Exploring Programmatic Interfaces (APIs)

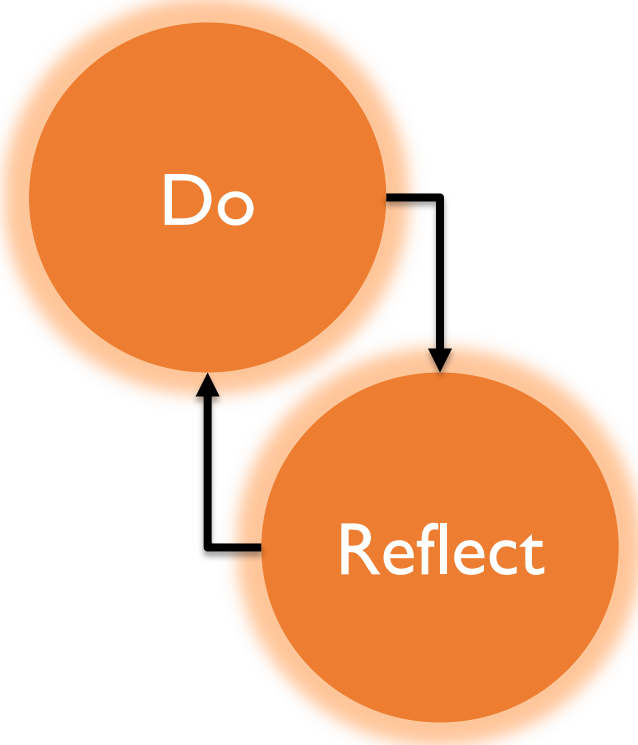
by Maaret Pyhäjärvi

VAISALA

http://maaretp.com

We Learn to Test by Testing

TestThisBox
Gilded Rose
Roman Numerals
ApprovalTests
E-Primer
DFEditor
Freemind
Zippopotamus
React-Weather
Odoo

