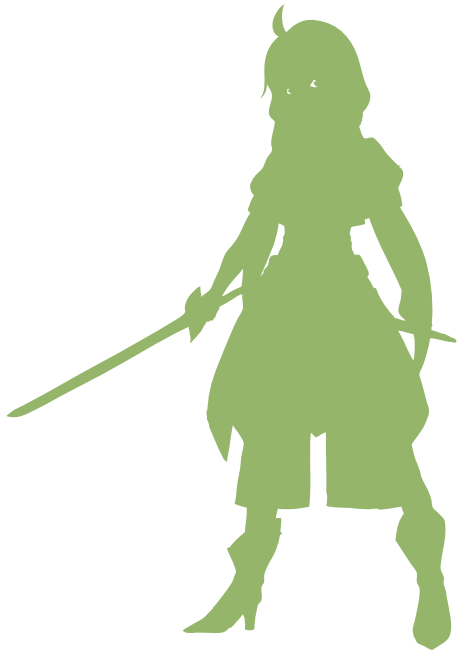
進化したDelphi/C++ RTLの活用！
既存コードの見直すべき10の理由

第35回エンバカデロ・デベロッパーキャンプ

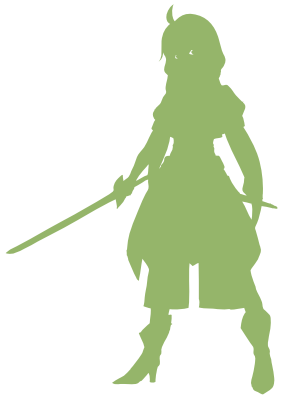


エンバカデロ・テクノロジーズ
セールスコンサルタント 筑木 真志

embarcadero®
DEVELOPER CAMP

“

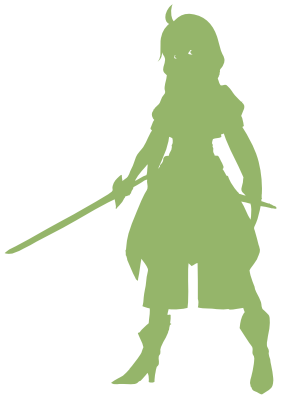
今使っているDelphi/C++Builderの機能で十分だから
アップグレードは不要です。



embarcadero®
DEVELOPER CAMP

本当にそうですか？

何故、「コードの見直し」が必要か？



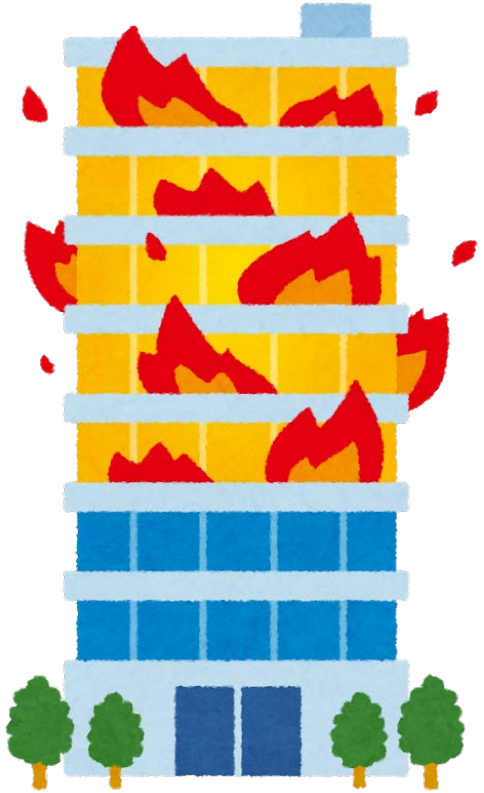
embarcadero®
DEVELOPER CAMP

とある、「モダナイズ」案件

- VB製C/Sシステムの Web化
  - 旧システムの設計書無し
  - ソースがバイナリより古い。バイナリの挙動とDBから
    リバースエンジニアリング
- Java + Swing製C/Sシステムの「書き直し」＆機能追加
  - 旧システムの設計書無し
  - 旧システムの挙動とDBからリバースエンジニアリングして設計書を作成
  - 作成した設計書を元にC/Sシステムの再構築と機能追加
- とあるWebシステムの機能追加
  - 設計書とソースコードが乖離
  - ソースはVBかDelphiから機械的にコンバージョンした形跡あり
  - コメント無し

共通しているのは...

