Intro

• Physics / Computer Science, Novosibirsk State University, Russia
• BS, MS in Computer Science from Purdue University
• Medtronic – Firmware Development / Test Automation: ~10 years
• Parasoft – Solution Architect. 5+ years (current)
• Maple Grove, MN
• Three children + one wife + one dog
• Hobbies: Soccer, Martial Arts, Children and Creativity
Before we start a serious talk

• https://www.youtube.com/watch?v=foncl5pEYxg
Why IoT?

Everything is connected, but why?
The Internet of Things
From connecting devices to human value

01 Device connection
- IoT devices
- IoT connectivity
- Embedded intelligence

02 Data sensing
- Capture data
- Sensors and tags
- Storage

03 Communication
- Focus on access
- Networks, cloud, edge
- Data transport

04 Data analytics
- Big data analysis
- AI and cognitive
- Analysis at the edge

05 Data value
- Analysis to action
- APIs and processes
- Actionable intelligence

06 Human value
- Smart applications
- Stakeholder benefits
- Tangible benefits

THE INTERNET OF THINGS
AN EXPLOSION OF CONNECTED POSSIBILITY

2020
50.1 BILLION

2018
42.1 BILLION

2017
28.4 BILLION

2016
18.2 BILLION

2015
11.2 BILLION

2013
8.7 BILLION

2012
2.9 BILLION

2011
1.2 BILLION

2010
34.8 BILLION

2010
22.9 BILLION

2010
14.4 BILLION

2009
0.5 BILLION

2009
IoT INCEPTION

2009
1,000,000

2002
100,000

2002
Inception of the population of the Internet

"Taking population projections into account, there will be almost 50 billion devices on the planet in 2020."

"Due to the Internet of Things, every single thing around us will have a connection to the Internet and be aware of the products and services around it.

"The Internet of Things is not just about connecting your smartphone to your fridge, but about connecting all the smart devices in your home and making your life easier.

"In 2010, there were an estimated 22.9 billion devices connected to the Internet, compared to 2009's 0.5 billion.

"The growth of the Internet of Things has been exponential, with the number of connected devices increasing every year.

"The Internet of Things has the potential to revolutionize the way we live and work, making our lives easier and more convenient.

"As of 2010, there were 34.8 billion devices connected to the Internet, more than double the 2009 figure.

"The Internet of Things has the potential to change the way we live and work, making our lives easier and more convenient.

"In 2009, there were an estimated 1,000,000 devices connected to the Internet, compared to 2002's 100,000.

"The growth of the Internet of Things has been exponential, with the number of connected devices increasing every year.

"The Internet of Things has the potential to change the way we live and work, making our lives easier and more convenient."
Why does IoT testing start to matter?

- Connected “things” become integral part of our LIFE
- Basic Human Needs: Safety & Security

![Diagram of Maslow's Hierarchy of Needs with IoT devices]
The Three Parts of an IoT Solution

---

**Edge**
- Sensors and Actuators
- Endpoints and Gateways
- Decision Models

**IoT Platform**
- Data Ingestion
- Data Analytics
- Policy and Orchestration
- Device and Platform Management

**Enterprise**
- Business Applications
- Business Processes
- Data Super Source

---

© 2018 Gartner, Inc. and/or its affiliates. All rights reserved.
Talk outline – high level

Goal: Interactive information exchange

• IoT Introduction
• IoT Edge is moving into safety critical domain
• Embedded (Edge) testing
• IoT Platform and Enterprise testing
• Unique trends in Testing IoT
• Q&A
Embedded Testing: Testing the Edge

What is specific to IoT domain?

• Future proof for functional safety

• Application Security

• CI/CD integration
API Testing: IoT Platform and Enterprise layers

• Additional communication protocols
• DevOps implementation in heterogeneous test environments
• Test Data Management
• Who should test APIs?
• API’s are shared across domains: Security
• API Simulations
Internet Of Things: Definition and Trends
Future: Connecting people with services
**Industry 4.0**

- **Industry 4.0** is a name for the current trend of automation and data exchange in manufacturing technologies.

