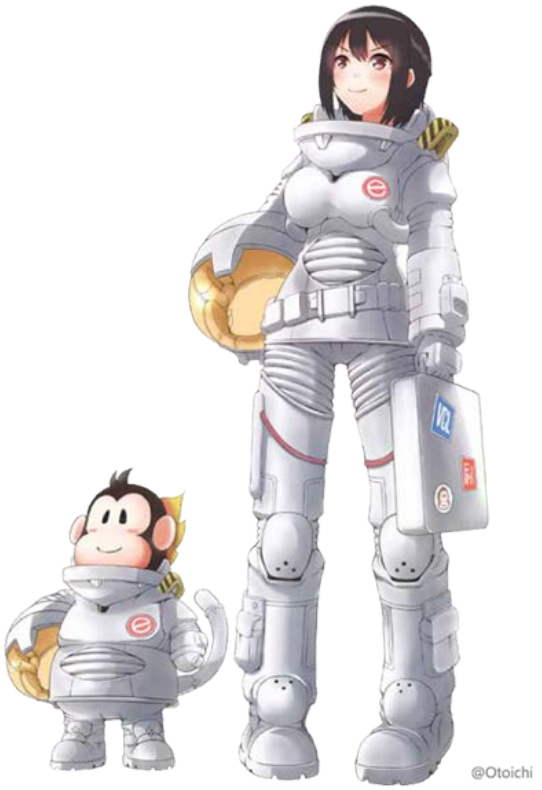
プロジェクトが大きくなっても慌てない！
RAD Studioチーム開発の心得

第34回 エンバカデロ・デベロッパーキャンプ

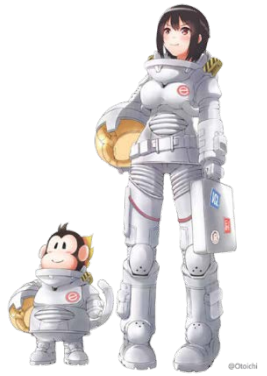


エンバカデロ・テクノロジーズ

embarcadero®
DEVELOPER CAMP

はじめに

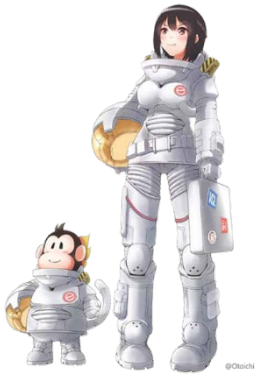
- 小規模からの開発が可能なRAD技術。とはいえ、ビジネス環境の変化に伴いシステムは巨大化していくものです。プロジェクトが大きくなるにつれ、開発手法もプロジェクトの構成も変更が必要。チーム開発の手法とその実例を紹介します。チームでのソース管理やビルド方法、ユニットのテスト方法からアジャイル開発手法など、Delphi/C++Builderを用いた実例を交えて解説します。後半は、アトラシアンの長沢氏をゲストスピーカーに迎え、チーム開発の実践法を解説してもらいます。



