

JaSST'21 Tokyo

Autify を使った自動テストにおける属人化解消について
株式会社 チームスピリット 生井 龍聖



Agenda

- 1 はじめに
- 2 解決したい課題と過去の失敗例
- 3 **Autify**を使ったアプローチ
NoCode
LowCode
自動化の検討
- 4 まとめ



Section1

はじめに



Ryusei Namai

TeamSpirit Inc.

QA Team Manager

JSTQB Test Analyst & Test Manager



[riririusei99](#)



<https://www.linkedin.com/in/ryusei-longsheng/>



<https://recruit.teamspirit.com/job/>



開発体制について

チームスピリットでは2017年からスクラムでの開発を行っており、
現在、製品ごとの各サブシステムごとのスクラムチームで開発を行っている。

（チーム数は6。そのうち2つはシンガポール拠点で開発を行っている）

QAエンジニアは「開発チームの一員」として携わっており、

ストーリーテストの他にも、バグレポートの作成や自動テストの作成・メンテを行う。

専任の自動テストエンジニアは所属していない状態。

はじめに

発表について



自動テスト



運用の仕方

自動テストの導入についての事例発表です

はじめに

解決したいテーマ

属人化



なぜ起こるのかというのを考えました



Section2

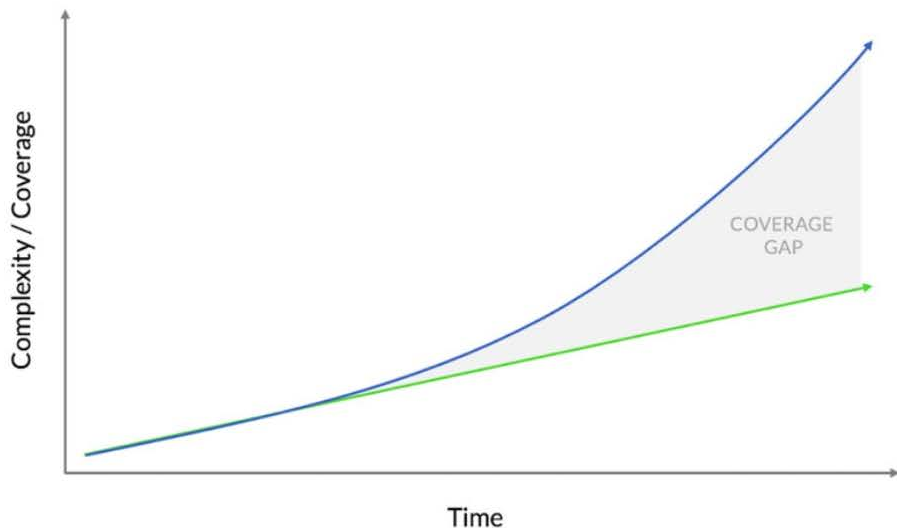
解決したい課題と過去の失敗例

第一次 E2E自動テスト(Java)



バックエンドエンジニアと二人で開発&メンテナンス

プロダクトの成長



Features

Complexity increases exponentially as new features and states interact with existing features

Tests

Test coverage grows linearly because they can only be added one at a time

Source: AI for Software Testing, Jason Arbon, CEO, test.ai
<https://www.linkedin.com/pulse/ai-software-testing-jason-arbon/>

