ構造図、ER図

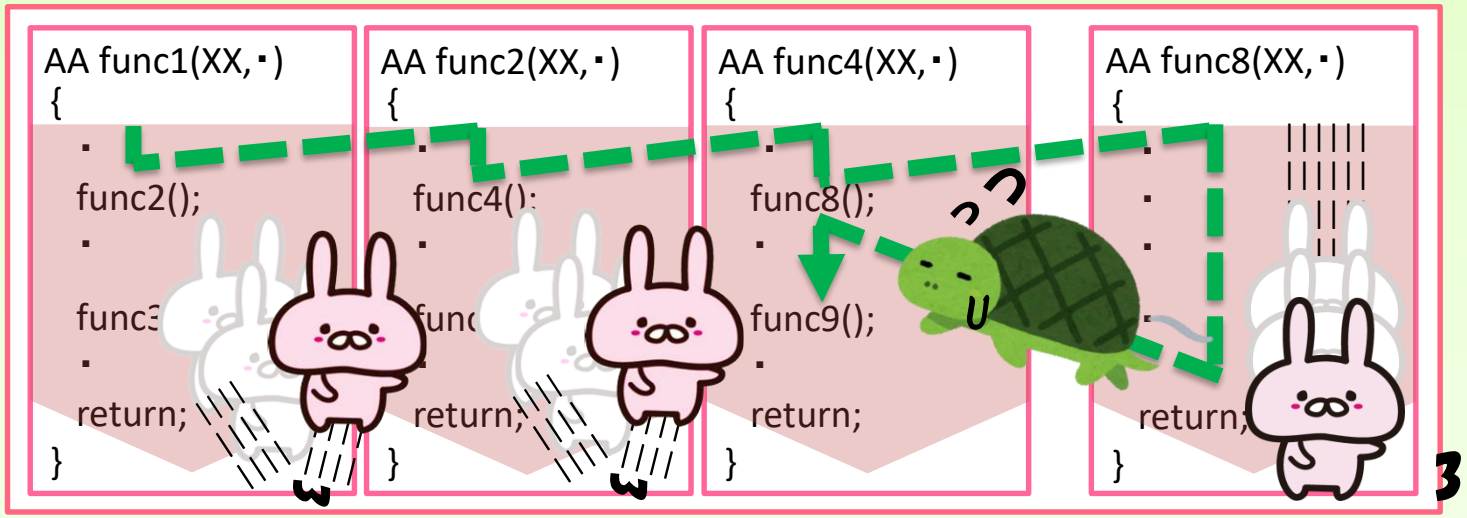
処理構造のスペックアウト

- 処理の呼び出し構造を表現する
  - 一つの関数から順次呼び出し構造を「構造図」に表現する
  - 関数の処理の詳細な理解は後回し



処理構造のスペックアウト

- 処理構造



探索ルート:コードリーディングの順番

カメの探索ルートの悪い点
- 範囲が狭い (線で捉えている)
- 目的が変わる
- 構造を把握できない

ウサギの探索ルートの良い点
- 範囲が広い(面で捉えている)
- 目的が変わらない
- 全体の構造を把握できる

成果物の一例

カメの探索ルートで作成した成果物の例

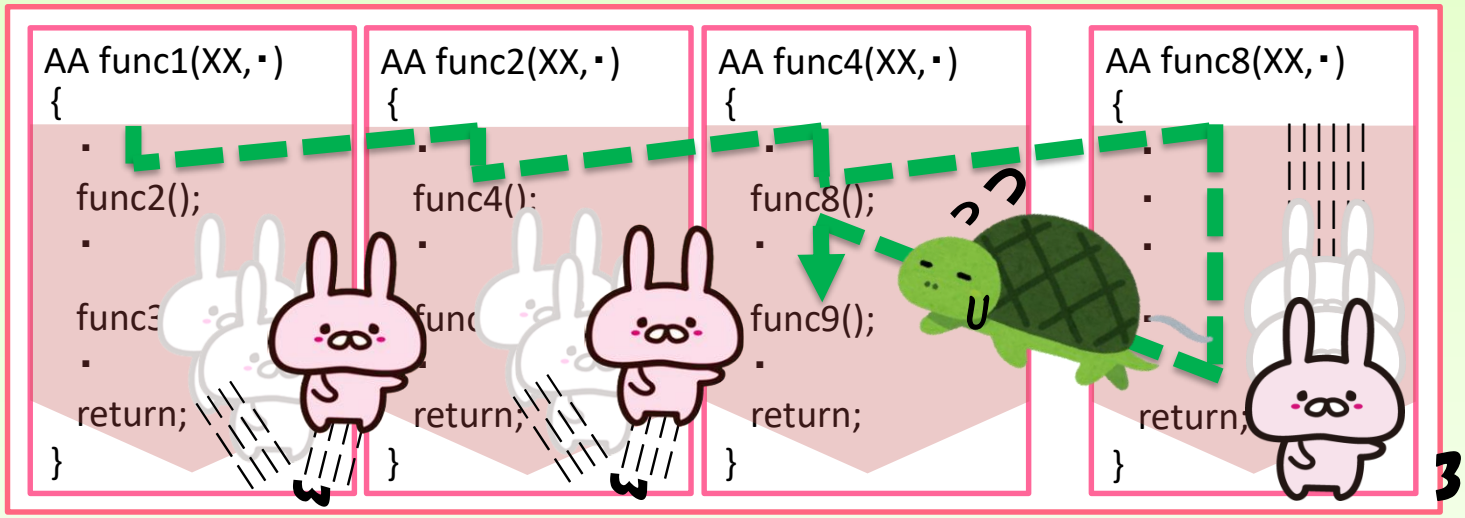
```
func1()
        // 受信データ設定
        func2()
                //次に、各大項目レベルでの値格納処理に飛ぶ
                func4()
                                func8()
                                func9()
                                .
                                .
                func5()
```