embarcadero DEVELOPER CAMP

Delphi / C++Builder 小規模からの開発

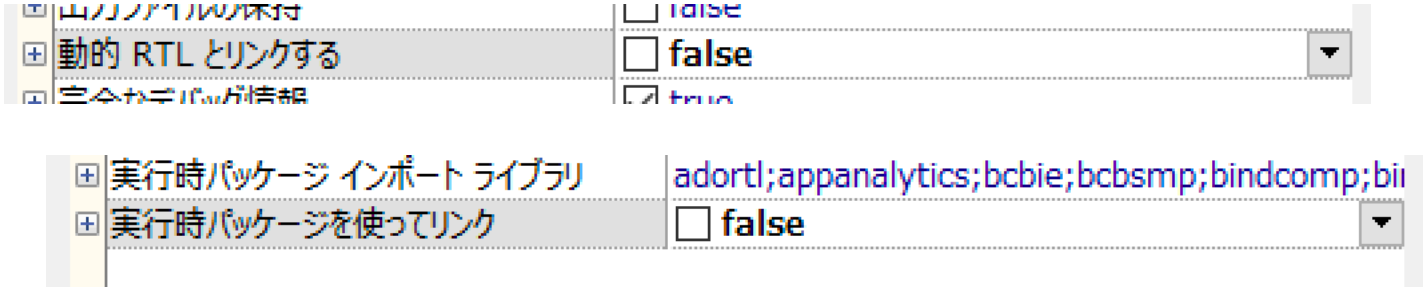
- さくっと作ってさくっと納品



embarcadero DEVELOPER CAMP

Delphi / C++Builder 小規模からの開発

- Delphi / C++Builderの 小規模システムのメリット
  - EXEファイルのコピーでインストールやアップデートが完了する



Delphi / C++Builder 環境の変化に伴いシステムは巨大化

- 1つのEXEファイルでなんとかなっていたシステムは
  環境の変化や時経て巨大化していきます。

320K Byte(フォームのみ)

2M Byte(フォームのみ)
90K Byte (実行時パッケージ=true)