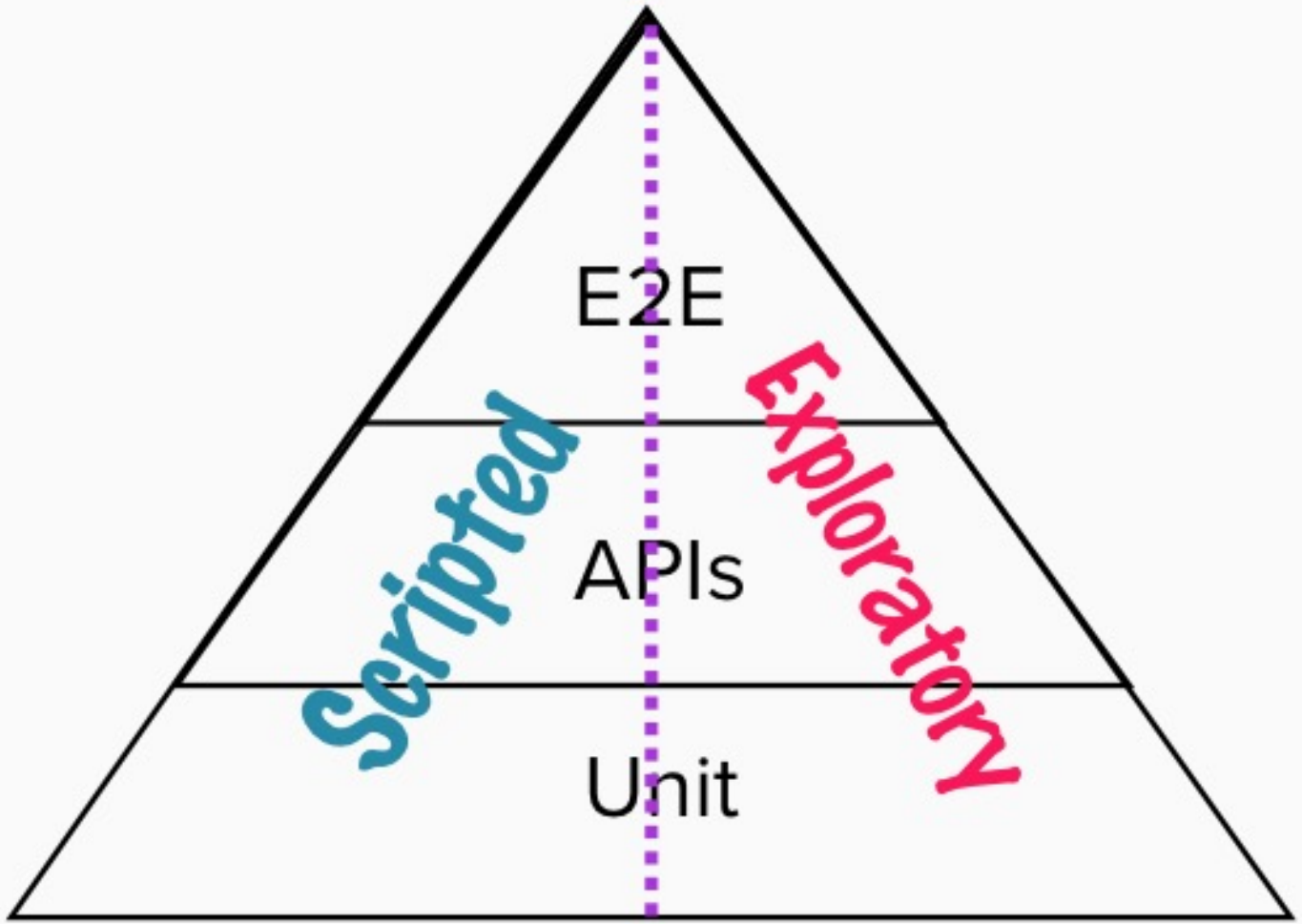


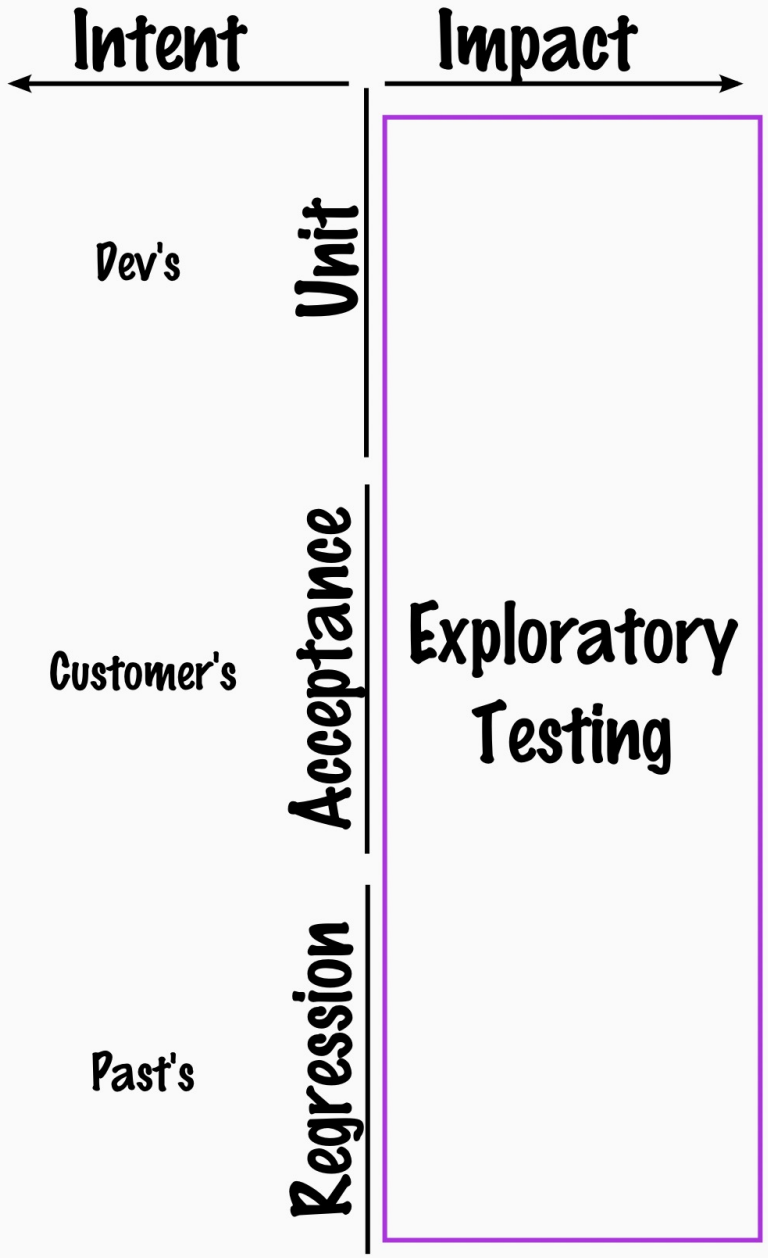
First Experience to Exploratory Testing?

