実機テスト自動化によるテスト効率改善について

大規模なテスト自動化におけるテストスクリプト設計方針と実現

12/29/2011

于　澎
ACCESS CO., LTD.

Confidential

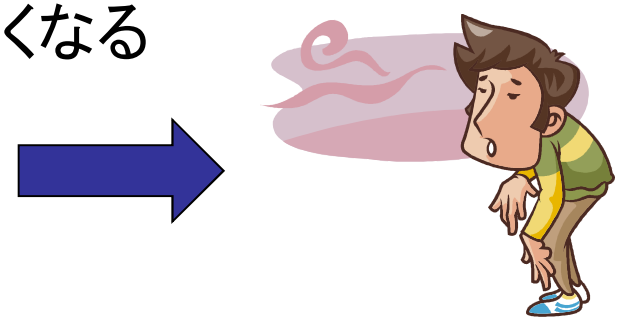
自己紹介

- 于澎（ウポン）
  - Twitter:@upontu
  - Facebook:于 澎(upontu@gmail.com)

- 中国北京市出身、来日7年目
- 電気通信大学修士課程（画像処理）
- （株）ACCESSに入社して3年目
- 品質保証関係の仕事をしている（テスト自動化）
- 趣味：歩くこと（早足で）

歩くメリット

-  ジョギング
  - 走る途中でやめてしまう
    - 筋肉痛、息切れで投げ出したくなる
    - 意志が弱い
  - 三日坊主になる
    - そもそもストレスが溜まる
    - 他のことを並行でできないため、時間がない



-  歩く
  - 効果もそこそこあり、なんと言っても、

長く続けられる！

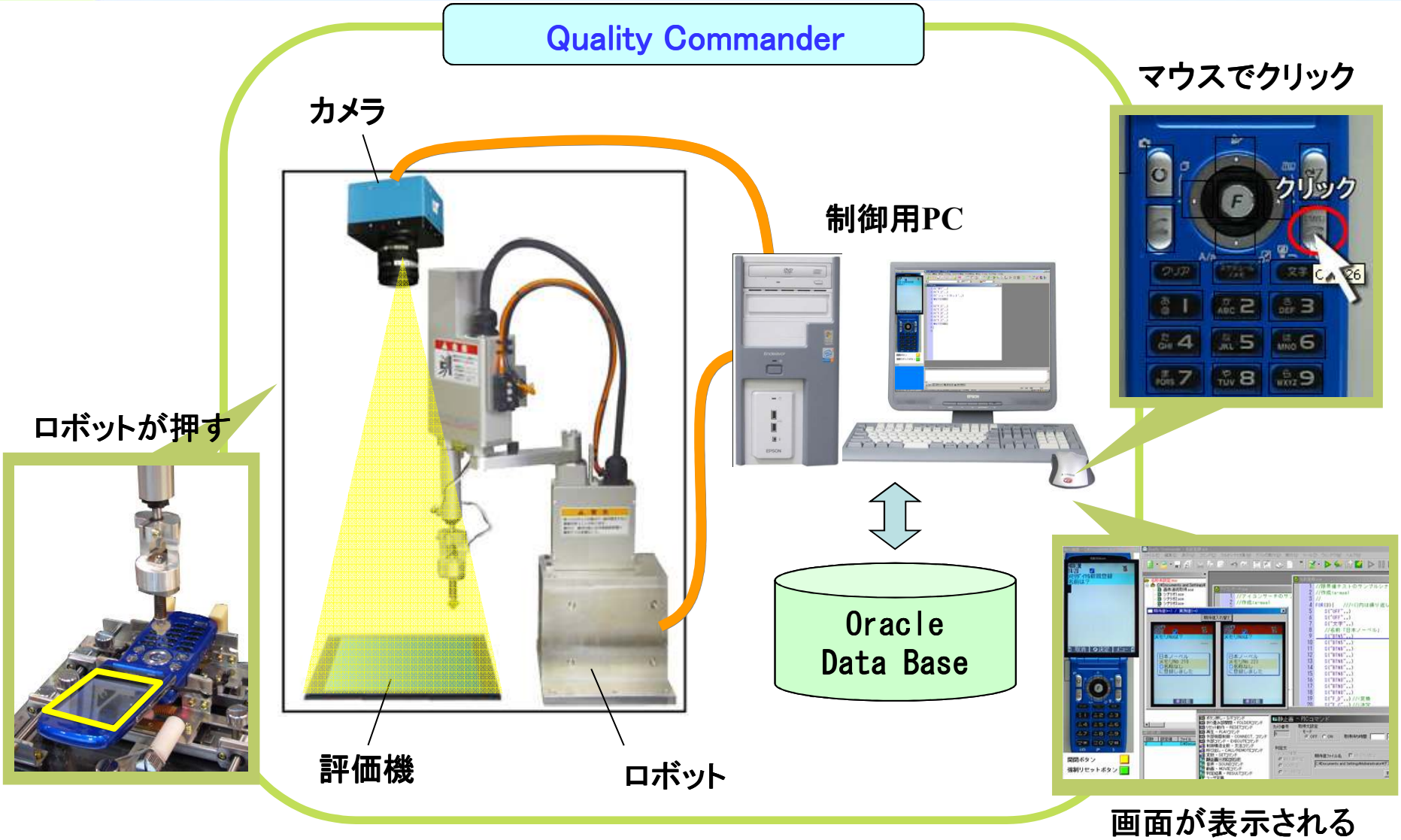
Contents

- 案件事例について
- テストスクリプトの品質特性
- テストスクリプト設計方針
- 実機テスト自動化の今後

案件事例について

- 評価対象
  - 実機：某ゲーム機メーカーさんの新型ゲーム機（タッチパネル）
  - 評価対象：弊社主力製品NetFront Browser
- テストツール
  - Quality Commander（日本ノーベル株式会社）
- 実施したテスト
  - システムテスト（兼リグレッションテスト）
    - テストコンテンツを読み込み、リンクをクリックしたりしてから、表示された画面のレイアウト、文字、画像などを確かめる。
  - ストレステスト
  - ユーザシナリオテスト
- スケジュール：週1回か2週間に1回テストを実施