- デバッグ情報を含めたEXEファイルが時には数百メガに。。
  - 沢山のユニットファイルリソースなど小規模と同じようにビルド

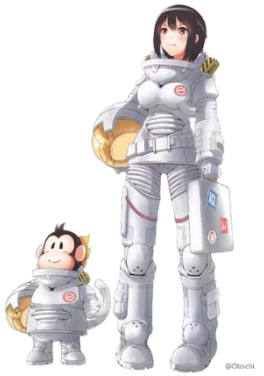
100M Byte

巨大化した1つのEXEファイルのデメリット

- ビルド時間 / デバッグ実行時間
- ロードも遅い
- コピーも遅い
- アップデートが失敗する率

モジュール単位での開発

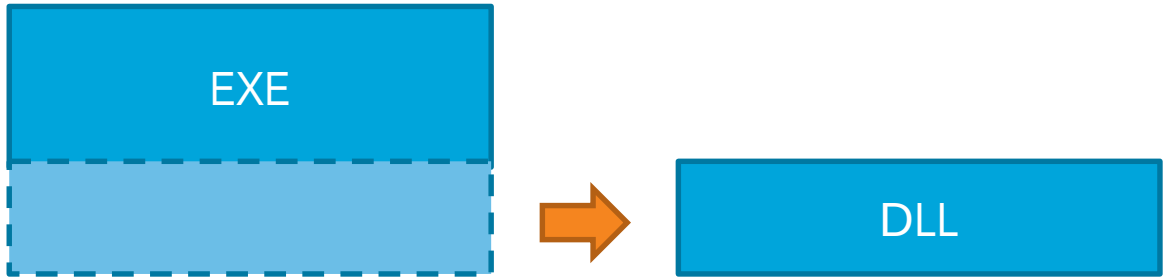
- 共通機能の単体テスト効率、精度の向上



embarcadero DEVELOPER CAMP

アプリケーションサイズ肥大化を抑止する策

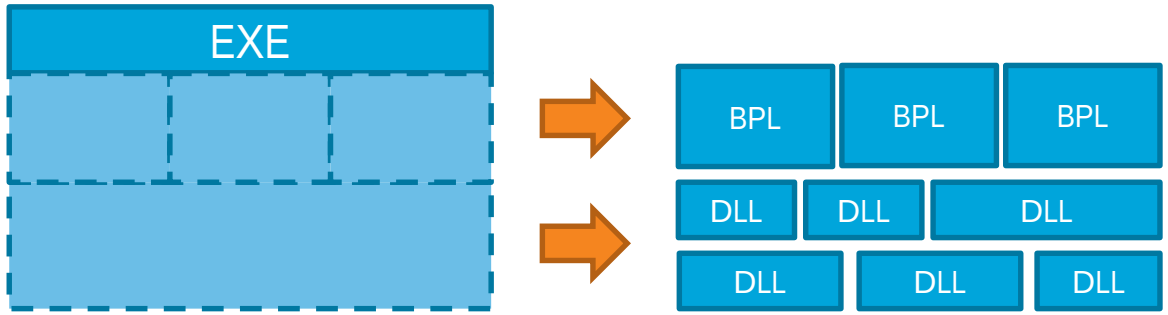
- 例. 現在実施中の方式



一部モジュールのDLL化

効果は限定的

- 推奨する実施方式



アプリケーション機能のパッケージ化

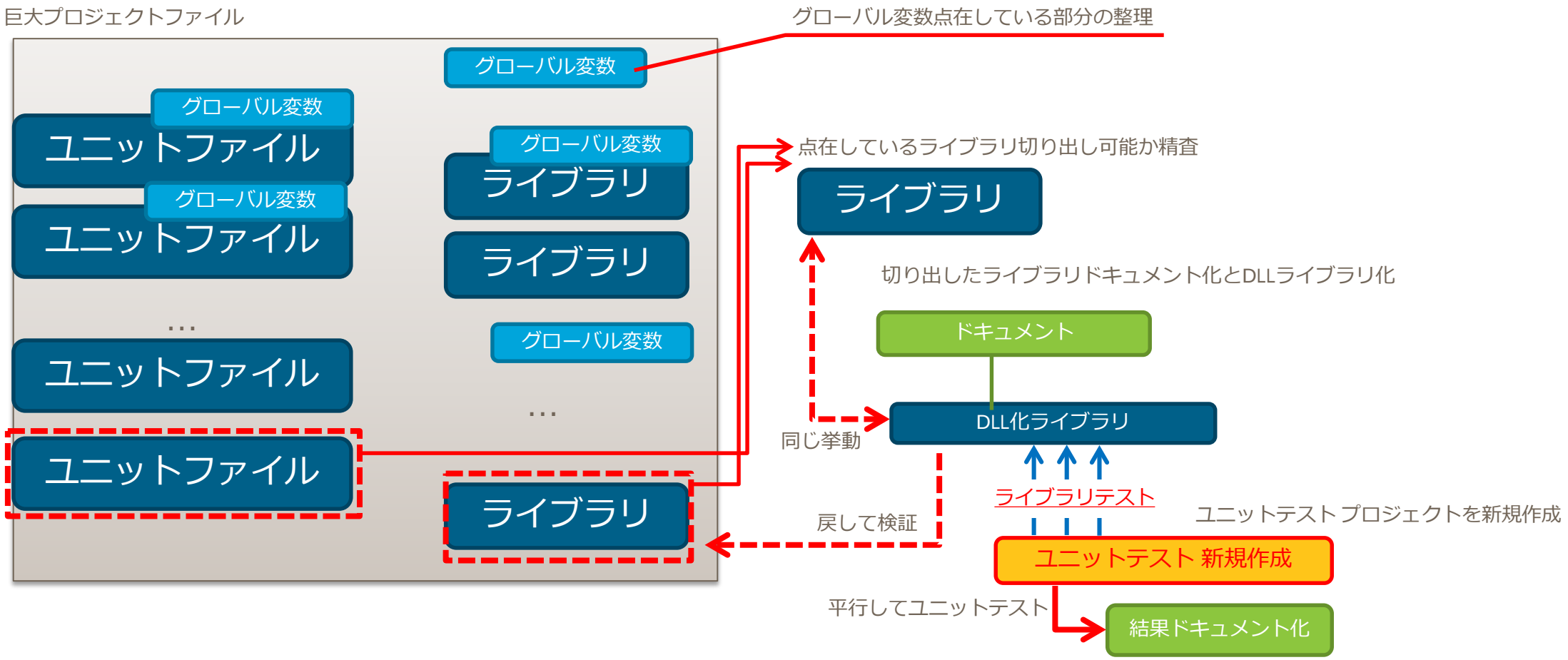
共通モジュールのDLL化

(DLL側関数への速度が必要な場合、利用方法に注意)

今後の肥大化に対しても十分対策可能

モジュール切出作業

- さまざまな方法でライブラリ化してくのですが、その中の１つ

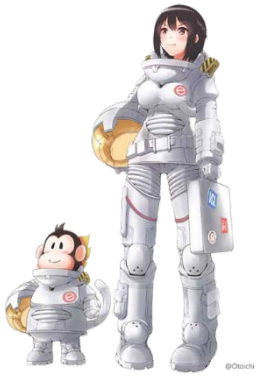


コードカバレッジ

- 文網羅
  - ソースコードの各文がテストで実行されたかどうかで判断する。
- 分岐網羅
  - 制御構造上の分岐でそれぞれの分岐方向がテストされたかどうかで判断する。
- 条件網羅
  - 分岐条件の各項でtrueとfalseの両方がテストされたかどうかで判断する。
- 経路網羅
  - 対象コードの考えられる全ての経路についてテストで実行されたかどうかで判断する。
- 入口/出口網羅
  - 存在する全ての関数呼び出しがテストで実行されたかどうかで判断する。