- It includes **cyber-physical systems**, the **Internet of things**, **cloud computing** and **cognitive computing**.

- Industry 4.0 is commonly referred to as the **fourth industrial revolution**.
Autonomous Vehicles
Driving Transition to Transportation-as-a-Service

2015
$2.25 TRILLION
US CONSUMER TRANSPORTATION

$2.15 TRILLION
CAR OWNERSHIP

$63 BILLION
PUBLIC TRANSPORTATION

$36 BILLION
RENTAL CARS

$12 BILLION
TAXIS & LIMO

$1 BILLION
BLACK CARS

RIDE-SHARING

Future State

CAR OWNERSHIP

TRANSPORTATION AS A SERVICE

Simpistic example of a connected automotive system.

Car

Hub: QNX / C++ / Android Java

- Infotainment
- MQTT
- Main Controller
- CAN bus

- Sensors
- Embedded Systems (barebone C)
- Controllers

REST over HTTP(s)

V2V, V2X

cloud(s)
The Three Parts of an IoT Solution

<table>
<thead>
<tr>
<th>Edge</th>
<th>IoT Platform</th>
<th>Enterprise</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensors and Actuators</td>
<td>Data Ingestion</td>
<td>Business Applications</td>
</tr>
<tr>
<td>Endpoints and Gateways</td>
<td>Data Analytics</td>
<td>Business Processes</td>
</tr>
<tr>
<td>Decision Models</td>
<td>Policy and Orchestration</td>
<td>Data Super Source</td>
</tr>
<tr>
<td></td>
<td>Device and Platform Management</td>
<td></td>
</tr>
</tbody>
</table>
Trends in IoT Testing

• Edge is moving into safety critical domain.
• End To End Testing and Debugging is expensive
• Focus on interoperability and API testing
• Embedded and IT must coexist
• Security
But - Is IoT testing anything special?

• Back to the fundamentals of test automation

• In 2012 Martin Fowler wrote about the test pyramid, this was a concept developed by Mike Coen.

https://medium.com/@timothy.coehran/test-pyramid-the-key-to-good-automated-test-strategy-gf3y3c02d5
Building a solid Testing Pyramid in IoT

- Minimize brittle end to end and UI driven tests, focus on highly automated API centric testing
- Focus on API testing and integrate into the CI pipeline
- Establish a solid foundation of early stage Unit Tests
IoT: Testing on the Edge
Continuous Testing Pipeline

- Policy Management: Requirements Defined
- Development
- Defect Prevention
- Development Testing
- Static Analysis
- Unit/Component
- Peer Review
- Code Coverage

- Integration Testing
  - Automated Tests
    - API/Service Tests
    - Smoke Test
    - Security Tests

- System Testing
  - Automated Tests
    - Functional Tests
    - Scenario Tests
    - Performance Tests

Test Environment Access

Business Decision
- Go
- Release Path

Defect Remediation Tasks

Go

No Go

Release Path
Part 1: Development Testing practices

![Car Diagram]

- Hub: QNX / C++ / Android Java
  - Infotainment
  - Main Controller
  - Sensors
  - Embedded Systems (barebone C)
  - Controllers

- Static Analysis
- Unit Testing
- Integration Testing
- Code Review
- Functional Safety Compliance
- Safety and Security
- ...

**Embedded Systems**

- **Sensors**
- **Controllers**
The story of Shift Left

Find detects earlier, save money!
Functional Safety Standards

- MEDICAL [IEC 62304]
- PROCESS INDUSTRY [IEC 61511]
- AUTOMOTIVE [ISO 26262]
- NUCLEAR [IEC 60880, IEC 60987, IEC 61226]
- TRANSPORTATION [EN 50126, EN 50128, EN 50129]
- AGRICULTURE [ISO 25119]
Future proof your application for compliance

A real use case in an automotive market

- A vision sensor was developed and tested as a prototype
- Major issues were fixed and startup was sold to Tier 1 supplier
- Manufacturer requires ISO 26262 compliance