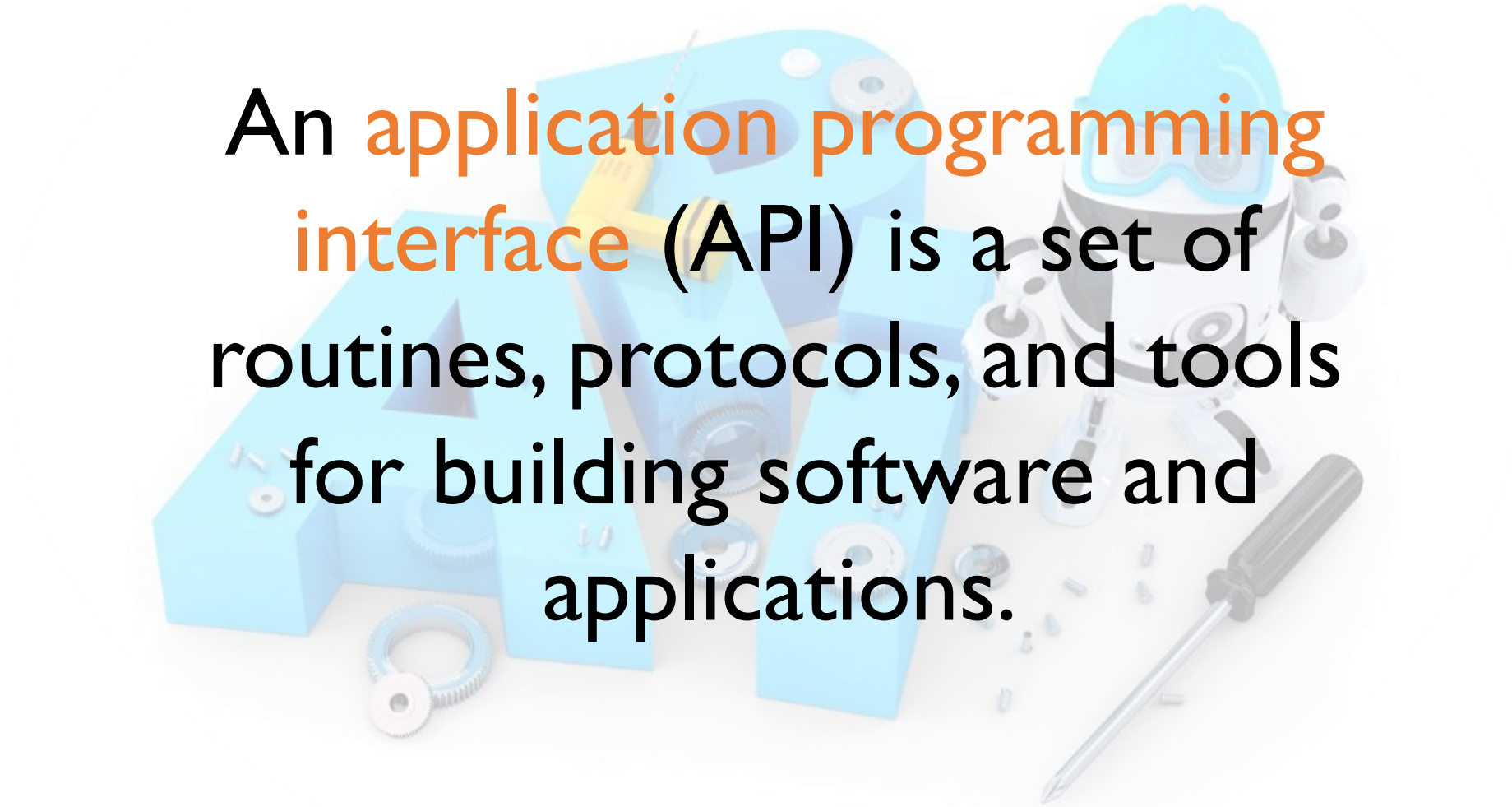
Spec => Tests
+ a Session to Find all Bugs