共通機能の単体テスト効率、精度の向上

- 共通機能を切り出してDLL化することで、機能仕様に合致するかどうかに集中した単体テストが可能となります



embarcadero DEVELOPER CAMP

モジュール単位での開発、テストの効率化／最適化

- RAD Studio に組み込まれているユニット テスト(DUnitX)機能
  - DUnitX は、NUnit テスト フレームワークに基づくオープンソースのユニット テスト フレームワーク
  - xUnit のアイデアも一部取り入れられています。

  - DUnitX フレームワークの RAD Studio インテグレーションにより、Delphi / C++Builder アプリケーションを、Win32、Win 64、OS X、Linux に対して開発しテストを実行することができます。

  - DUnitX テスト フレームワークには、さまざまな条件をテストするための一連のメソッドが独自に用意されています。
  - 用意されているメソッドを使用して、多数の条件をテストできます。
  - これらのメソッドは一般的なアサーションを表していますが、独自のカスタム アサーションを作成することもできます。

DUnitX でのテストの作成

- ユニット テストの構造は、テスト対象となるクラスやメソッドの機能によって大きく異なります。
- ユニット テスト ウィザードで、テスト プロジェクトのスケルトン テンプレート、setup および teardown メソッド、基本的なテスト ケースが生成されます。
- 特定のメソッドをテストするため固有のテスト ロジックを追加できます。

DUnitX の関数

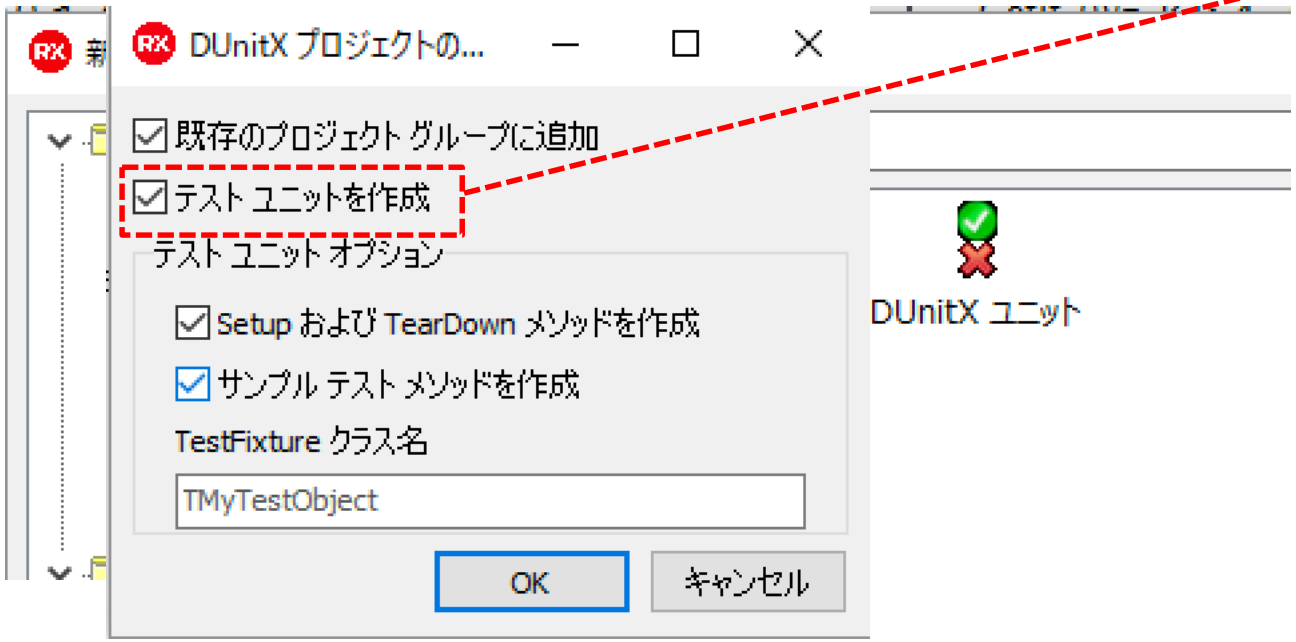
- テストで使用できる多数の関数を備えた Assert クラスが用意されています。