- 設計書と既存コードの見直しをしていなかった
- システムを俯瞰できる人たちがいなかった



結果的に
炎上プロジェクト化

システムは「生物」です

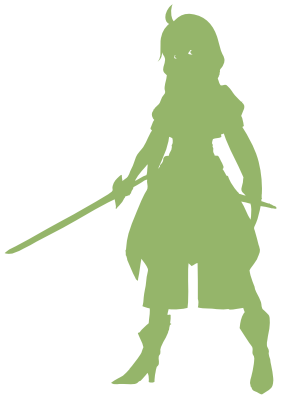
- メンテナンス~~されない~~出来ないシステム＝システムの「死」
  - トラック係数/バス係数
    - メンバーのうち何人事故に遭ったらプロジェクトが成立しなくなるかを表す指標
  - 「秘伝のソースコード」、「一子相伝、口伝のソースコード」
    - そのソースコード、永久にメンテナンスし続けられますか？
- システムとドキュメントの乖離はありませんか？
  - 少なくとも、コメントと差異はないですよね？
- 過去に執着し続ける「○○おじさん」になっていませんか？
  - セキュリティ対策、OSのアップグレードに対応出来ますか？

既存システムのモダナイズ

- ○○のシステム、モバイル対応出来ない？
- ○○のシステム、Web対応出来ない？
- ○○のシステム、Windows10対応出来ない？

  - 「誰もメンテナンス出来ないので判りません」でいいのですか？
  - 「疎通レベルでは動いています」でいいのですか？

既存コードの見直すべき10の理由



embarcadero®
DEVELOPER CAMP

理由その1：属人性の高いコードは改修コストも高くなる

- もし、属人性の高いコードを放っておくと…
  - 誰もメンテナンス出来ない「秘伝のコード」
  - バグが出ても直せない
  - 機能追加も出来ない

どうするか？
- コードレビューなどでのナレッジの共有

その結果…
- チーム全体のスキル向上
- コードそのものの品質向上
- 誰でもメンテナンス出来るコードに

理由その2：可読性が悪いソースはバグの温床

- ごちゃごちゃした「スパゲティコード」はバグの温床
  - DelphiやC++は定義と実装の位置が離れがち
    - 例えば、メソッドの中で１回しか使われないコールバックとか…
- C++の場合、コンテナをforループで列挙すると定義が長くなる
  - コンテナの型が変わった場合どうしますか？

```
for (std::vector<hoge>::iterator iter = vec.begin(); iter != vec.end(); ++iter) {
    // 何らかの処理
}
```

どうするか？
➢ リファクタリングの実施

リファクタリングの実施例①

- コールバックの無名メソッド化
  - 無名メソッド（匿名メソッド）を使用して宣言を無くす

```
tasks[0] := TTask.Create (procedure ()
begin
  FDTable1.Open();
  while FDTable1.Eof = false do
  begin
    TInterlocked.Increment(TotalRead);
    FDTable1.Next;
  end;
  FDTable1.Close();
end);
tasks[0].Start;
```

リファクタリングの実施例②

- C++11で導入された型推論、範囲forを使用して、無駄な宣言をコンパイラにお任せする。

```
__fastcall TForm1::TForm1(TComponent* Owner) : TForm(Owner)
{
  cboEra->Items->Clear();

  // 元号一覧の取得
  TFormatSettings LocaleSettings = TFormatSettings::Create(TLanguages::UserDefaultLocale);

  for (auto era : LocaleSettings.EraInfo) {
    cboEra->Items->Add(era.EraName);
  }
}
```

理由その3：非推奨機能は将来OSでの動作を保証しない

- 非推奨機能はOSのバージョンアップに追随出来ません
  - セキュリティホールの原因になる可能性も
  - バージョンアップで突然動かなく場合もあります
  - http://docwiki.embarcadero.com/Libraries/Tokyo/ja/カテゴリ:非推奨

どうするか？
➢ 代替機能で置換する

理由その4：UXを向上させよう！

- 扱うべきデータ量は増える一方
  一覧表示とか遅くないですか？
  - 長いループだと、「応答無し」になる

- Application.ProcessMessagesでループ待ちしていませんか？

- 長い待ち時間はUXの低下をもたらします

どうするか？
➢ 並列処理、非同期処理で待ち時間を減らす

並列処理、非同期処理の例

- 並列処理、非同期処理で待ち時間を減らすことが出来ます

- TTaskはTThreadより手軽に並列処理を実現できます
  - TThreadを継承したクラスを作成する必要はありません

理由その5：RTLの新機能でコードの可読性を向上する

- たとえば、みんな大好きTStrings/TStringList
  - SetStrings
    - TStringsをコピーする
  - KeyNames
    - キーのリストを取得する
  - AddPair
    - 「数珠つなぎ」っぽいスタイルで文字列リストにKey-Value形式で文字列を追加する