The question:

How do I get to compliance when all the code is written and operational?
The Answer

screwed!
ISO 26262: ASIL Levels and Safety

The risk of safety impacts the rigor that must be followed

- Autonomous Vehicles
  - No Driver

- Advanced Driver Assistance
  - Software ADAS
  - Driver assist - Driver in the Loop

- Controls and Infotainment
  - Driver full Control

RISK

- ASIL D
- ASIL C
- ASIC B
- ASIL A

Cost of Compliance
ISO 26262 – the V diagram

These parts of the model are detailed in Section 6
5.4.7 To support the correctness of the design and implementation, the design and coding guidelines for the modelling, or programming languages, shall address the topics listed in Table 1.

NOTE 1 Coding guidelines are usually different for different programming languages.

NOTE 2 Coding guidelines can be different for model-based development.

NOTE 3 Existing coding guidelines can be modified for a specific item development.


Table 1 — Topics to be covered by modelling and coding guidelines

<table>
<thead>
<tr>
<th>Topics</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a Enforcement of low complexity(^a)</td>
<td>++ ++ ++ ++</td>
</tr>
<tr>
<td>1b Use of language subsets(^b)</td>
<td>++ ++ ++ ++</td>
</tr>
<tr>
<td>1c Enforcement of strong typing(^c)</td>
<td>++ ++ ++ ++</td>
</tr>
<tr>
<td>1d Use of defensive implementation techniques</td>
<td>o + ++ ++</td>
</tr>
<tr>
<td>1e Use of established design principles</td>
<td>+ + + ++</td>
</tr>
<tr>
<td>1f Use of unambiguous graphical representation</td>
<td>+ ++ ++ ++</td>
</tr>
<tr>
<td>1g Use of style guides</td>
<td>+ ++ ++ ++</td>
</tr>
<tr>
<td>1h Use of naming conventions</td>
<td>++ ++ ++ ++</td>
</tr>
</tbody>
</table>

\(^a\) An appropriate compromise of this topic with other methods in this part of ISO 26262 may be required.

\(^b\) The objectives of method 1b are
- Exclusion of ambiguously defined language constructs which may be interpreted differently by different modellers, programmers, code generators or compilers.
- Exclusion of language constructs which from experience easily lead to mistakes, for example assignments in conditions or identical naming of local and global variables.
- Exclusion of language constructs which could result in unhandled run-time errors.

\(^c\) The objective of method 1c is to impose principles of strong typing where these are not inherent in the language.
Coding Standards and Guidelines

- MISRA C 2012
- CERT C/C++, JAVA (Computer Emergency Readiness Team)
- JSF
- UL 2900
- AUTOSAR
- CWE –SANS, TOP 25
- High Integrity C++
- MISRA C++ 2008
- OWASP Top 10
- Barr Group Coding Standard
- Effective C++ (Scott Meyers)
- Power of Ten
MISRA compliance framework

Defined by MISRA 2016 Compliance Document

- **Guideline Enforcement Plan** - demonstrates how each MISRA guideline is verified

- **Guideline Re-categorization Plan** - is used to communicate the agreed upon importance for a guidelines as part of the vendor/client relationship.

- **Deviations report** - documents the deviated violations of guidelines (i.e. exceptions to while the violation will not be remediated) with appropriate rationale.

- **Guidelines compliance summary** - is the primary record of overall project compliance.
Embedded Security: What can we realistically do with SA?

Example - CERT C.