Quality Commander システム構成



スケジュールの一例

	テストツール（ロボット）	人間（2人）
金曜日	スクリプトテスト、その他テスト、スクリプト改修確認、テスト実施	スクリプト修正、リファクタリング、システムメンテナンス
土曜日	テスト実施	休み
日曜日	テスト実施	休み
月曜日	テスト実施	改修確認、テスト結果確認
火曜日	テスト実施	改修確認、テスト結果確認
水曜日	スクリプトテスト、その他テスト、スクリプト改修確認	テスト結果確認、BTSでバグを報告、テストレポート作成、提出
木曜日	スクリプトテスト、その他テスト、スクリプト改修確認	スクリプト修正、リファクタリング、システムメンテナンス

実績（テストスクリプト作成工数の実績 ）

テストスクリプト作成実績日数	18日
テストスクリプト作成実績工数	2.13人月
テストスクリプト作成実績本数	5270本
テストスクリプト行数 （空行抜き、説明文注釈込み）	188669行
テストスクリプトあたり行数	35.8行/本
一行あたりの作成時間	3.97人秒/行
実質テストスクリプト行数 （空行抜き、注釈抜き）	126497行
期待値画像枚数	21140枚
手動記入テストスクリプト行数	6550行
テストスクリプト自動作成 Excelマクロ行数	330行

実績（テスト実施工数）

手動で実施する場合（予想）

	1ヶ月目	2ヶ月目	3ヶ月目	4ヶ月目	5ヶ月目	6ヶ月目	7ヶ月目	合計(人月)
テスト工数	3.00	5.50	5.50	5.50	5.50	5.50	5.50	36.00

教育　　テスト実施(5パス)

自動で実施する場合（実績）

	1ヶ月目	2ヶ月目	3ヶ月目	4ヶ月目	5ヶ月目	6ヶ月目	7ヶ月目	合計(人月)
スペシャリスト	0.50	1.20	1.00	0.50	0.50	0.20	0.20	3.9
エンジニア	0.50	1.20	1.00	1.00	1.00	1.00	1.00	6.5

スクリプト設計　スクリプト作成　テスト実施(8パス)

自動テスト実行時間の実績

	1パス目	2パス目	3パス目	4パス目
実行時間(h)	107.57	120.28	107.44	117.00
実質実行時間	78.22	92.55	94.25	96.74
稼働率※1	72.71%	76.94%	87.73%	82.69%
途中で止まった回数	12	1	4	1
平均故障間隔(h)	8.9643	120.28	26.859	117

不具合検出件数

	1パス目	2パス目	3パス目	4パス目	5パス目	6パス目	7パス目	8パス目
新規不具合件数	13	0	0	0	0	0	2	0

導入効果

- 合計でコストをおよそ70%に削減できた
- リリース毎にテスト実施ができ、不具合調査、改修が早くなる
- 人間の休み時間にテスト実施ができ、最終的にテスト3回多めに実施できた
- 人間が実施に向いていない長時間ストレス系テスト実施も楽になり、チームモチベーション維持に貢献

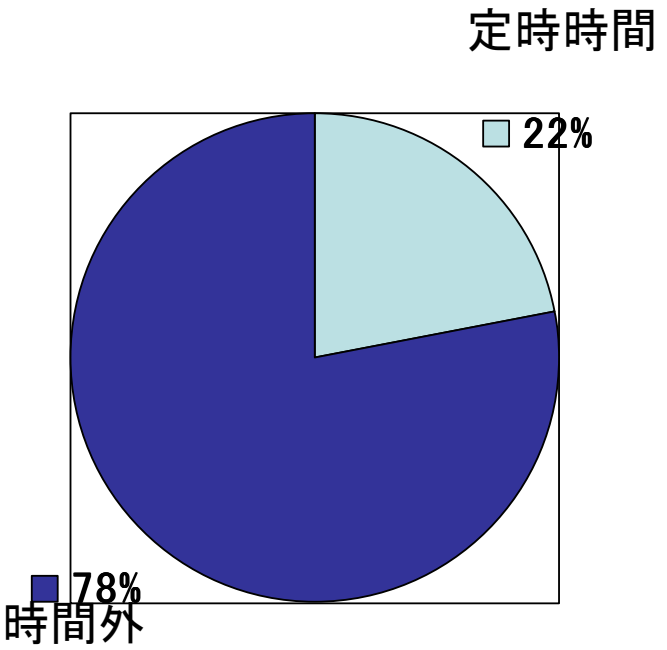
テスト自動化のメリット

- コスト削減
  - 生産性が高い：夜間、休日の区別なく稼動させることができる
- スクリプトが再利用しやすい
  - 一度スクリプトを作れば、何度でも使える
  - 後継案件にでも使える
- その他
  - 操作ミスがない
  - 実行時間短縮
  - テストにかかる時間を正確に把握
  - モチベーションアップ