階層が深くなればなるほど見づらくなる
どこまで調査すればいいのか判断しづらい

この探索ルートもシーケンス図を作成するときには有効な場合もあり

処理構造のスペックアウト

- 処理構造



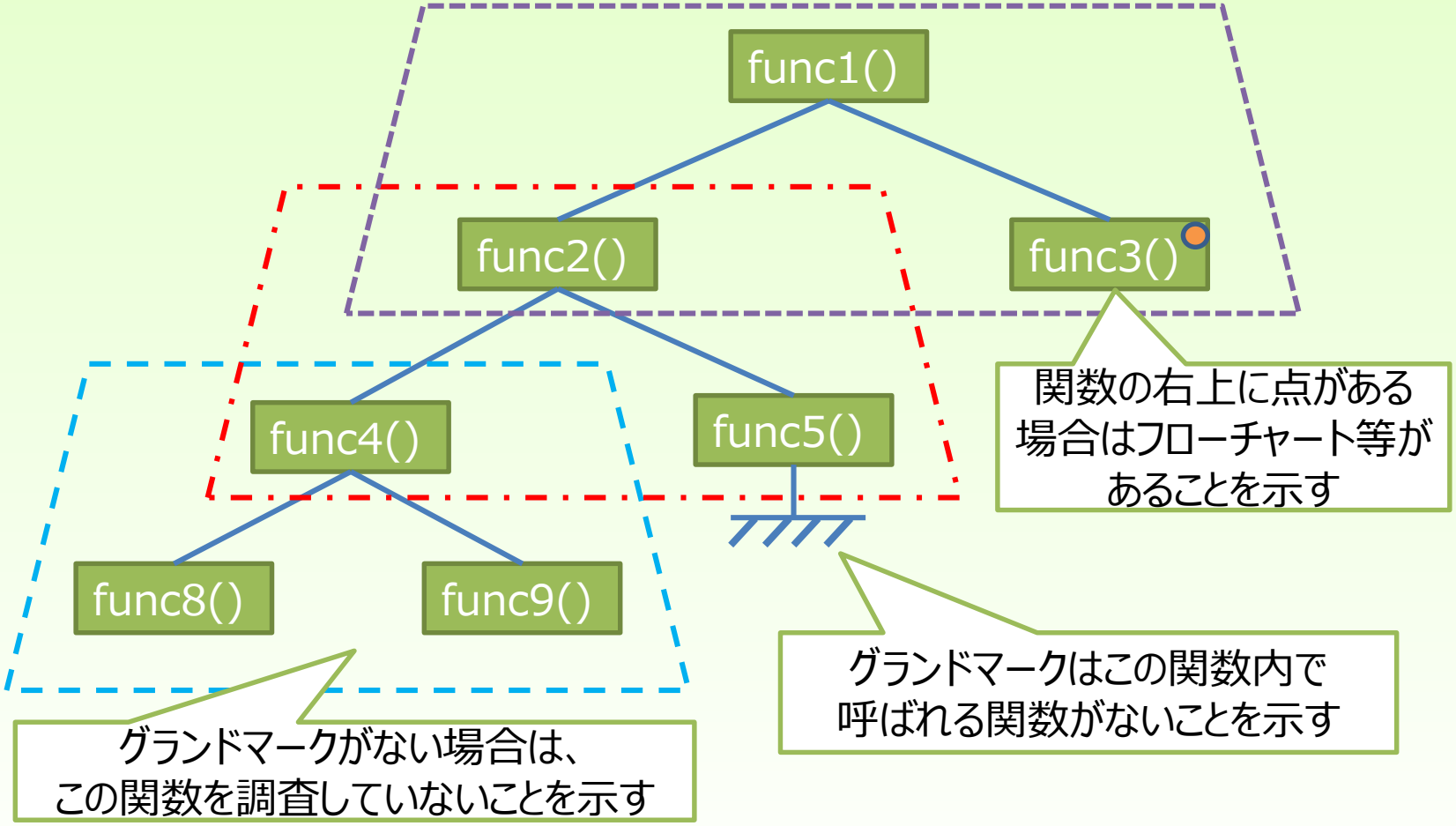
探索ルート:コードリーディングの順番

カメの探索ルートの悪い点
- 範囲が狭い (線で捉えている)
- 目的が変わる
- 構造を把握できない

ウサギの探索ルートの良い点
- 範囲が広い(面で捉えている)
- 目的が変わらない
- 全体の構造を把握できる

成果物の一例

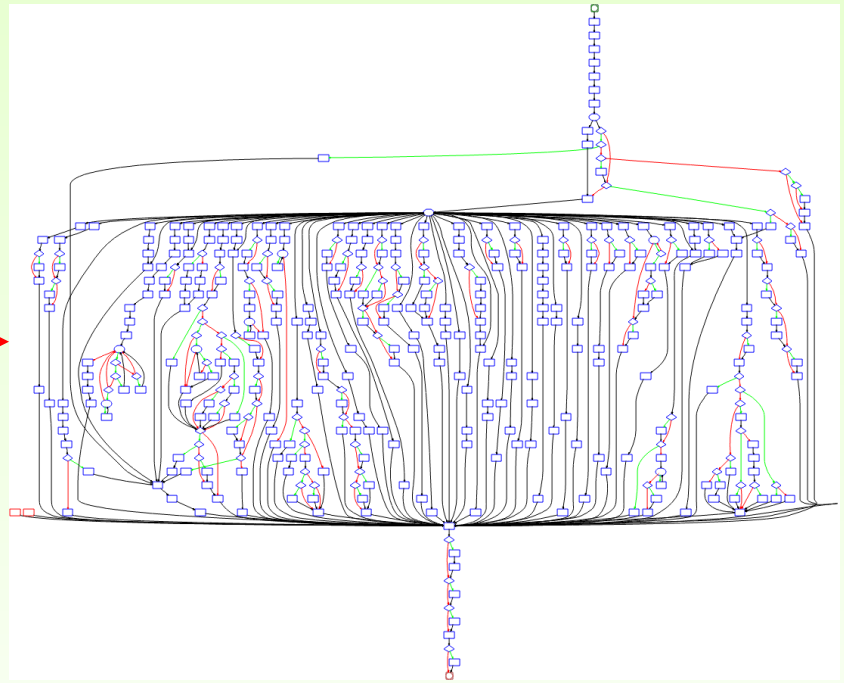
ウサギの探索ルートで作成した成果物の例(構造図)