機能追加などのメンテナンス頻度が高まった

時間軸の変化で起こること

- メンバーの成長

新しいことができるようになる

- 新しいスクラムチームの発足

メンテナンスするにも修正内容のキャッチアップが必要

- メンバーの異動・昇進・退職など

人事発令に対する適応力の低さが露見した



プロダクトの変化の他にも個人・チーム・ビジネス全体にも変化が起こる

時間の変化



Sugiyama Daisuke/杉山 大輔 🍷 2018年1月4日 17:44

みなさん初めまして。
杉山と申します。
また改めてご挨拶させていただきます。
よろしくお願いいたします！



5



1



時間の変化はチームにも変化を与える

原因を考える



幅広いスキルセットが必要

テスト設計・環境構築・コーディング・メンテナンス等



継続開発の考慮不足

プロダクトの成長・環境の変化に適応できる

二つの要素が属人化に影響していると考えた



Section3

Autifyを使ったアプローチ

Autifyとは

1/24 02:12 In Out

Time & Attendance DEPT1 / Sista Aadmi Working Type TST_Fix Attendance Report

Request	Date	Clock-In	Clock-Out	Time Tracking	2	14	16	18	20	22	24	26
+	1 金			+								
+	2 土			+								
+	3 日			+								
+	4 月			+								
+	5 火			+								
+	6 水			+								
+	7 木			+								
+	8 金			+								
+	9 土			+								
+	10 日			+								
+	11 月			+								
+	12 火			+								
+	13 水			+								
+	14 木			+								
+	15 金			+								
+	16 土			+								
+	17 日			+								
+	18 月			+								
+	19 火			+								
+	20 水			+								
+	21 木			+								
+	22 金			+								

画面録画

3 5秒間待機する

4 出勤を選択 - クリックする

5 勤怠打刻: 出勤 - クリックする

9 10:51

10 勤怠打刻: 退勤 - クリックする

11 10秒間待機する

15 10秒間待機する

16 10:47

17 21 火 22 水 23 木

21 (00:00)

22 初期化 - と入力する

23 10:25 (00:00) 10:10

初期化

初期化

初期化

完了

シナリオ生成

シナリオ

- ✓ [New-arch V1] Regression Attendance - No8 | Web打刻
- ✓ [New-arch V1] Regression Attendance - No11 | 承認者の
- ✓ [New-arch V1] Regression Attendance - No45 | チームタ
- ✓ [New-arch V1] Regression Attendance - No1 | 全社カレ
- ✓ [New-arch V1] Regression Attendance - No2,3 | カレン
- ✓ [New-arch V1] Regression Attendance - No5 | 月度移動
- ✓ [New-arch V1] Story AttendanceGENIE-18177 | 勤務表月