関数	説明
Pass	ルーチンが正常に機能することを確認します。
Fail	ルーチンが正常に機能しないことを確認します。
AreEqual	項目が等しいことを確認します。
AreNotEqual	項目が等しくないことを確認します。
AreSame	2 つの項目が同じ値であることを確認します。
AreNotSame	2 つの項目が同じ値でないことを確認します。
Contains	項目がリストに含まれていることを確認します。
DoesNotContain	項目がリストに含まれていないことを確認します。
IsTrue	条件が成立することを確認します。
IsFalse	条件が成立しないことを確認します。
IsEmpty	項目の値が空であることを確認します。
IsNotEmpty	項目の値が空でないことを確認します。
IsNull	項目が null であることを確認します。

IsNotNull, WillRaise, StartsWith, InheritsFrom, IsMatch

DUnitX 使い方

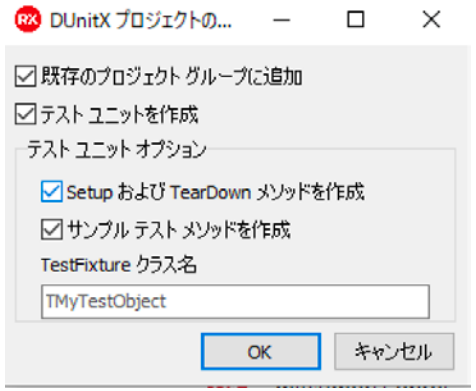
- DUnitXプロジェクトを作成



Unit1が作られます
必要なメソッドにコードを追加

DUnitX テスト ユニット テンプレート

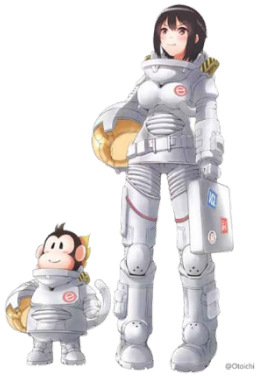
- テスト ユニット テンプレートを構成
  - ［テスト ユニット オプション］を設定



項目	詳細
Setup および TearDown メソッドを作成	SetUp メソッドと TearDown メソッドの宣言および空の定義がテスト ユニット テンプレートに追加
サンプル テスト メソッドを作成	サンプル テスト メソッド（Test1 および Test2）の宣言と空の定義がテスト ユニット テンプレートに追加
TestFixture クラス名	クラス名を入力します。デフォルトの名前は TMyTestObject です

アプリケーション機能ごとの開発
組み上げが容易に

- 機能ドリブンでの開発が可能となる上、プラグイン化により、最終組み上げ段階でのビルドエラー発生のリスクを大幅に回避できます



embarcadero DEVELOPER CAMP

静的ライブラリ DLL

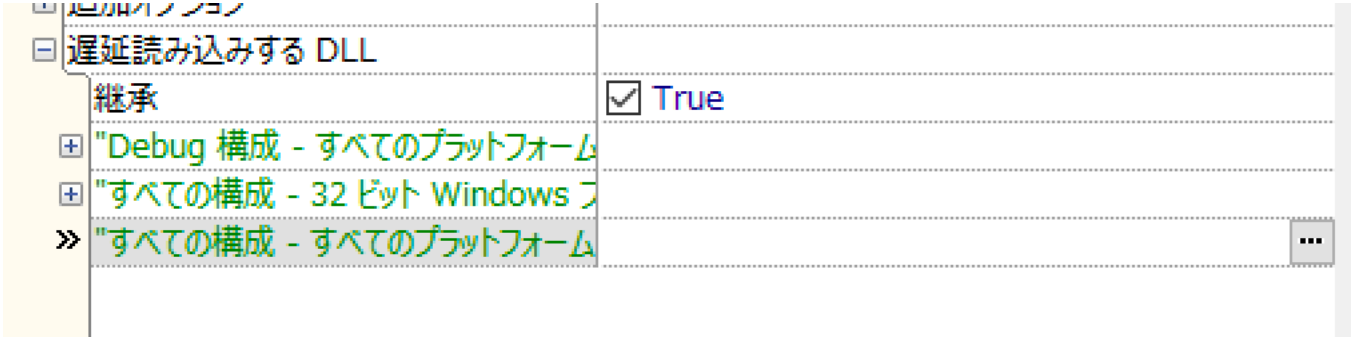
- Delphi / C++Builder はシンプルな方法で
  静的にDLLをロードできます