状態遷移のスペックアウト

- ツールで起こしたフローチャートを見て、イベントに対する動作が理解できますか？

```
switch(state)
{
case WAITE:
  wsprintf(language, "↓↓手を選んで下さい↓↓");
  wsprintf(gT_word, "じゃんけん");
  SelectObject(hdc , hFont2);
  TextOut(hdc, 550, 260, gT_word, strlen(gT_word));

  SetBkMode(hdc,OPAQUE);
  SelectObject(hdc , hFont1);
  SetTextColor(hdc, RGB(255, 255, 240));//ivory
  SetBkColor(hdc , RGB(255 , 99 , 71));//tomato
  TextOut(hdc, 660, 560,language, strlen(language));
  break;
case WAITE2:
  wsprintf(language, "↓↓手を選んで下さい↓↓");
  wsprintf(gT_word, "あいこで");
  SelectObject(hdc , hFont2);
  TextOut(hdc, 550, 260, gT_word, strlen(gT_word));

  SetBkMode(hdc,OPAQUE);
  SelectObject(hdc , hFont1);
  SetTextColor(hdc, RGB(255, 255, 240));//ivory
  SetBkColor(hdc , RGB(255 , 99 , 71));//tomato
  TextOut(hdc, 660, 560,language, strlen(language));
  break;
```



ソースコード

フローチャート

状態遷移のスペックアウト

- どのような「状態」と「イベント」があるか調べる
  - 「状態」と「イベント」はソースコードから抽出

```
switch(state)
{
case WAITE:
  wsprintf(language, "↓↓手を選んで下さい↓↓");
  wsprintf(gT_word, "じゃんけん");
  SelectObject(hdc , hFont2);
  TextOut(hdc, 550, 260, gT_word, strlen(gT_word));

  SetBkMode(hdc,OPAQUE);
  SelectObject(hdc , hFont1);
  SetTextColor(hdc, RGB(255, 255, 240));//ivory
  SetBkColor(hdc , RGB(255 , 99 , 71));//tomato
  TextOut(hdc, 660, 560,language, strlen(language));
  break;
case WAITE2:
  wsprintf(language, "↓↓手を選んで下さい↓↓");
  wsprintf(gT_word, "あいこで");
  SelectObject(hdc , hFont2);
  TextOut(hdc, 550, 260, gT_word, strlen(gT_word));

  SetBkMode(hdc,OPAQUE);
  SelectObject(hdc , hFont1);
  SetTextColor(hdc, RGB(255, 255, 240));//ivory
  SetBkColor(hdc , RGB(255 , 99 , 71));//tomato
  TextOut(hdc, 660, 560,language, strlen(language));
  break;
```