回帰実行

解決策を考える



幅広いスキルセットが必要

シンプルに使える



継続開発の考慮不足

習得が容易

この二つを満たす手段としてAutifyの利用を開始した

Before

BE + QA

実装・メンテナンス

QA

自動化の検討

After

はじめて

NoCode

チームでつくる

経験者

LowCode

大事な部分に集中する

運営

自動化の検討

チームで取り組めるようにする

NoCode開発

Autifyをシンプルに使うってテストケースを増やす



NoCode開発

Autifyのシナリオ作成機能を使って自動テストを作成
必要なスキルのハードルを下げることで、チームでつくる



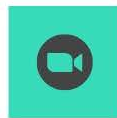
STEP #1

- Autifyのレクチャー
- チーム内ルールを読んでもらう



STEP #2

- チームのバックログに組み込む
- テストシナリオを準備しておく



STEP #3

- Autifyでシナリオ作成する
- 成功したらテストプランに組み込む



完成！

- 並列実行で失敗する場合は別タスク

NoCode開発の利点

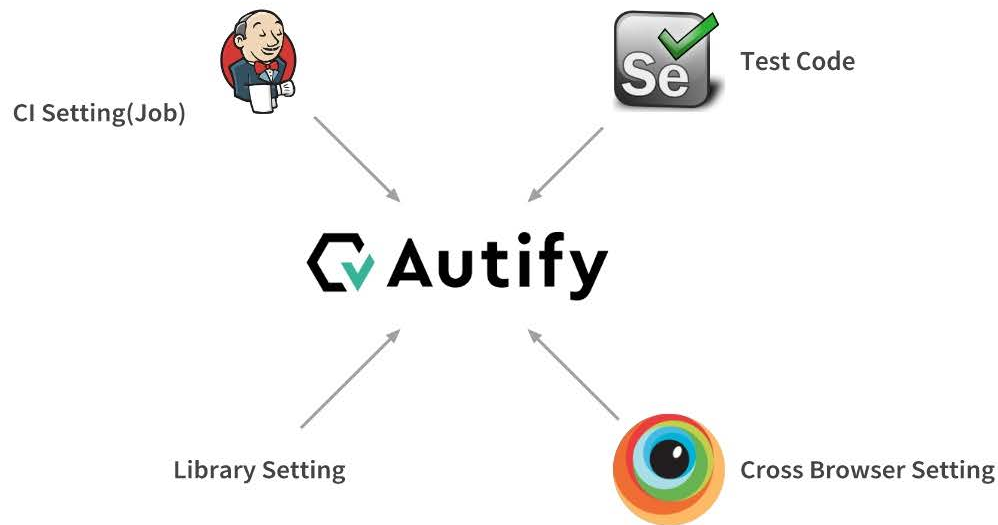
自動化のハードルを下げることで、自動テストに関わる人を大幅に増やすことができた

- 作業者が増えたことで開発できる総量が増えた
NoCodeの作業だけで回帰実行できるテストシナリオも多い
- 環境構築の手間が不要
失敗の原因を実行環境による違いなどから除外でき、
Chromeプラグインをインストールすればすぐに着手できる
- 設定部分がAutifyに集約され理解しやすくなった