定時時間外の活用

月	平日	休日	休日（土日、祝日、その他）
1月	19	12	元日、成人の日、年始休暇（1日）
2月	19	9	建国記念日
3月	21	10	春分の日
4月	21	9	昭和の日
5月	18	13	憲法記念日、みどりの日、子供の日、振り替え休日
6月	22	8	
7月	22	9	海の日
8月	18	13	盆休み（3日）
9月	19	11	敬老の日、国民の休日、春分の日
10月	21	10	体育の日
11月	19	11	文化の日、勤労感謝の日
12月	22	9	天皇誕生日、年末休暇（3日）
日数	241	124	有給休暇含めず

年間時間	①365日×24時間	8760時間
定時時間	②241日×8時間	1928時間
時間外	① - ②	6832時間

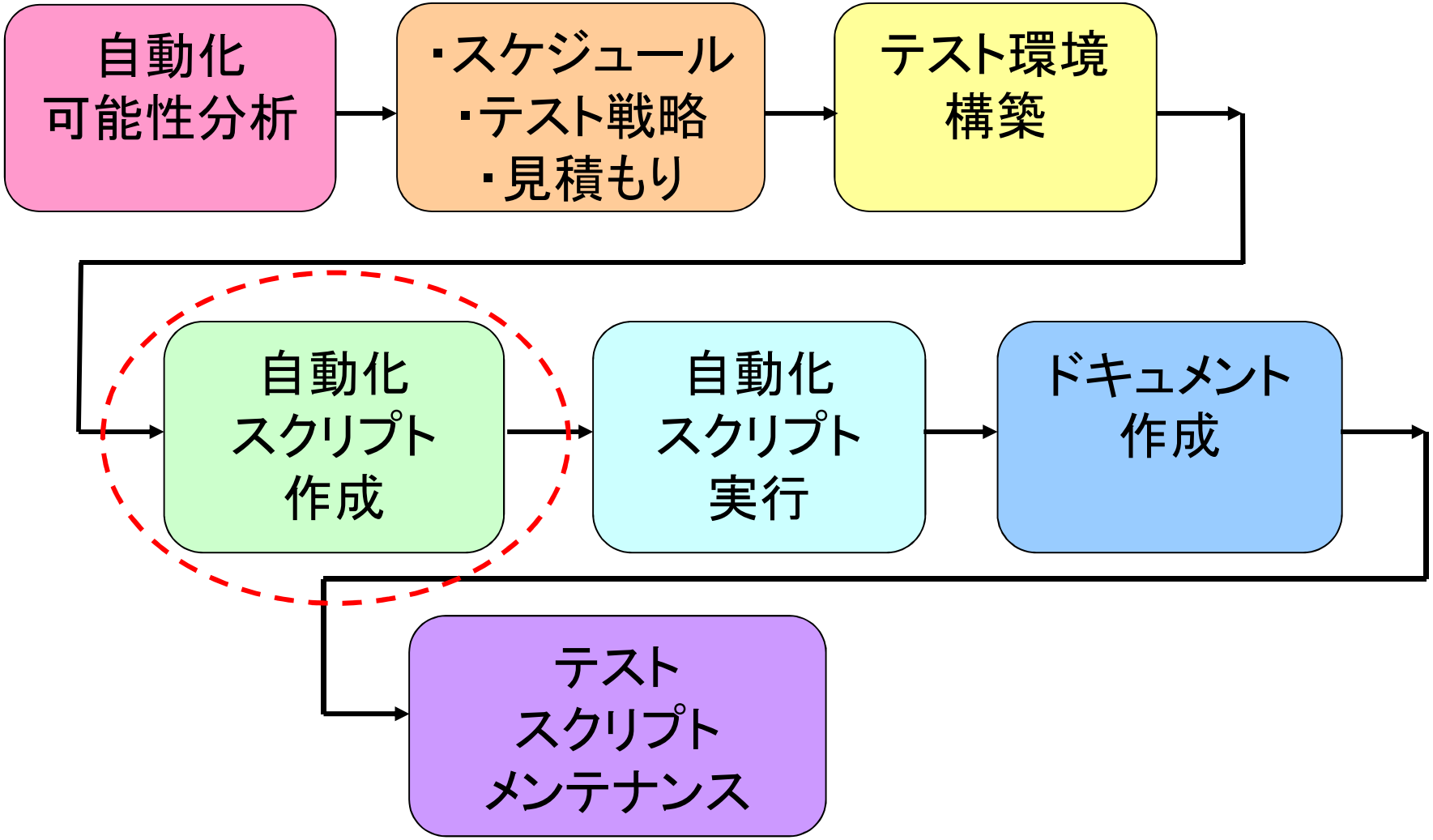


時間外割増手当　25%以上　　深夜割増手当　25%以上　　休日割増手当　35%以上

テスト自動化のメリット

- 高橋寿一さんの著書、第9章「テストの自動化という悪魔」
  - 筆者の行ったかなりの自動化プロジェクトはコスト的に見合うものではなかった。
  - 度重なるUI変更のせいで、自動化プログラムが半分ぐらい動かなくなった
  - 大学でテストの自動化スクリプトをどのように書くかを誰も教えてくれないのです。
- 自動化ツールさえ導入すれば、テスト自動化のメリットを活用できるというわけではない。