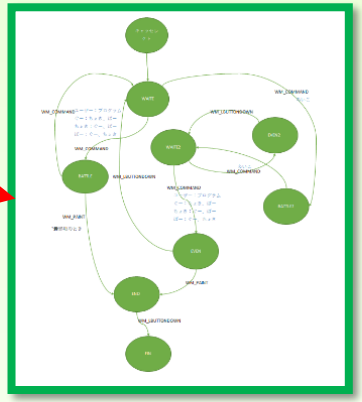
ソースコード

イベントと状態を書き出す

イベント	WM_PAINT	WM_COMMAND	WM_LBUTTONDOWN
状態			
WAITE	—	WIN Load Drawへ	—
	テキスト表示位置など	プログラムの手の判定 勝敗の記録	なし
	—	WIN Load Drawへ	—
WAITE2	テキスト表示位置など	プログラムの手の判定 勝敗の記録	なし
	ENDへ	—	WAITへ
BATTLE	テキスト表示位置など 勝敗の判定	なし	【勝敗の判定（2回2敗なら）】とキャラ・言葉の設定？
BATTLE2	—	—	WAITE2へ
	テキスト表示位置など	なし	なし
EVEN	ENDへ	—	WAITへ
	テキスト表示,描画など	なし	【勝敗の判定（2回2敗なら）】とキャラの設定？
EVEN2	勝敗の判定	—	WAITE2へ
	テキスト表示,描画など	なし	なし
END	テキスト表示,描画など	なし	FINへ
	テキスト表示,描画など	なし	なし
FIN	—	—	STAT_ENDへ(stat)
	テキスト表示位置など 処理の終了(return 0)	なし	プログラムの設定、破棄、処理の終了(return 0)