開発チーム20人にAutifyを利用してもらっている

NoCode開発の利点

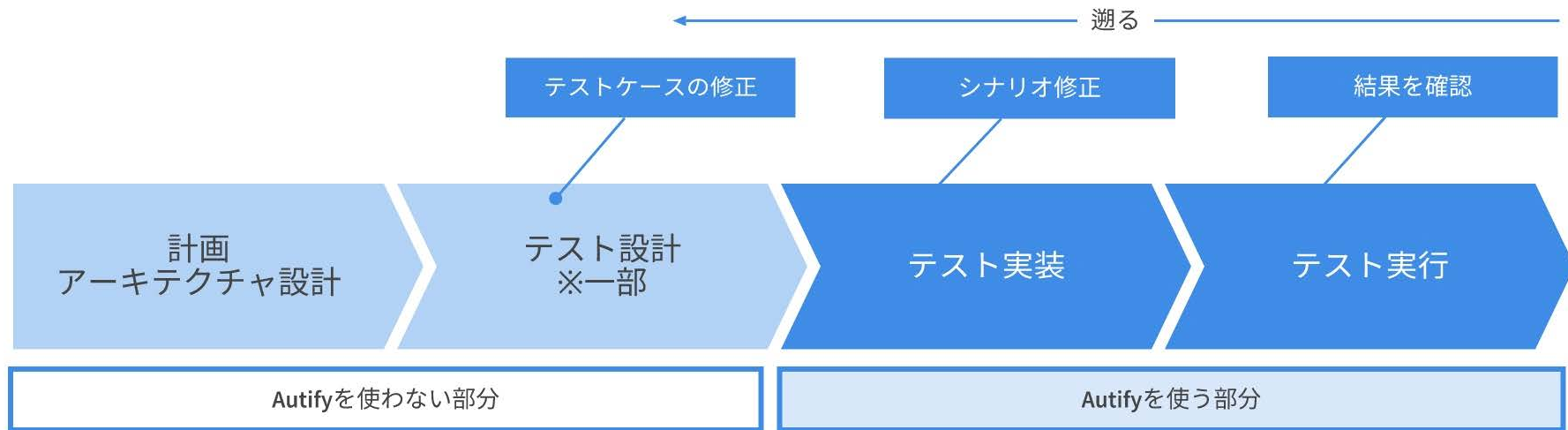
設定部分が集約され、理解しやすくなった

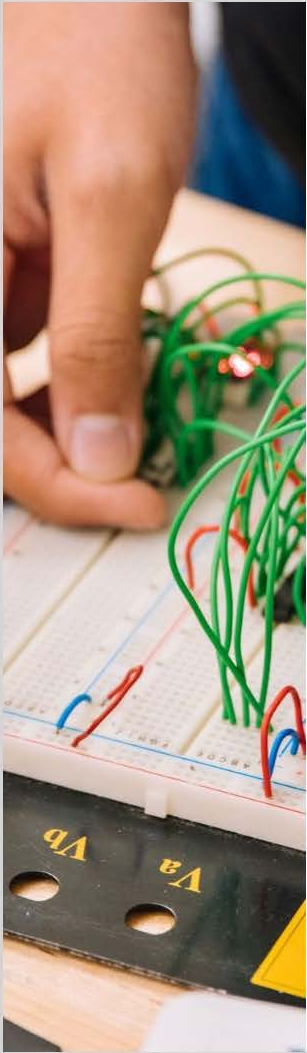


シンプルになり教えやすくなった

LowCode開発

安定した自動テストを作る





LowCode開発

自動化経験のあるメンバーを中心にNoCodeでできない部分をサポートすることで、特定のメンバーしかできない仕事に注力できる。



STEP #1

- 回帰テストの結果を確認
- メンテナンスチケットを作成する



STEP #3

- 成功したらテストプランに組み込み
- 関係するテストを全て実施する



STEP #2

- 落ちている失敗しているシナリオを修正
- JSステップ・待機ステップ等を追加



完成！

LowCode開発の利点

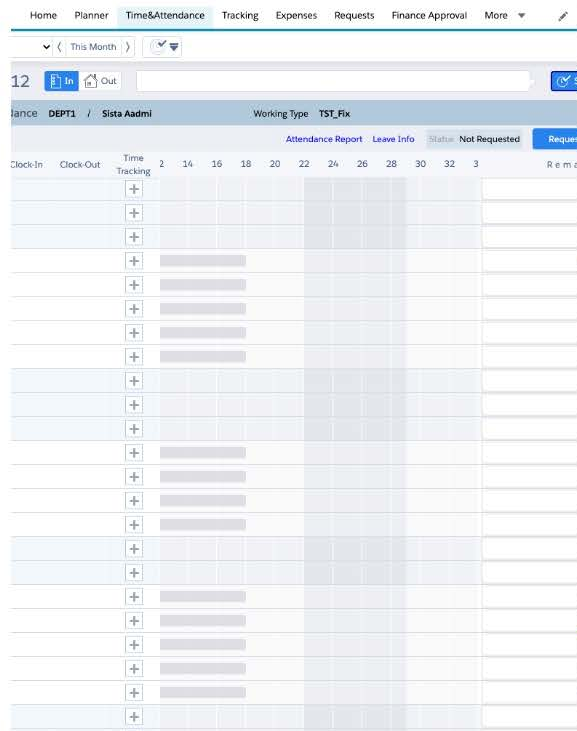
実装にまつわるカスタマイズの部分を機能として提供しているので、その部分の工数を肩代わりしてくれる。

やりたいこと	Autifyだと	自前でやると（例）
並列実行	プロジェクト設定から設定	フレームワークで並列設定
回帰実行	テストプラン作成時に “定期実行”から選択	CIに組み込む & 環境作成
クロスブラウザでの実行	テストプラン作成時に “実行環境”から選択	BrowserStackと連携 or 各Driverのメンテ
通知	Incoming WebHookを使って連携	ジョブを作成 & CI側で連携
レポーティング	“テスト結果”から閲覧	ライブラリ選定 & CI側で設定
トラブルシューティング	サポートあり & snapshotが見れる	SnapShot取得処理を書く & ログをみる
独自のwait処理等の実装	なし	プロダクト毎に開発が必要
特殊なことをやろうとする	JSステップに書く	テストコードに書く