テスト自動化の流れ

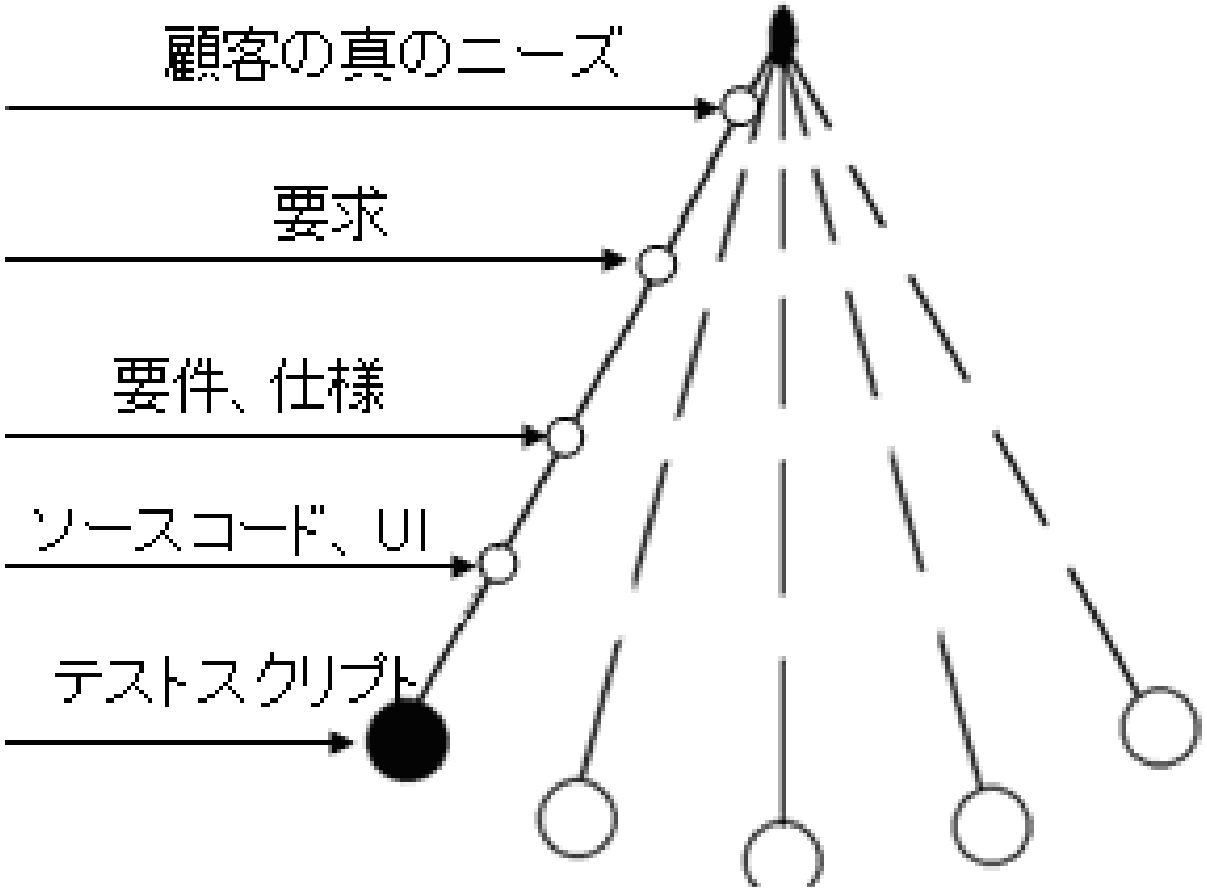


Contents

- 案件事例について
- テストスクリプトの品質特性
- テストスクリプト設計方針
- 実機テスト自動化の今後

テストスクリプトの品質特性1

- 保守性
  - 自動テストメリット1：一度スクリプトを作れば、何度でも使える



情報の分離

- どうせ仕様変更があるなら
- スクリプト変更を楽にする
  - スクリプト変更箇所を特定しやすくする
  - スクリプト仕様変更箇所を少なくする
  - スクリプト変更漏れ、スクリプト変更副作用を最低限に抑える
- そもそも仕様変更があっても困らないようにする
- スクリプト変更は暇なときにゆっくりできるようにする

テストスクリプトの品質特性2

- 信頼性
  - 自動テストメリット2：生産性が高い：夜間、休日の区別なく稼動させることができる
  - ＝＞信頼性を高める必要がある
- テストと止まる要因
  - スクリプトのバグ
  - 環境の影響（停電、社内ネットワーク故障）
  - テスト対象の既存バグの影響
  - テスト対象が操作を受け付けなかったりする