- Invoking unsafe macros with invalid arguments. (CERT C, Rule 01)
- Conversions on integers (CERT C, Rule 04)
- Protecting information leakage from trusted (kernel) to untrusted (application) level space. (CERT C, Rule 02)
- Using data outside of declared object’s storage durations (CERT C, Rule 02)
- Avoid accessing arrays out of bounds (CERT C, Rule 06)
- Heap and Stack usage violations (CERT C, Rule 08)
- Handle standard library errors (CERT C, Rule 12)
- Passing a non-null terminated string to a function that expects a string (CERT C, Rule 07)
- Avoid multithreading issues: deadlocks, priority inversion, race conditions (CERT C Rule 14)
- Ensure correct pointer arithmetic (CERT C, Rule 04)
- General rules to eliminate undefined behaviors that can lead to exploitable vulnerabilities.
Static Analysis Adoption

• False Positives rate
• Running slow
• “Don’t see value is fixing the violations. My code works fine. “
• “Too busy focusing on real issues”
• Commercial tools are expensive, open source tools are noisy.
• “Takes too much time to analyze the problems reported”
Practical approaches for Static Analysis for IoT

- Integrate Static Analysis practice into CI/CD workflow
- Create a baseline.
- Start small and solidify and issue resolution workflow.
- Focus on deltas between previous build and your changes.
- Integrate Static Analysis suppression review with Code Review workflow
- Integrate with development workflow (IDE, command line)
Static Analysis Tools

The IoT Hall-of-shame

9.4.3 The software unit testing methods listed in Table 10 shall be applied to demonstrate that the software units achieve:

a) compliance with the software unit design specification (in accordance with Clause 8);

b) compliance with the specification of the hardware-software interface (in accordance with ISO 26262-5:2011, 6.4.10);

c) the specified functionality;

d) confidence in the absence of unintended functionality;

e) robustness; and

EXAMPLE The absence of inaccessible software, the effectiveness of error detection and error handling mechanisms.

f) sufficient resources to support their functionality.

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>1a</td>
<td>++</td>
</tr>
<tr>
<td>1b</td>
<td>++</td>
</tr>
<tr>
<td>1c</td>
<td>+</td>
</tr>
<tr>
<td>1d</td>
<td>+</td>
</tr>
<tr>
<td>1e</td>
<td>+</td>
</tr>
</tbody>
</table>

a) The software requirements at the unit level are the basis for this requirements-based test.

b) This includes injection of arbitrary faults (e.g. by corrupting values of variables, by introducing code mutations, or by corrupting values of CPU registers).

c) Some aspects of the resource usage test can only be evaluated properly when the software unit tests are executed on the target hardware or if the emulator for the target processor supports resource usage tests.

d) This method requires a model that can simulate the functionality of the software units. Here, the model and code are stimulated in the same way and results compared with each other.
Unit Testing

```c
int16_t testThisFunction(uint32_t *myArray, size_t array_size)
{
    int16_t retVal = -1;
    // uint8_t checkedArraySize;
    char *stringPtr = NULL;
    FILE *fb;

    if (myArray != NULL)
    {
        // limit to MAX_ARRAY_LIMIT if larger than
        if (array_size > MAX_ARRAY_LIMIT)
        {
            array_size = MAX_ARRAY_LIMIT;
        }

        if (NULL != (fb = fopen("test.txt","w+")))
        {
            stringPtr = convertArrayToStr(myArray, array_size);
            // check return contents are not NULL
            if (stringPtr != NULL)
            {
                if (strlen(stringPtr) == fwrite(stringPtr, sizeof(char), strlen(stringPtr), fb))
                {
                    retVal = array_size;
                }
                freeString(stringPtr);
            }
            fclose(fb);
        }
    }

    return retVal;
}
```
Unit testing in real life

Industry adoption outside of safety critical industry is low. Why?

• “Unit Testing is useless – it does not find real issues”

• “I don’t have time to create and maintain unit tests”

• “There is no internal quality policy to create unit tests. It’s a best practice that I do when I have time. But instead, I’d rather spend time being productive and add more product features that customers will benefit from.”