専任のいない環境にとっては十分な機能提供

LowCode開発の利点

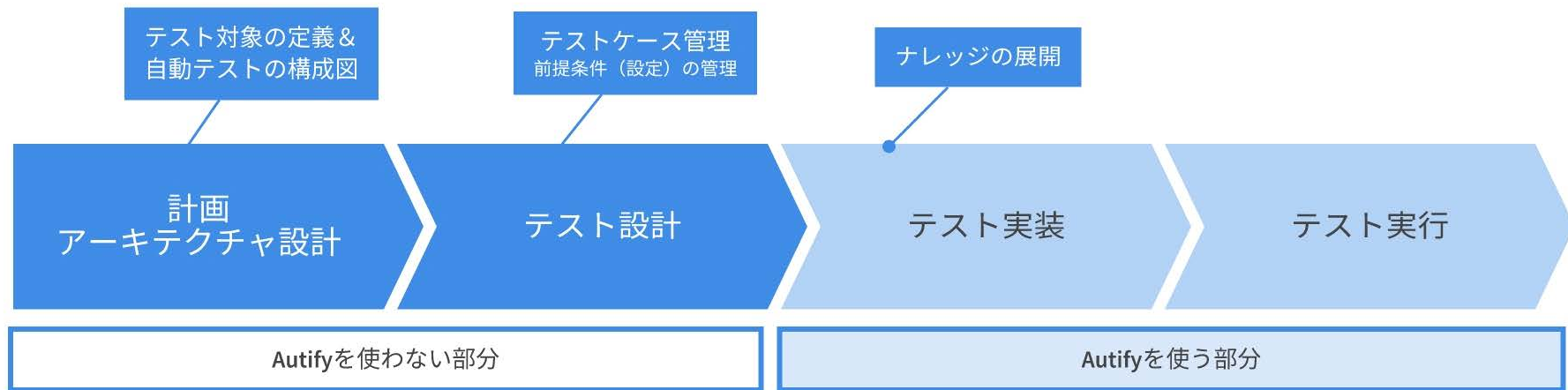
実装時の手間からの解放



- 要素の取得(指定もできる)
- 機能改修に伴うコードの修正
- AIによりUI変化を検知

自動化の検討

テストシナリオを作るまでの準備&ナレッジを蓄積させる



自動化の検討

導入時に準備しておいたこと

- 1 テストプラン(回帰実行の構成)の検討
- 2 テストケース設計と管理方法
- 3 諸問題の解決 & ナレッジ共有手段
- 4 ルール・ポリシーの策定





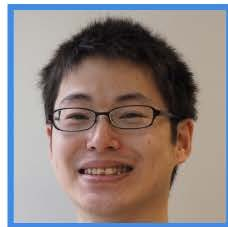
Section4

まとめ

成果



Test Manager
自動化の検討担当



QA Engineer
LowCode領域担当



QA Engineer
NoCode領域担当

導入初期からチームで取り組むことができた

成果

現在の自動テストは多くのメンバーが自動テストに携わっている

20人

NoCode

チームでつくる

5人

LowCode

大事な部分に集中する

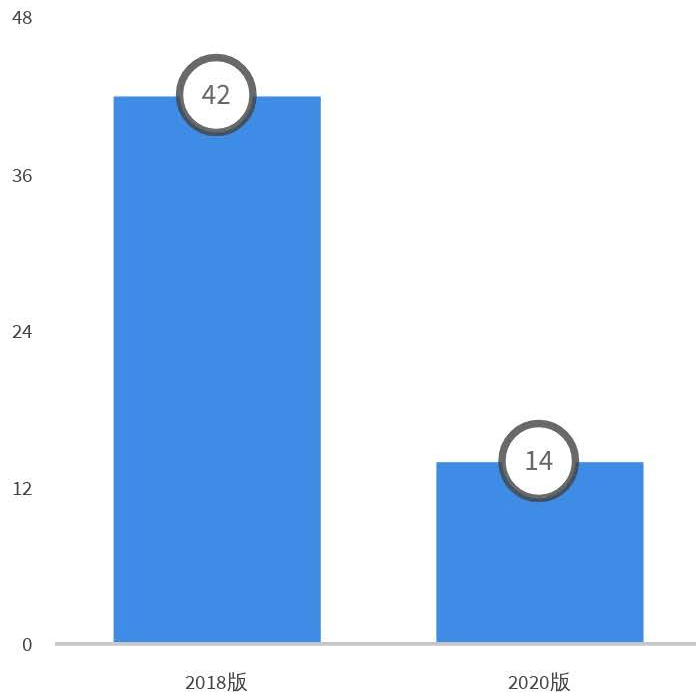
3人

自動化の検討

チームで取り組めるようにする

サイクルタイムの高速化

導入チケットの完了までの日数

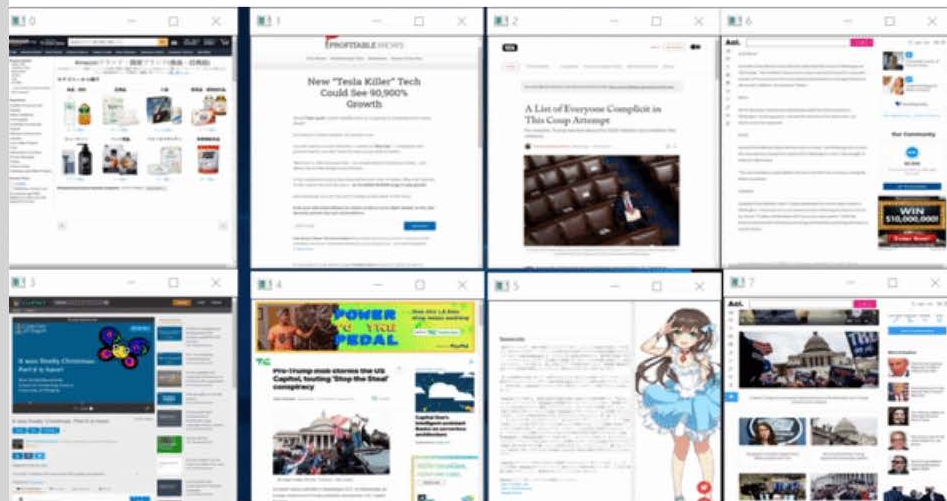


導入と初期のテスト実装チケットの完了までの日数を比較

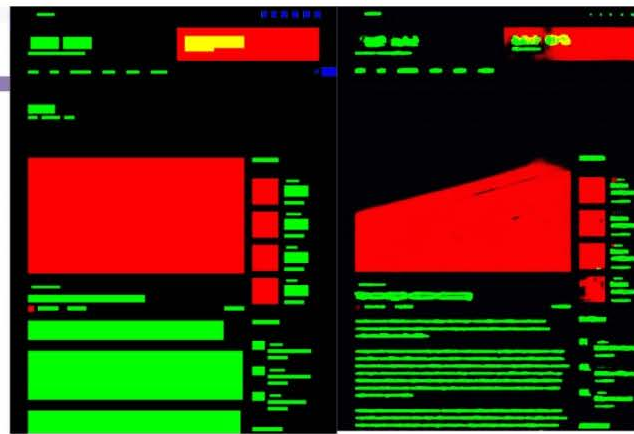
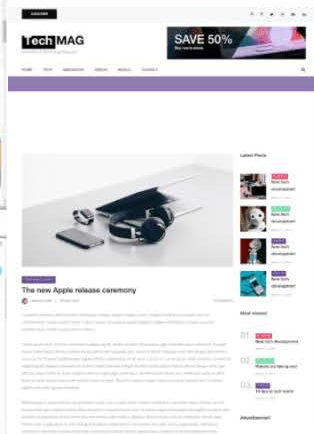
対応人数のボトルネックが解消され、サイクルタイムの高速化を実現。

自動テストにおけるスウォーミングが可能な体制になった。

今後の展望・期待



Intelligent Test Discovery



Visual Testing Engine

SaaSを使うことで高度なテストを使うことに注力できる

A silver laptop is open on a wooden desk. The screen displays a vibrant landscape wallpaper featuring a large, conical mountain under a dramatic, colorful sky (pink, orange, and blue). In the foreground, a river flows through a lush green valley, with a small waterfall cascading over rocks. The Windows taskbar is visible at the bottom of the screen, showing the Start button, several application icons, and the system tray with the time 10:36 AM and date 7/24/2019. A semi-transparent dark rectangle is overlaid on the center of the screen, containing white Japanese text.

課題を分析して、手段を選ぶ

注意！

まとめ

- 作業領域を分けたことで、メンバーを巻き込むことができた
初めての人にも手をつけられる作業になった
- Autifyを導入することでメンテナンスコストを下げた
間接作業を減らし、必要な仕事に集中できるようになった
- チームで使うための準備・仕組みづくりもまた重要
Autifyを使わない部分でも成果物を意識する必要がある



**Thank you
for listening!**