テストスクリプトの品質特性3

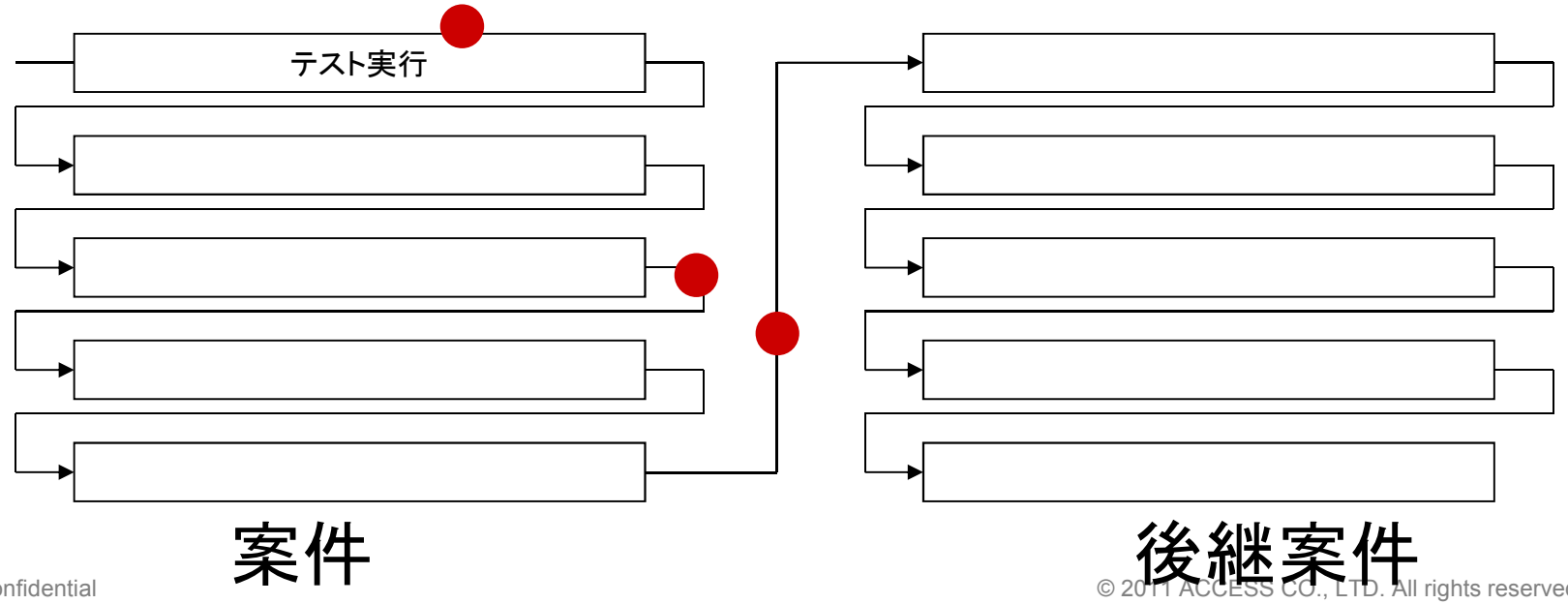
- 移植性
  - 自動テストメリット3：後継案件にでも使える
  - ＝＞移植性を高める必要ある
- 高めるには
  - テストスクリプトの機能を細分化して、機能もしくはモジュール毎に切り出せるように設計する
  - なるべくモジュールに汎用性を持たせる

テストスクリプトの品質特性

- 生産性が高い
  - 生産性が高い：夜間、休日の区別なく稼動させることができる → 信頼性
- スクリプトが再利用しやすい
  - 一度スクリプトを作れば、何度でも使える → 保守性
  - 後継案件にでも使える → 移植性

自動テストで最も重要なこと

長く続けられる
ようなテストにしよう



Contents

- 案件事例について
- テストスクリプトの品質特性
- テストスクリプト設計方針
- 実機テスト自動化の今後

テストスクリプト設計方針

- 情報の分離
- 座標変換
- リトライ
- テストスクリプトの自動生成
- テストスクリプトの階層化
- その他

情報の分離

- ハードコーディングしないこと
- 例1：a.htmlのリンクAをクリックする
  - PUSH(100, 100) ← よくない
  - TOUCH(“a.html”, “A”) ← よい
- 例2：画面Aから画面Bに遷移する
  - PUSH(100, 100) ← よくない
  - TRANSIT(“A”, “B”) ← よい

情報分離のメリット

- 可読性アップ
- 保守性アップ
  - スクリプト修正箇所が特定しやすくなる。
  - スクリプト修正箇所を最小限にできる

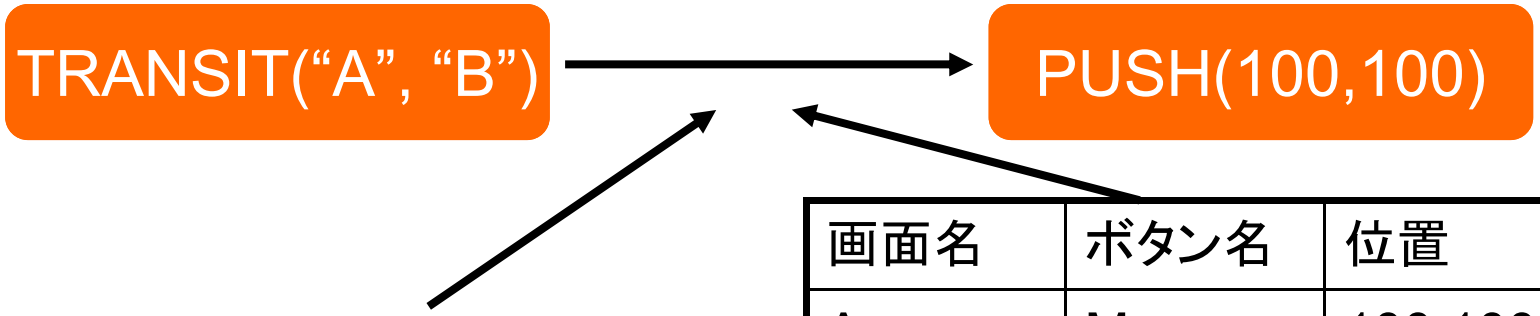
情報の分離

- 以下の情報を分離する
  - テストケース（どうテストする）
  - テストコンテンツ（UI、遷移方法など）
  - テスト対象ソフトウェア（UI、遷移方法など）
  - テスト対象ハードウェア（サイズなど）
  - テスト環境（実機の置き方など）
  - テストツール