• “We pay our developers to develop, not to test. We are outsourcing testing to save money to be able to pay the developers top salary they demand”

• “We’d rather spend time and money creating more top level tests to better verify integrated components”
The process of creating unit test is beneficial by itself

- Developer looks at the code through difference lenses → additional code review
- Review the interface to the functionality from external point of view
  - How will my code be used? → simplify the interface
  - What if I get invalid data? → adding error handling
- Revisit the interactions of the UAT to other components
  - Simulate fault injections
  - Test negative / positive responses from dependencies
- **Decide on boundaries of a Unit**
  - Only create meaningful tests
  - Easier traceability to Requirements and User Stories → reduces # of functional test cases
Unit Testing for Embedded: Host Vs Target

• **Option 1:** Compile with host compiler (gcc, visual studio) and execute on x86 (**host**)

  Pros:
  - Fast execution
  - Easier integration into CI/CD pipeline
  - Easier setup in some cases

  Cons:
  - Requires stubbing of hardware drivers
  - Different compiler and architecture. (harder to build functional safety case)
  - Need to maintain multiple projects
Unit Testing for Embedded: Host Vs Target

- **Option 2:** Compile unit tests with cross compiler and execute on **target**

**Pros:**
- Testing is more realistic – easier to qualify for FuSa
- Can test HW drivers

**Cons:**
- Requires a dev board
- Harder or impossible to run in CI/CD pipeline
- Harder to setup for some boards and communication channels
- Requires cross compiler license and debugger interface
Unit Testing for Embedded: Run on simulator!

- **Solution:** Compile unit tests with cross compiler and execute on **simulator**
- Run / debug tests on simulator.
- Integrate with CI/CD
- Re-run the same tests on board for formal verification

Simulators:
- QEMU (Open Source), Fast Models and Keil (ARM), IAR, WindRiver VxWorks sim, GreenHills,
Code Coverage Story

C/C++ Code Coverage metrics

- [x] Function coverage   [ ] Function hit count
- [x] Line coverage       [ ] Line hit count
- [ ] Statement coverage
- [ ] Block coverage
- [ ] Path coverage
- [ ] Decision coverage
- [ ] Simple condition coverage
- [x] MC/DC coverage
- [x] Call coverage
Unit Testing – stubs to help drive code coverage

```c
int16_t testThisFunction(uint32_t *myArray, size_t array_size)
{
    int16_t retVal = -1;
    // uint8_t checkedArraySize;
    char *stringPtr = NULL;
    FILE *fb;

    if (myArray != NULL)
    {
        // limit to MAX_ARRAY_LIMIT if larger than
        if (array_size > MAX_ARRAY_LIMIT)
        {
            array_size = MAX_ARRAY_LIMIT;
        }

        if (NULL != (fb = fopen("test.txt","w+")))
        {
            stringPtr = convertArrayToStr(myArray, array_size);
            // check return contents are not NULL
            if (stringPtr != NULL)
            {
                if(strlen(stringPtr) == fwrite(stringPtr, sizeof(char), strlen(stringPtr), fb))
                {
                    retVal = array_size;
                }
                freeString(stringPtr);
            }
            fclose(fb);
        }
        return retVal;
    }
```
Unit Testing: Popular frameworks and tools

• Open Source frameworks:
  • Google test
  • Unity
  • cppUTest
  • Boost

• Unit Testing tools for **functional safety:**
  • Parasoft C/C++test
  • LDRA
  • Cantata
  • VectorCast
Summary

• Identify potential Risks and Safety Integrity Level (SIL)
• Isolate safety critical functionality in files, modules, libraries, applications
• Establish and follow Coding Standards
• Run Static Analysis
• Establish Unit Testing baseline
• Test algorithmic functions at lower level
• Establish Code Review workflow
• Require frequent check ins of code changes.
• Integrate Static Analysis and Unit/Regression testing into your CI/CD pipeline
• Focus on isolating and testing components: e.g. RTOS Tasks, logical modules, ISR’s.
IoT API testing
Testing Platform and Enterprise layers
API Testing: IoT Platform and Enterprise layers

• Additional communication protocols
• DevOps implementation in heterogeneous test environments
• Test Data Management
• Who should test APIs?
• API’s are shared across domains: Security
• API Simulations
IOT Communication Stack