Repeat list of test cases
+ a conference awakening

Some tests, budget of time, QA on results

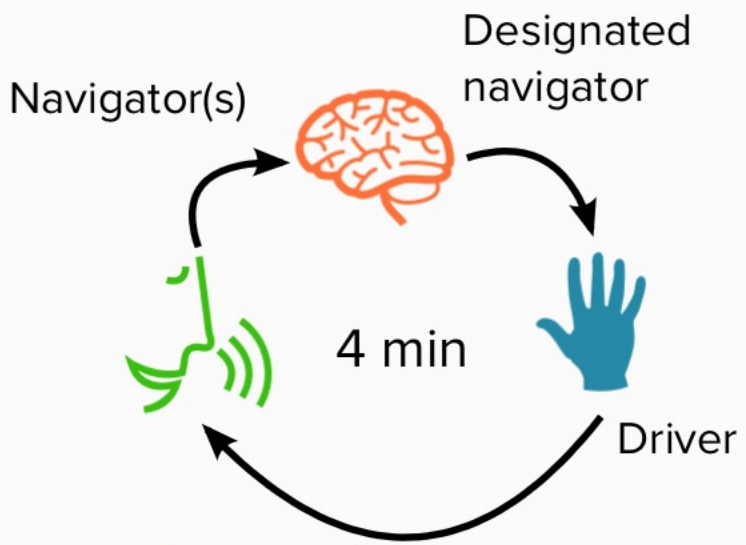

E2E
APIs
Unit
Scripted
Exploratory


Intent
Impact
Dev's
Unit
Customer's
Acceptance
Past's
Regression
Exploratory Testing



An application programming interface (API) is a set of routines, protocols, and tools for building software and applications.

Theory of Ensemble Testing



Lang & API Expert

Facilitator

Bias to action

Instruct on highest level of abstraction

No decisions on keyboard

Maaret Pyhäjärvi (from Finland)

EuroSTAR TESTING EXCELLENCE AWARD

2020

AGILE TESTING DAYS

MIATPP

Most Influential Agile Testing Professional Person

2016

EXPLORATORY Testing

Email: maaret@iki.fi
Twitter: @maaretp
Web: maaretp.com
Blog: visible-quality.blogspot.fi
(please connect with me through Twitter or LinkedIn)

#PayToSpeak #TechVoices
#EnsembleTesting #EnsembleProgramming #StrongStylePairing
#ExploratoryTesting #TestAutomation
#ModernAgile
#AwesomeTesters

DX

Developer Experience

Building it the First Time

Debugging and Maintaining

REST APIs (http(s))

Streaming APIs (ws(s))

Libraries

Frameworks

Languages

Methods

CLI

Protocols and Messages

Exploratory Testing: Frame of Management

Coaching

Test Manager

Debriefing

Past
Results
Obstacles
Outlook
Feelings

Perception of quality and coverage

Quality Report

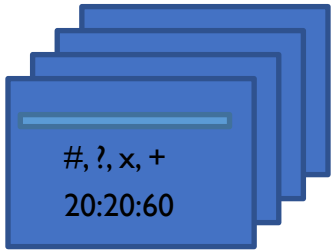
Tester

"A day's work"