状態遷移表

イベントに対する動きを確認するため、ダイアグラムを書く



ダイアグラム

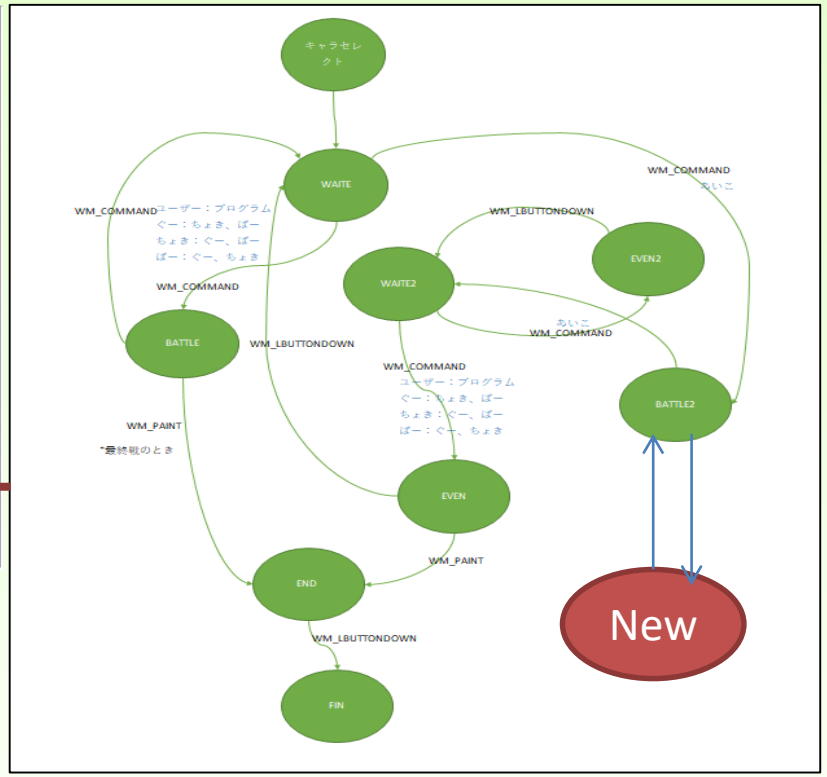
関連資料

状態遷移のスペックアウト

- 変更要求にあわせて、状態遷移図、ダイアグラムを変更する

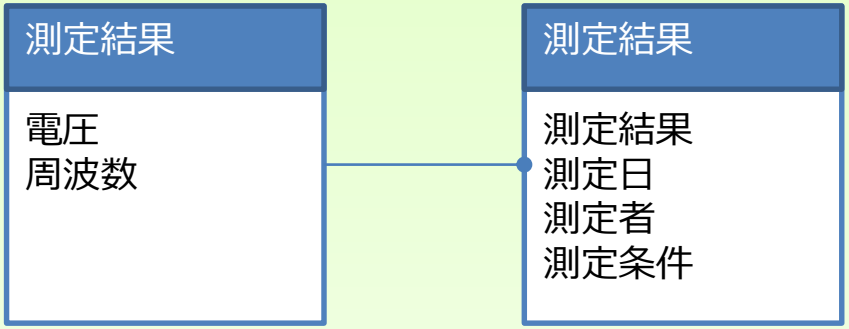
イベント	WM_PAINT	WM_COMMAND	WM_LBUTTONDOWN
状態			
WAITE	—	Win_Lose_Drawへ	—
	テキスト表示,描画など	プログラムの手の判定 勝敗数の記録	なし
WAITE2	—	Win_Lose_Drawへ	—
	テキスト表示,描画など	プログラムの手の判定 勝敗数の記録	なし
BATTLE	ENDへ	—	WAITEへ
	テキスト表示,描画など 勝敗の判定	なし	勝敗数の判定（2勝2敗なら）とキャラ・音楽の変更？
BATTLE2	—	—	WAITE2へ
	テキスト表示,描画など	なし	なし
EVEN	ENDへ	—	WAITEへ
	テキスト表示,描画など 勝敗の判定	なし	勝敗数の判定（2勝2敗なら）とキャラの変更？
EVEN2	—	—	WAITE2へ
	テキスト表示,描画など	なし	なし
END	—	—	FINへ
	テキスト表示,描画など	なし	なし
FIN	—	—	STAT_ENDへ(stat)
	テキスト表示,描画など 処理の終了(return 0)	なし	フラグの設定、破棄、処理の終了(return 0)

新たな状態を追加



データ構造のスペックアウト

- データ構造図、ER図



- 2019年のT19の活動テーマ

スペックアウトのやり方

Step 1

スペックアウトの準備
関連資料の整理
ソースファイルの整理

Step 2

スペックアウトと
スペックアウト資料の作成

処理構造　状態遷移　データ構造　…

Step 3

スペックアウト資料から仕様を作成