Focus on API Testing and Test Environments

- **Car**
  - Verify Infotainment: API: MQTT, REST
  - Verify Car: API: REST, V2V, V2X, CAN bus
  - Verify Sensors: API: CAN

- **Main Controller**
  - MQTT

- **Cloud(s)**
  - REST over HTTP(s)
  - V2V, V2X

- **IoT Platform**
  - Verify IoT platform: API: REST, V2V, V2X

- **Sensors**

- **Controllers**

- **Cloud(s)**
RESTful API (synchronous)
MQTT – publish/subscribe (asynchronous)

Continuous Testing Pipeline

- **Policy Management**
  - Requirements Defined

- **Defect Prevention**
  - Development

- **Development Testing**
  - Development
  - Static Analysis
  - Unit/Component
  - Peer Review
  - Code Coverage

- **Integration Testing**
  - Automated Tests
    - API/Service Tests
    - Security Tests
    - Smoke Test

- **System Testing**
  - Automated Tests
    - Functional Tests
    - Scenario Tests
    - Performance Tests

- **Test Environment Access**

- **Defect Remediation Tasks**

- **CI Build**

- **Go**
  - Release Path
  - Business Decision

- **No Go**
Challenge – how to isolate Application Under Test?

Divide-and-Conquer

- The whole problem we want to solve may too big to understand or solve at once.
- We break it up into smaller pieces, solve the pieces separately, and combine the separate pieces together.

- Divide-and-conquer is a general algorithm design paradigm:
  1. **Divide**: divide the input data \( S \) in two or more disjoint subsets \( S_1, S_2, \ldots \)
  2. **Recur**: solve the subproblems recursively
  3. **Conquer**: combine the solutions for \( S_1, S_2, \ldots \), into a solution for \( S \)

- The base case for the recursion are subproblems of constant size
- Analysis can be done using **recurrence equations**
Virtual Services

- Simulate response payload
- Can respond with realistic data and simulate delays
- **Lightweight, can be deployed in application container (e.g. Docker), reusable**
- Use Case: Integration Testing
- **NOT Virtual Machines with real services deployed.**
IoT: Increasing complexity of testing requirements
Omni/Multi-Channel Testing
Omni/Multi-Channel Testing

Parasoft SOATest

Available and fee-based 3rd party systems

Unavailable or fee-based 3rd party systems

Uncontrollable behavior

“Agile Roadblock”

Unable to ‘shift-left’ performance testing

Application Under Test
Total control of the Test Environment

Parasoft SOATest

Application Under Test

Parasoft Virtualize

- 500 Internal Server Error
- Malformed Response
- Expose a security Exception
- Test the boundaries of performance SLAs

Mobile

Web

API

SQL/NoSQL

Test Data
Digital Transformation applied to an IoT Ecosystem
What am I testing (Integration Layer)
Test to Edge
Test to Edge
Simulate Cloud

Edge Computing

AUT

Industrial PC

Gateway

Micro Data Center

Cloud
Simulate Cloud
Simulate Edge
Isolation enables DevOps: Testing on Demand
Testing a component of IoT system
API Simulators

• Market: API providers and API consumers
• API providers make development kits for the API (e.g. AWS, Azure)
• API consumers have access to API simulators
• API consumers develop tests against simulators.
• API providers can validate and certify correct API usage by API consumers
Car API Simulator - example

https://car-simulator.developer.mercedes-benz.com/orgs/5vzj/emulators/483D1C27D27A39132C#/

![Car API Simulator](https://car-simulator.developer.mercedes-benz.com/orgs/5vzj/emulators/483D1C27D27A39132C#/)
## IoT Platform and Enterprise testing

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comprehensive protocol stack</td>
<td>• HTTP, MQTT, RabbitMQ, JMS, XML, JSON, REST, SOAP, ... many more</td>
</tr>
<tr>
<td>Interoperability between Dev and QA</td>
<td>• Complex test environment require programming skills</td>
</tr>
<tr>
<td>Security and Performance Testing into CI/CD</td>
<td>• Run early stage performance tests leveraging functional tests created by the team</td>
</tr>
<tr>
<td>End-To-End testing</td>
<td>• Combine API, Web, Mobile and DB interactions into one test scenario</td>
</tr>
<tr>
<td>Integration with DevOps</td>
<td>• Deploy and Destroy Test Environment model</td>
</tr>
</tbody>
</table>
Key takeaways