Vision ("Sandbox")	Current Charter
Other Charters	Details # ? x

Metrics summary

Session sheets of the past testing

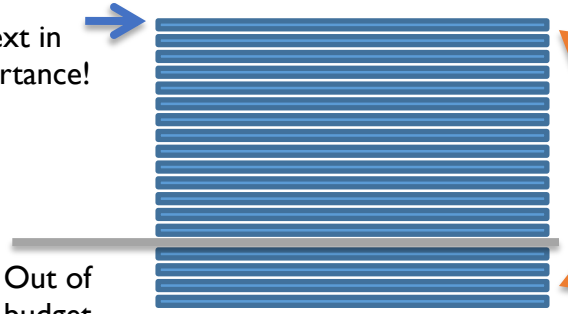


Playbooks

Coverage outlines

Charter backlog of the future testing

Next in importance!



Out of budget

Idea of exploration

Bug Reports

The Pieces in Management Framework

- A disciplined tester replanning on various levels
- Session with charter that provides a report
- Classification of information created as metrics
- Prioritizing of what test idea comes next
- Supporting reporting by debriefing
- Supporting skills development by coaching or pairing
- Creating a combined judgement of quality by quality reporting

Gilded Rose

By Emily Bache

(available in 32 languages…)

https://github.com/emilybache/GildedRose-Refactoring-Kata

Gilded Rose Requirements Specification
==========

Hi and welcome to team Gilded Rose. As you know, we are a small inn with a prime location in a prominent city ran by a friendly innkeeper named Allison. We also buy and sell only the finest goods.
Unfortunately, our **goods are constantly degrading in quality as they approach their sell by date**. We have a **system in place that updates our inventory for us.** It was developed by a no-nonsense type named Leeroy, who has moved on to new adventures.
First an introduction to our system:

- All items have a SellIn value which denotes the number of days we have to sell the item
- All items have a Quality value which denotes how valuable the item is
- At the end of each day our system lowers both values for every item

Pretty simple, right? Well this is where it gets interesting:

- Once the sell by date has passed, Quality degrades twice as fast
- The Quality of an item is never negative
- "Aged Brie" actually increases in Quality the older it gets
- The Quality of an item is never more than 50
- "Sulfuras", being a legendary item, never has to be sold or decreases in Quality
- "Backstage passes", like aged brie, increases in Quality as its SellIn value approaches;
  Quality increases by 2 when there are 10 days or less and by 3 when there are 5 days or less but
  Quality drops to 0 after the concert

Just for clarification, an item can never have its Quality increase above 50, however "Sulfuras" is a legendary item and as such its Quality is 80 and it never alters.

How to Explore

- Do something with it
- Find out what is the common thing to do with it
- Find out what you could do with it

- Reflect after anything and everything – make notes
  - What do we know from other connections?
  - What do we know from empirical evidence?
  - How do we turn it all into empirical evidence?

REST API

Nouns

Verbs

"CRUD"

Methods

HTTP GET
HTTP POST
HTTP PUT
HTTP DELETE
HTTP PATCH

Status Codes

1xx: Informational
2xx: Success
3xx: Redirection
4xx: Client Error
5xx: Server Error

Verbs

Authorization/authentication

Data

Errors

Responsiveness

http://qa-matters.com/2016/07/30/vader-a-rest-api-test-heuristic/

Patterns of Exploration for APIs

Lessons Distilled

Working with limited understanding: Focus!

Calls and Operations.
Inputs and Outputs.
Exceptions.

Target of your testing
vs. the environment
it depends on

Specific user with a specific purpose

Building it the First Time

Debugging and Maintaining

Why would anyone use this?

Think lifecycle

Versioning and Deprecation

Google for Concepts

like “Conservative Overloading Strategy”

Fast-track your understanding: collaborate

Documentation Matters a Lot for APIs

Explore to create documentation

Some patterns become visible with repetition

Existing automation allows for exploration reach

Creating automation forces exploration of details

Failing automation is
invitation to explore