座標変換

- スクリプト実行時にメンテナンスしやすいコマンドから、機械が分かるコマンドに変換　→　辞書を用意する必要がある
  - PUSH(100, 100)→機械が分かるコマンド
  - PUSHEX(“A”, ”B”)→メンテナンスしやすいコマンド

スクリプト実行時の動作



	A	B	C
A		Menu	Browser
B	Back		Browser
C	Back	-	

辞書1：遷移表

画面名	ボタン名	位置
A	Menu	100,100
A	Browser	100,200
B	Back	200,100
B	Browser	200,200
C	Back	100,100

辞書2：ボタン位置表

辞書

- マクロ定義
- 画像処理で変換
- 連続番号
- 連続番号＋マクロ
- 空気の読める座標変換
- テストコンテンツ解析

辞書の形1　―　マクロ定義

- 画面要素位置が変わらない場合に有効
- 画面要素位置に頻繁に変更が入る場合は何度も辞書を修正する必要があるため、効率的ではない
- 画面要素位置が一定ではない場合も適用しづらい

画面名	ボタン名	位置
A	Menu	100,100
A	Browser	100,200
B	Back	200,100
B	Browser	200,200
C	Back	100,100

辞書の形2　ー　画像処理で変換

- 画面要素の見た目が変わらない場合に有効
- 画面要素の見た目に頻繁に変更が入る場合は何度も辞書を修正する必要があるため、効率的ではない
- 一つの画面に同じ見た目の画面要素が複数ある場合は適用しづらい

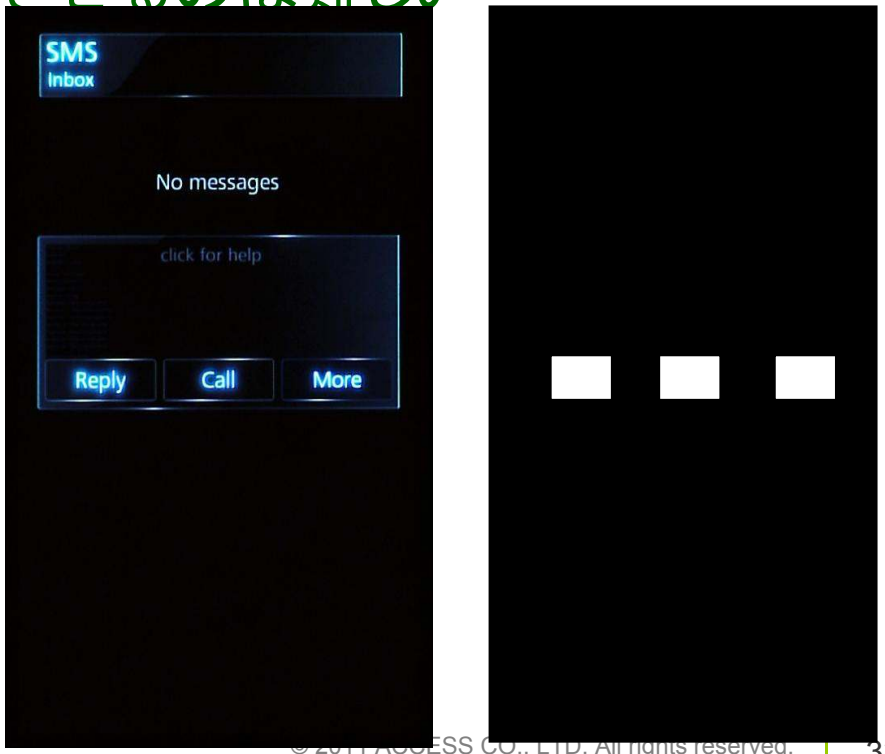
画面名	ボタン名	ボタン画像
A	Menu	
A	Browser	
B	Back	
B	Browser	
C	Back	

押下する前に画面から
ボタン画像を探して
座標を取得してから押下する

辞書の形3　―　連続番号

- 画面要素の数、相対位置が変わらない場合に有効
- 画面要素の数、相対位置に頻繁に変更が入る場合、相対位置が一定でない場合は何度も辞書を修正する必要があるため、効率的ではない
- 画面要素の抽出を100%成功させるのは難しい

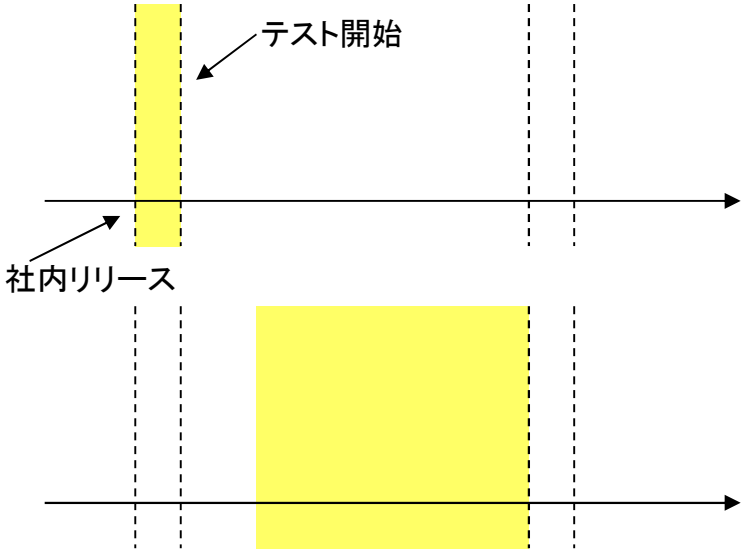
画面名	ボタン名	連続番号
A	Menu	2
A	Browser	3
B	Back	1
B	Browser	2
C	Back	1