スペックアウト資料から仕様の作成

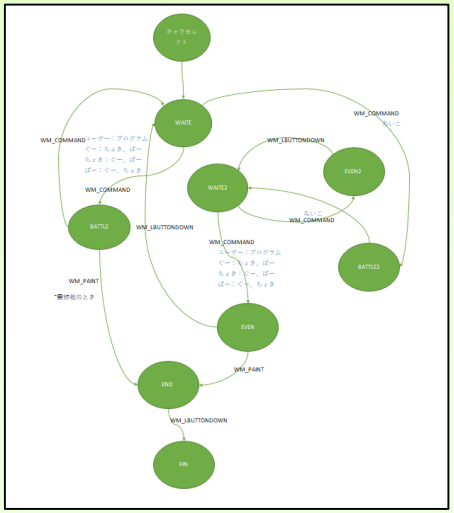
スペックアウト資料

```
switch(state)
{
case WAITE:
  wsprintf(language, "↓↓手を選んで下さい↓↓");
  wsprintf(gT_word, "じゃんけん");
  SelectObject(hdc , hFont2);
  TextOut(hdc, 550, 260, gT_word, strlen(gT_word));

  SetBkMode(hdc,OPAQUE);
  SelectObject(hdc , hFont1);
  SetTextColor(hdc, RGB(255, 255, 240));//ivory
  SetBkColor(hdc , RGB(255 , 99 , 71));//tomato
  TextOut(hdc, 660, 560,language, strlen(language));
  break;
case WAITE2:
  wsprintf(language, "↓↓手を選んで下さい↓↓");
  wsprintf(gT_word, "あいこで");
  SelectObject(hdc , hFont2);
  TextOut(hdc, 550, 260, gT_word, strlen(gT_word));

  SetBkMode(hdc,OPAQUE);
  SelectObject(hdc , hFont1);
  SetTextColor(hdc, RGB(255, 255, 240));//ivory
  SetBkColor(hdc , RGB(255 , 99 , 71));//tomato
  TextOut(hdc, 660, 560,language, strlen(language));
  break;
```