- DLLを分割している場合必ず依存関係でDLL HELLにぶち当たります

遅延読み込みするDLL

- DLL の遅延読み込みは、アプリケーションではめったに使用されない DLL に対して有効です

- 遅延読み込みが指定された DLL は、DLL 内のエントリ ポイントが実際に呼び出されるまでは、読み込みも初期化もされません。



動的ライブラリ DLL

- 呼び出し元の EXE がメモリ マネージャとして BORLNDMM.DLL を使用する。

```
extern "C" __declspec(dllexport) void* __stdcall GetClassInstance() {
        return static_cast<void*>(new Stack);
}

extern "C" int _libmain(unsigned long reason) {
        return 1;
}
```

動的ライブラリ DLL

```
typedef void* (*VoidReturnFunc)();
void _fastcall TForm1::LoadButtonClick(TObject *Sender) {
      HINSTANCE load = LoadLibrary(L"StackLibrary.dll");
       if (load) {
            ShowMessage("Library loaded!");
            EnableButtons();

            VoidReturnFunc myFunc;

            // GetProcAddress は、読み込んだメソッドのポインタを返す
            myFunc = (VoidReturnFunc)GetProcAddress(load, "GetClassInstance");

            // GetClassInstance メソッドを呼び出す
            stack = (BaseStack*) myFunc();
       }
       else {
            ShowMessage("Library not loaded!");
       }
}
```

BPL 実行時パッケージ

- RAD Studio 固有の特別な機能を備えた DLL です。
- dpk ソース ファイルの基本名がパッケージの基本名になります。

BPL 実行時パッケージ

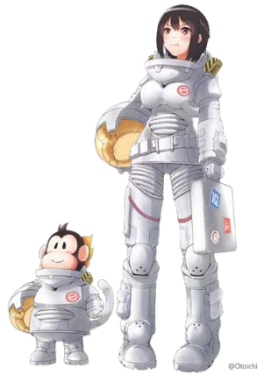
- LoadPackage()を使った呼び出し
- HMODULE

```
HMODULE bpl_;
TForm *frm = 0;
typedef String WINAPI (*t_test1)();
//---------------------------------------------------------------------------
void _fastcall TForm1::FormCreate(TObject *Sender)
{
    bpl_ = reinterpret_cast<HMODULE>(LoadPackage("f3.bpl"));
   FARPROC proc = GetProcAddress(bpl_, "test1");
    t_test1 test1 = reinterpret_cast<t_test1>(proc);
    String s_ = test1();
  Application->CreateForm( TFormClass(GetClass("Tfm_s")), &frm );
}
```

アプリケーション性能の向上 + メモリ効率化

- パッケージ化により、実メモリに展開されるモジュール数を制御できるため、不要なモジュールをロードせず、利用可能メモリ拡大が期待できます

継続的インテグレーションの実現



embarcadero®
DEVELOPER CAMP

継続的インテグレーションの実現

- 構成管理ツールを活用した機能ドリブン開発が実施可能に
  - 機能ごとに分業した開発により、特定機能のリリースタイミングを開発進捗に応じて柔軟に変更でき、複数のリビジョン構成で開発を進めることができます
- デイリービルドの実施
  - ビルドプロセスを自動化し、夜間バッチ等でアプリケーションを構築できるようになり、開発効率、テスト効率、メンテナンス効率の向上が期待できます

パッケージ化に伴う開発プロセス改善の効果

- モジュール単位での開発、テストの効率化／最適化
  - 共通機能の単体テスト効率、精度の向上
    - 共通機能を切り出してDLL化することで、機能仕様に合致するかどうかに集中した単体テストが可能となります
  - アプリケーション機能ごとの開発、組み上げが容易に
    - 機能ドリブンでの開発が可能となる上、プラグイン化により、最終組み上げ段階でのビルドエラー発生のリスクを大幅に回避できます
  - 分業化、外注化が容易に
    - 要件仕様が明確となり、機能単位でテスト可能な開発体制となるため、分業化が容易になり、部分外注などが柔軟に行えます
- アプリケーション性能の向上
  - メモリ利用の効率化
    - パッケージ化により、実メモリに展開されるモジュール数を制御できるため、不要なモジュールをロードせず、利用可能メモリ拡大が期待できます
  - プラグイン機能の提供
    - 将来的に追加機能をプラグイン等で提供可能になるため、アプリケーションの機能強化が柔軟かつ容易に行えるようになります