辞書の形4　―　連続番号＋マクロ

- 画像処理がうまく行かなくても、位置が変わっても、もう片方の情報で補うことができ、変更にさらに強くなる
- 管理する情報が増える
- 仕様変更に対応するタイミングをずらすことができる
  - ログで逆に仕様変更を検出することができる
  - 仕様変更への対応をテスト実行後に実施することができる。

画面名	ボタン名	連続番号	位置
A	Menu	2	100,100
A	Browser	3	100,200
B	Back	1	200,100
B	Browser	2	200,200
C	Back	1	100,100



辞書の形5 ― 空気の読める座標変換(研究中)

- さらに精度が強くなる
- さらに管理する情報が増える

画面名	ボタン名	連続番号	位置	ボタン画像
A	Menu	2	100,100	
A	Browser	3	100,200	
B	Back	1	200,100	
B	Browser	2	200,200	
C	Back	1	100,100	

辞書の形6 ー テストコンテンツ解析（研究中）

- テストコンテンツを解析して、ボタン名からボタン位置を特定する
- 辞書が要らなくなる。
- 解析する仕組みが必要なので、ハードルが高すぎる。

ライブラリ

- 辞書を使うためには、辞書の引き方を記述するライブラリを用意する必要がある
- 画像処理用ライブラリも必要になる。OpenCV
- 他にもライブラリが必要

リトライ

- 端末を操作するたびに、端末が操作を受け付けたか確認し、受け付けなかったらリトライする仕組み
- リトライ処理を行わなければ、ほぼ確実にテストは失敗する
  - ロボットで実機を操作する場合、実機が100%反応するわけではない！！
  - PC上で操作する場合も100%にならないことがある
- その後の操作がすべて無意味になり、場合によって、テストが続けられなくなることさえある。
- テストにかかる時間の見積もりが困難になる

実行回数	1回以上失敗する確率
1回	1 - 99.9% = 0.1%
100回	1 – 90.5% = 9.5%
1000回	1 – 36.8% = 63.2%
10000回	99.995%

テストスクリプトの自動生成

```
//TestID:1
//大項目:Home画面
//テスト手順：1．テストコンテンツを読み込む
//　　　2．リンクを押す
//　　　3．結果を確認する
//期待結果：次の画面に遷移できる

READ(“c:¥workspace¥testcontents¥Home¥01.html”)
PUSH(1)
PIC(0)
ASSERT(“c:¥workspace¥Expected Result¥Home¥01.jpg”)
```