その結果…
- ➢ 新機能でコードが書きやすくなります
- ➢ コード量の低下＝可読性の向上

理由その6：汎用的な自作機能は思わぬバグの元

- 自作できそうな機能の例
  - ハッシュ値
  - エンコード、デコード
  - ファイルの圧縮、解凍
  - XMLやJSONのI/O
  - など
- 汎用的なものは自作しない。十分にテストされていないので思わぬバグの原因になる。

どうするか？
- RTLに用意してあります。
- 場合によっては、OSSの活用

XE8以降で追加された便利なクラス色々

クラス	機能
THashMD5	MD5 ハッシュ
THashSHA1	SHA-1 ハッシュ
THashSHA2	SHA-2 ハッシュ
THashBobJenkins	Jenkins ハッシュ
TURI	URLのエンコード／デコード、国際化ドメイン名、URIのパースなど
TJsonTextReader TJsonTextWriter	JSON形式のシリアライズI/O
TBsonReader TBsonWriter	BSON(バイナリ型JSON)形式のシリアライズI/O

理由その7：型とロジックの「融着」を排除して保守性を向上

- クラスは違えど、ロジックは「同じ」だからとコピペで済ませていませんか？
  - 思わぬ所でコピペミスとか
- TStringListを連想配列の代わりに使っていませんか？
  - TStringListの「連想配列」はパフォーマンスが良くありません

どうするか？

➢ ジェネリックスコレクションを使って型とロジックを分離

➢ 共通化出来る部分は共通化する

ジェネリックスコレクションを使ったロジック

- ジェネリックスコレクションを使って型とロジックを分離
- C++Builderユーザーの皆様はテンプレートやSTLでおなじみです

機能	ジェネリックスコレクション	STL(C++)
配列	TArray	std::vector
連想配列	TDictionary、TObjectDictionary	std::map/std::hashmap
リスト	TList、TObjectList	std::list
キュー（FIFO）	TQueue、TObjectQueue	std::queue
スタック(LIFO)	TStack、TObjectStack	std::stack
値のペア	TPair	std::pair/std::tuple

理由その8：OSの差異に右往左往してはならない

- OSの仕様に依存したパス名の処理はバグの温床
  - EndsTextで拡張子の判定
  - PosEx関数でドライブレターやフォルダ名の分離
- ドキュメントフォルダ、ダウンロードフォルダの取得
  - パスを決め打ちしていませんか？
- UNCパス、Linux、macOSなどどうしますか？

どうするか？
- OSに依存した処理は排除する
- パス名処理関数の使用（TPath、TDirectory、TFile）

理由その9：文字列の扱いは注意が必要

- String型の違い
  - Delphi/C++Builder 2007以前：AnsiString
  - Delphi/C++Builder 2009以降：UnicodeString

- マルチバイト時代を基準にした文字列処理はNG
  - 文字コード、文字列リテラルを直書きで判定
  - 文字列のインデックス
    - デスクトップ：1始まり
    - モバイル：0始まり

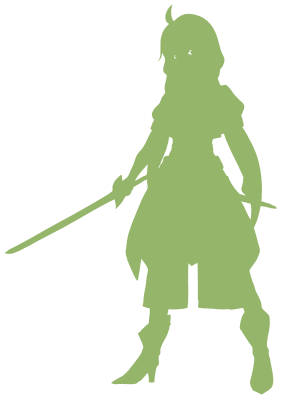
どうするか？
TCharHelperを使った文字種判定
TStringHelper を使用

理由その10：独自の年月処理は改元対応の妨げ

- 独自に年月処理とか実装していませんか？
  - 特に、和暦から西暦の変換
- 改元とは直接関係無いですが…
  - 標準時（GMT）と現地時間の変換、夏時間期間の考慮

どうするか？
RTLの日付処理ルーチンの使用
- 日付型から和暦への変換
  DateTimeToString/FormatDateTime
- 和暦から西暦への変換
  StrToDate/DateTimeToStr

まとめ



embarcadero®
DEVELOPER CAMP

Delphi/C++Builderも日々進化しています

- RTLの強化により今まで手の届かなかった「痒い」部分も掻けます
- VCL／FMXも同様です
- DocWikiの「新機能」を是非チェックしてください

持続可能なシステムであるために

- 人間が体や脳の定期的なメンテナンスやリフレッシュが必要であるのと同様、システムも定期的なメンテナンスやリフレッシュが必要です

- 新しい運用形態への対応やリスクの軽減の為に定期的なコードの見直しが必要となります

THANKS!

www.embarcadero.com/jp

第35回 エンバカデロ・デベロッパーキャンプ