ソースコード

イベント / 状態	WM_PAINT	WM_COMMAND	WM_LBUTTONDOWN
WAITE	― / テキスト表示,描画など	Win_Lose_Drawへ / プログラムの手の作成 / 勝敗数の加算	― / なし
WAITE2	― / テキスト表示,描画など	Win_Lose_Drawへ / プログラムの手の作成 / 勝敗数の加算	― / なし
BATTLE	END へ / テキスト表示,描画など / 勝敗の判定	― / なし	WAITEへ / 勝敗数の判定（2勝2敗なら）とキャラ・背景の変更？
BATTLE2	― / テキスト表示,描画など	― / なし	WAITE2へ / なし
EVEN	END へ / テキスト表示,描画など / 勝敗の判定	― / なし	WAITEへ / 勝敗数の判定（2勝2敗なら）とキャラの変更？
EVEN2	― / テキスト表示,描画など	― / なし	WAITE2へ / なし
END	テキスト表示,描画など	なし	FINへ / なし
FIN	テキスト表示,描画など / 処理の終了(return 0)	なし	STAT_ENDへ(stat) / プログラムの終了、後始末、処理の終了(return 0)

状態遷移表



ダイアグラム

USDM

要求	NEK	本戦において2勝2敗になると出てくるボスの出現条件をユーザが2勝した時点で出現するように変更したい	
	理由	ユーザが2勝しないと出現しないため、1勝4敗の場合、ボスが出現せずにゲームが終わってしまうため	
	要求	NEK01	表示キャラクター、背景をボス（ねこまっちょ）仕様にする条件を、ユーザが2勝かつ2敗の場合という条件から、ユーザが2勝の場合に変更する。ボス表示に移行させているのは、状態"BATTLE"と"EVEN"の2か所から移行しているため、両方の条件を変更する必要がある。
	理由	NEK01	ボスに切り替える条件は、2勝かつ2敗の場合のみとなっている。これを2勝の場合とすることで変更要求を満たすことが出来るため
		<ボス切り替え条件>	
	□□□	NEK01-1	状態"BATTLE"時、どのキャラクター表示をするか管理している変数char_numへボス(3)を設定している個所の条件分岐からユーザの敗北数(gT_defeat)の条件を削除する
	□□□	NEK01-2	状態"EVEN"時、どのキャラクター表示をするか管理している変数char_numへボス(3)を設定している個所の条件分岐からユーザの敗北数(gT_defeat)の条件を削除する

スペックアウトの効果

- 設計仕様の理解
  - ソースコードを調査することで、設計書では漏れていることが多い
    準正常系や異常系も抽出可能
  - 現状の設計のままでは問題が発生する可能性に気づくことも
    ⇒リファクタリング、アーキテクチャの再構築
- 変更箇所・影響範囲の特定
- 設計技術の習得
  - 派生開発でやみくもにソースを変更してるだけでは身につかないが、
    スペックアウトで作成する成果物は設計技術そのもの
  - ソースコードを図にしたときの関係が見えるので設計の落としどころが分かる
  - 他人のソースコードを読むことで経験を積める

まとめ

- スペックアウトは目的と成果物が重要
  - 目的に応じた成果物を作る
  - 次の工程のインプットとなる成果物を作る

- スペックアウトは、派生開発の現場の混乱を
  これ以上広げないための守りの技術

最後に

スペックアウト技術を手に入れたら・・・

リファクタリングやアーキテクチャの再構築といった次のステップに進むことができるようになります