• Focus on API Testing
• Make API’s simple and stateless
• Create reusable virtual services to isolate components under test
• Define who owns API testing domain
• Integrate with CI/CD
• Perform security and pen testing earlier at component level
IoT: Embedded and Cloud
Similarities between Enterprise IT and Embedded/IoT

- Functionalities spanning across many layers
- Open Architecture
- 24x7 Connectivity
- Constant Security threat
What can Embedded learn from IT?

• Connected embedded systems resemble architectures known in IT

• Complexity stemming from interconnections shifts the gravity in testing

• Embedded Left-focus strategy no longer optimal
Meet in the middle?

Embedded development
• Focus on defect prevention and early detection
• Unit Testing, Software Coding Guidelines
• Minimize testing fully assembled integrated systems
• Avoid debugging of integrated systems
• Focusing on compliance, not speed

Enterprise IT development
• Focus on agile, rapid development
• Volume of code too large to widely apply unit testing
• Automated component/API level testing
• Testing and debugging integrated solutions
Continuous Testing Pipeline

- Policy Management: Requirements Defined
- Defect Prevention: Development
- Development Testing: Development
  - Static Analysis
  - Unit/Component
  - Peer Review
  - Code Coverage
- Integration Testing: Automated Tests
  - API/Service Tests
  - Smoke Test
  - Security Tests
- System Testing: Automated Tests
  - Functional Tests
  - Scenario Tests
  - Performance Tests

Test Environment Access

- CI Build: Go
- Business Decision: No Go
- Release Path

Defect Remediation Tasks

- Bamboo
- TeamCity
- Parasoft
Wrap up
Key takeaways

- Prepare for compliance
- Focus on creating clean simple and testable API’s.
- API Testing is the key.
- IoT testing: Safety and Security is the focus of the future to avoid crisis
- Automated testing using simulations to enable DevOps
- Service Virtualization to help isolate components under test
- Understand the cultural differences
- Test early to avoid debugging costs at later stages.
- Integrate all testing phases into one CI/CD pipeline
- Don’t try to do all of the above all at once.
HOW PARASOFT SUPPORTS THE SDLC

**Developer**
- Focused on innovation and minimizing testing burden
  - Easy, quick, and integrated
  - Minimize overhead
  - Get back to development

**QA / Tester**
- Focused on eliminating defects and reducing risk
  - Limited time + complex testing problem
  - Test Coverage, Security, Performance

**Release Mgmt**
- Focused on assessing risk and application readiness

- Static Code Analysis
- Unit Testing
- Coverage & Traceability
- API Testing & Service Virtualization
- Load and Performance Testing
- Reporting & Analytics
- Jtest, C/C++test, dotTEST, Insure++
- SOAtest, Virtualize
- Load Test
- Parasoft DTP
Comprehensive Testing Solutions for the Entire SDLC

Developer
- Static Analysis
- Runtime Analysis
- Unit Testing

Tester/QA
- Functional & Non-Functional Testing
- Service Virtualization

Management
- Management & Analytics

Testing Challenges
- Security
- Scalability
- Reliability
- Quality
- Safety
- Risk

SDLC Integration
- Jenkins
- Bamboo
- TeamCity
- VSTS
- JIRA / RALLY
- HP / ALM
- GIT
- SVN

Strategic Ecosystem
- Microsoft
- HP
- AWS
- Selenium
- BURPSUITE
- appium
- Azure

Financial Services Clients
- Charles Schwab
- ING
- Capital One
- Barclays
- Bank of America
- Fidelity
Thank you!

Contact information:

andrey.madan@parasoft.com
https://www.linkedin.com/in/andreymadan/