

#### Lesson learnt from integrating MBT for Messaging App

By Chan Chaiyochlarb

© 2013 Microsoft Corporation. All rights reserved. Microsoft, Windows, Windows Vista and other product names are or may be registered trademarks and/or trademarks in the U.S. and/or other countries. The information herein is for informational purposes only and represents the current view of Microsoft Corporation as of the date of this presentation. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information provided after the date of this presentation. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS PRESENTATION.

# Why MBT?



#### Motivations for adopting MBT

Find bugs earlier Exploration of missed paths Increase validation Easy to adjust to specification changes Real world scenarios Help find the last 20% of bugs Reduce test cost



## MBT For Messaging App



#### MBT For Messaging App

90% of test automations are MBT generatedReal user End-to-End scenariosPermutations of actions yielding high coverageError scenariosUI and backend verifications





## Analysis: The Good



#### MBT finds the most high priority bugs

Uncover a lot of functional specification bugs Catch a lot of regressions Lots of validations



![](_page_6_Picture_3.jpeg)

#### Find bugs early

Test the specification Model development in parallel with product

![](_page_7_Figure_2.jpeg)

#### Agility

Easily react to new feature changes Reusability of test semantics Early test engagement Drive quality upstream

![](_page_8_Picture_2.jpeg)

## Analysis: The Bad

![](_page_9_Picture_1.jpeg)

#### MBT is not easy

Different mind shift from traditional testing Steep learning curve and high ramp up cost Need to pick the right tool set Difficult to explain test coverage

![](_page_10_Picture_2.jpeg)

#### Complex Design

Single model which represents the whole Messaging Application Model is nearly as complex as the product Bug in model is difficult to find State tracking and other book keeping (for validation) make things even worse

Every behavioral change has large impact to existing scenarios

![](_page_11_Picture_3.jpeg)

#### Maintenance Costs

Complexity kills Bug ratio

![](_page_12_Figure_2.jpeg)

More test code means higher maintenance costs Bug turn around time nearly double developers

![](_page_12_Picture_4.jpeg)

### Reflections

![](_page_13_Picture_1.jpeg)

#### Back to our original motivations

Find bugs earlier Yes Exploration of missed paths Yes Increase validation Yes Easy to adjust to specification changes Yes Real world scenarios Maybe Help find the last 20% of bugs Maybe Reduce test cost No

![](_page_14_Picture_2.jpeg)

## Moral of the story

![](_page_15_Picture_1.jpeg)

#### What did we learn?

MBT is different Model Design is important Smaller model is okay It's okay to have multiple models for different feature set "Use MBT to generate a lot of test cases" paradigm is misleading Resist the temptation to use MBT for everything

![](_page_16_Picture_2.jpeg)

#### Knowing MBT strengths and weaknesses

MBT is highly effective for stateful system, or with systems lots of input/output combinations

For stateless system with simple inputs/outputs, it might be more effective using data-driven approach instead

For undeterministic behavior, MBT might not be a good fit Example: Image resizing algorithm, data decompressor, etc.

![](_page_17_Picture_4.jpeg)

# Thank you

![](_page_18_Picture_1.jpeg)