- 継続的インテグレーションの実現
  - 構成管理ツールを活用した機能ドリブン開発が実施可能に
  - デイリービルドの実施

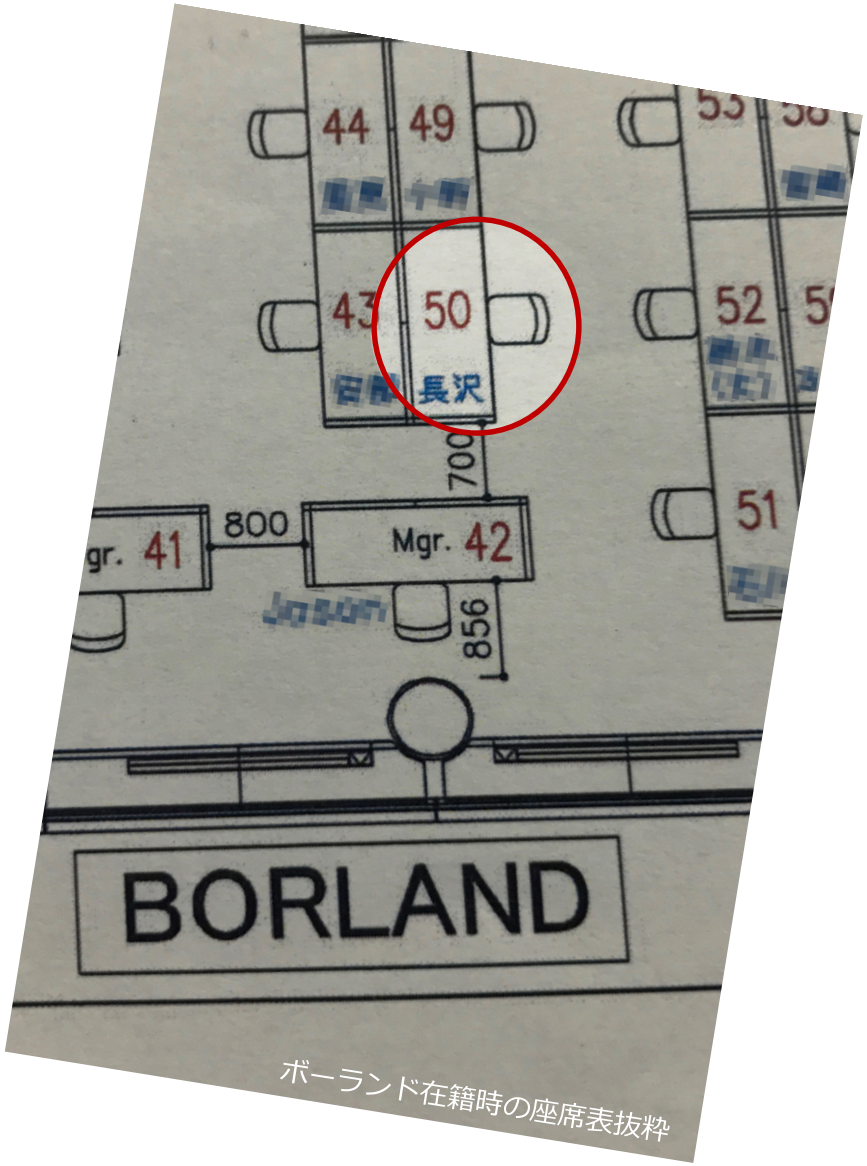
後半の講師は...

長沢 智治

アトラシアン シニア エバンジェリスト

略歴

- 1996.04 – 2000.04: インテック
- 2000.05 – 2003.05: 日本ラショナルソフトウェア
- 2003.06 – 2004.12: 日本アイ・ビー・エム
- 2005.01 – 2006.12: ボーランド
- 2007.01 – 2013.12: 日本マイクロソフト
- 2014.01 – 現在　　アトラシアン



ボーランド在籍時の座席表抜粋

THANKS!

www.embarcadero.com/jp

第34回 エンバカデロ・デベロッパーキャンプ