テストスクリプトの自動生成

- 試験仕様書からテストスクリプトを100%自動生成することは難しい（自然言語の解析が必要）
- ただし、手動で書く内容を最小限にしたい
- VBAを活用しよう

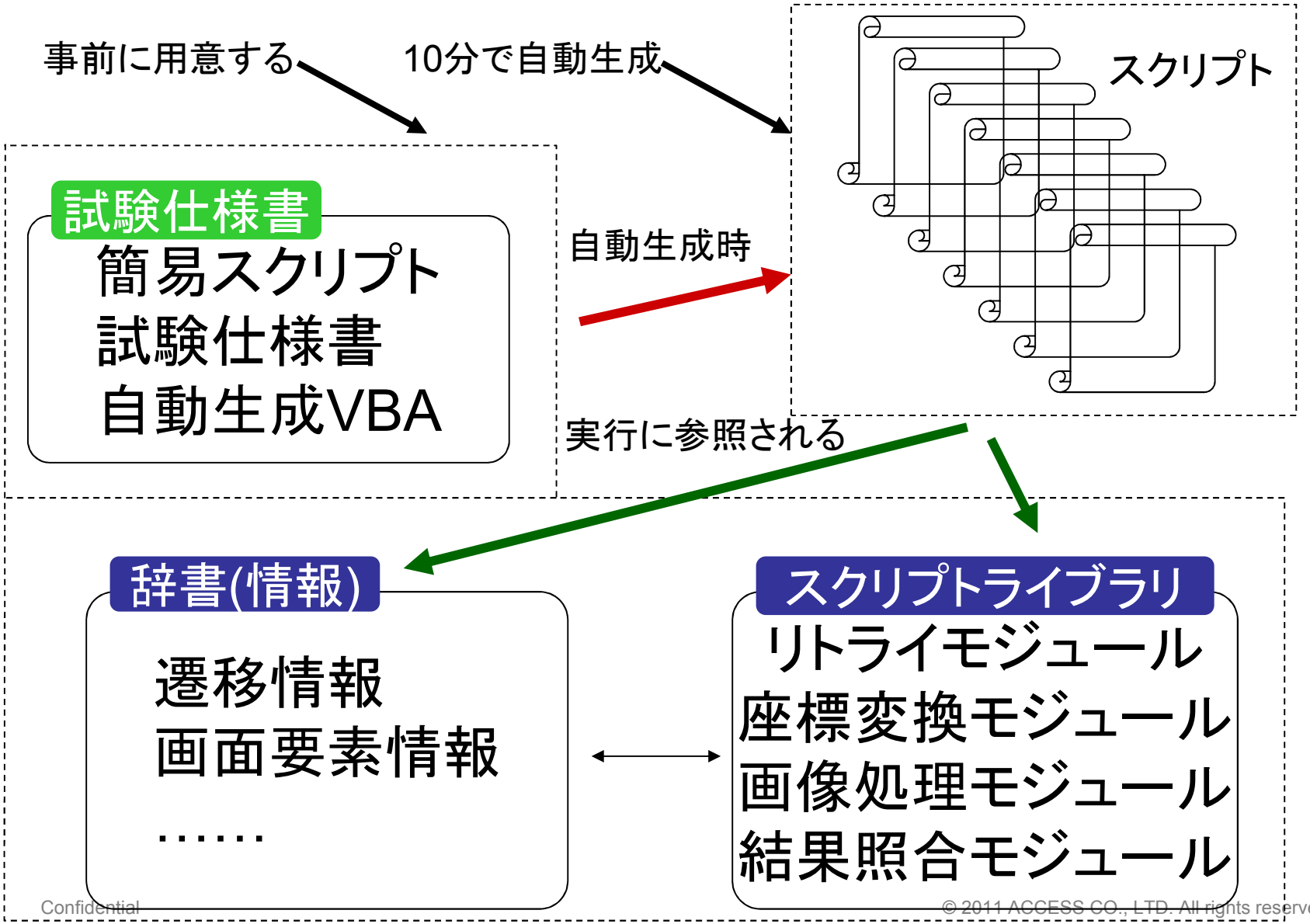
試験仕様書 ← → 簡易スクリプト

ID	大項目	テストコンテンツパス	テスト手順	テスト結果	結果画像パス	テストスクリプト			
1	…	C:¥...	リンクを押す	次の画面に遷移する	C:¥...	READ()	TOUCH(1)	PIC()	ASSERT()
2	…	C:¥...	Backボタンを押す	前の画面に戻る	C:¥...	READ()	TOUCH(“BACK”)	PIC()	ASSERT()

テストスクリプトの自動生成のメリット

- 試験仕様書に記載された情報を二度と書かなくて済む
  - 情報一元管理。保守性アップ！
- エクセルの便利な機能が使える
  - フィルハンドルでコピペが楽
  - 一覧性がよい
  - 置換機能でスクリプト変更が楽
- 自動生成すると同時に、スクリプトのメトリクスが測れる！

テストスクリプトの生成



テストスクリプトの階層化

- 各層の機能が明確になり、問題が出た際に問題特定に要する時間が少なくなる
- 今後の案件も同様な階層化モデルで設計することで、テストスクリプト(ライブラリ部分)の移植性が向上する
- 各層で行われる作業が明確になることでバグを作りこむことが少なくなり、信頼性が向上する。また、修正時の修正漏れと修正副作用も少なくなる

その他

- 一般プログラミング手法
  - 上位ドキュメントとのトレーサビリティ確保。テストスクリプト変更となったときに変更箇所が特定でき、変更漏れを防ぐ
  - 分岐を減らし、スクリプト複雑度を低減
  - スクリプト可読性の向上
  - 気が向いたらリファクタリング
- 既存バグを調べておく
  - 自動テストが可能な程度の安定性を持っているコンポーネントと持っていないコンポーネントの見分けがつき、テストスクリプト作成の優先度を決めることができる。
  - テストスクリプト設計自体に大きな影響を与えるバグがある場合、設計段階から考慮できる（例：極端に発熱量が多いというバグが未解決の場合、1時間毎にテストを10分停止させてから復帰させるように設計しなければならない）。

Contents

- 案件事例について
- テストスクリプトの品質特性
- テストスクリプト設計方針
- 実機テスト自動化の今後

テストスクリプト仕様の理想像

- スクリプト仕様のオブジェクト指向化
  - 簡単そうに見えるが、実は仕組みが非常に複雑
  - ライブラリ部分の保守性、流用性がアップ
- テストプラットフォームの構築
  - 0から構築していると、時間がかかりすぎ
  - ツールメーカーさんが一部を用意してくれると導入ハードルは下がるが、会社ごとに環境がばらばらすぎて、難しい。画像処理の部分なら何とかできるかも
  - 自社製品に適したライブラリを用意する

まとめ

- テスト自動化のメリットはスクリプトを工夫することで最大限に発揮できる。
  - 夜間、休日も仕事できる ⇔ 信頼性重視
  - 一度作れば、何度も使える ⇔ 保守性重視
  - 後継案件にも使える ⇔ 移植性重視
- 仕様変更を見越して、長く続けられるようなテストにしよう
  - 仕様変更があってもスクリプト変更しなくても済むようにしよう ⇔ 辞書の活用
  - スクリプト変更箇所を最小限にしよう ⇔ 情報分離、自動生成
  - スクリプト変更タイミングをずらせるようにしよう ⇔ 辞書の活用
  - 途中で止まらないようなテストにしよう ⇔ リトライ
  - 後継案件に移植できるようにしよう ⇔ スクリプト階層化
- 自動化テストの今後

Thank you