IoT Development and Testing

Part 5:
IoT Tools, Automation, and Environments:
IoT Test Architecture to Lab Creation

Early Draft Version

Jon Duncan Hagar

2018

This is an online eBook. I do not plan a published hardcopy. It is part of a series where I am keeping costs low to make it affordable for everyone. Early versions change frequently, and later I plan continuing updates as needed. I am always looking for comments and inputs. You can email or tweet those to me. I hope to see posts on social media about it.

Tweet: jonduncanhagar

Linkedin name: jonduncanhagar

Email: jon.d.hagar@gmail

Release Notes:

This version is 50% complete for now. TBD indicate areas that are not complete, underwork, and still in outline format.

Contents

Preface

I offer this series of eBooks to help teams moving into the Internet of Things (IoT) development and testing. The focus is on testing, but in IoT, I believe teams should not separate development (dev) and testing efforts from each other. My experience is that IoT teams must consider the hardware, software, system, and operations to be a great success. Further, teams coming from a particular engineering environment tend to focus on the familiar and have learning curves in other areas.

I have structured this series of eBooks to address these considerations yet be cost-effective. So instead of that one big expensive book, I opted to create small eBooks, following an agile approach, keeping costs low, putting materials online (eBooks), and seeking user feedback. To these ends, readers can buy the full bundle or opt for just one or two eBooks. The selection of eBooks depends on the reader's context.

Keeping with an Agile philosophy, I plan ongoing updates, so please contact me with ideas, improvements, and what you like. I also hope to have online or traditional training sessions using the eBooks. The exact format of the training is yet to be determined, but again, I hope to keep costs and time minimized. I wish to help teams learn about IoT as this part of the industry grows.

This IoT Dev and Test eBook series has the following structure to help readers:

Part 1: What is IoT, How is testing & development different or alike than other development & testing.

Part 2: IoT Development and Test – A general high-level introduction for teams starting to work with IoT development (dev) and test, containing references to more details.

Part 3: IoT test planning and strategy – A basic introduction and starting point for test planning covering the informal to formal levels. Part 3 provides a start for beginners and contains tidbits for the experienced test manager.

Part 4: IoT test design and security – An eBook that contains test design and implementation details specific to the IoT environment. Test levels move from white box to black box testing as well as non-functional types of testing. Part 4 is a guide for teams doing tests in small start-ups to a larger scale, higher risk IoT systems.

Part 5: IoT environments, architectures, and tools – This part is a necessary addition to the test design of part 4 addressing test labs, automation, domains, and support tools.

Finally, I expect that readers have some knowledge of standard or typical development lifecycles including software testing. No book can address all aspects of technology. I provide references, both traditional hardcopy books as well as online resources. To grow in knowledge and skill, readers must have a library of reference materials. I have such libraries, and this is one reason why some people regard me as an expert. I do not know everything, but I *do know* when and how to look up materials. A definition of expert that I like is, "An expert is a person who knows what they

do not know as well as how to go about learning the unknown." To this, readers should regard thoughts in the eBook, and for that matter many other references, with degrees of skeptics while seeking confirmation by experience and testing. Skepticism should be second nature to the development and test engineers. A saying comes to mind "trust but verify."
(ref https://en.wikipedia.org/wiki/Trust,_but_verify).

Acknowledgements

I would like to thank and recognize our many mentors, colleagues, and teachers, who have taught me so much through the years. There are ideas in this book from many of them, which I try to correctly cite throughout the book (I apologize if I miss some). We are standing on each other's shoulders. My most significant acknowledgement goes to my first reviewer and wife, Laura— software–systems and SQA geek. Without her, this book would not be possible. I hope the reader will use this eBook to make the world safer, more secure and fun.

I particularly would like to thank the following people who, over the years, have helped me in test thinking:

Cem Kaner

James Whittaker

Dot Graham

James Bach

Rex Black

Lisa Crispin

Lee Copeland

Laura Hagar

About the Author:

Jon Hagar is a senior tester with 40 years in software development and testing currently working with Grand Software Testing, LLC. He has supported software product design, integrity, integration, reliability, measurement, verification, validation, and testing on a variety of projects and software domains (environments). He has an M.S. degree in Computer Science with specialization in Software Engineering and Testing from Colorado State University and a B.S. Degree in Math with specialization in Civil Engineering and Software from Metropolitan State University of Denver, Colorado. Jon has worked in business analysis, systems, and software engineering areas, specializing in testing, verification, and validation. Projects, he has supported include the domains of embedded, mobile devices, IoT, and PC/IT systems as well as test lab and tool development.

Introduction to IoT Architecture and Environments

The other eBooks in this series have addressed IoT basic concepts, test planning, design, and approaches. The prior eBooks addressed the concepts of automation, environments, tooling, and aspects of the cyber-physical system which are IoT. Generally, I will refer to these collectively in the eBook as the test/V&V environment and architecture. I find it essential to introduce the idea that each level of testing has different environments and architectures, which means the test tools, type of automation, and characteristics of the Cyber-Physical Systems (CPS) are different. Indeed, other types of software have these differences, but IoT systems magnify and expand these differences. Of particular importance for IoT is the validation of the device in the real world or realistic environments.

It may be tempting to an organization save the test/V&V until deployment release and only do real-world testing. However, experience has shown this will delay the official release of the product and may increase costs because of rework associated with a problem found late in the lifecycle. Hence, most projects take an incremental approach to test/V&V were there are multiple levels of activities. In fact, many of us have started test/V&V during the proposal stage to help "prove" ideas and concepts. Early testing can be less formal and done at a lower level, which then feeds into later V&V efforts ultimately concluding in real-world field testing.

Since the real world of IoT is infinite, this eBook cannot address every possibility of test environments and architectures. I have tried for some general topology classification, but readers will have to extrapolate to the local environments.

NIST (National Institue of Standard and Technology US government) defines "cyber-physical systems (CPSs) or "smart" systems as "co-engineered interacting networks of physical and computational components." CPS technologies include

• the IoT,

• the Industrial Internet of things (IIoT),

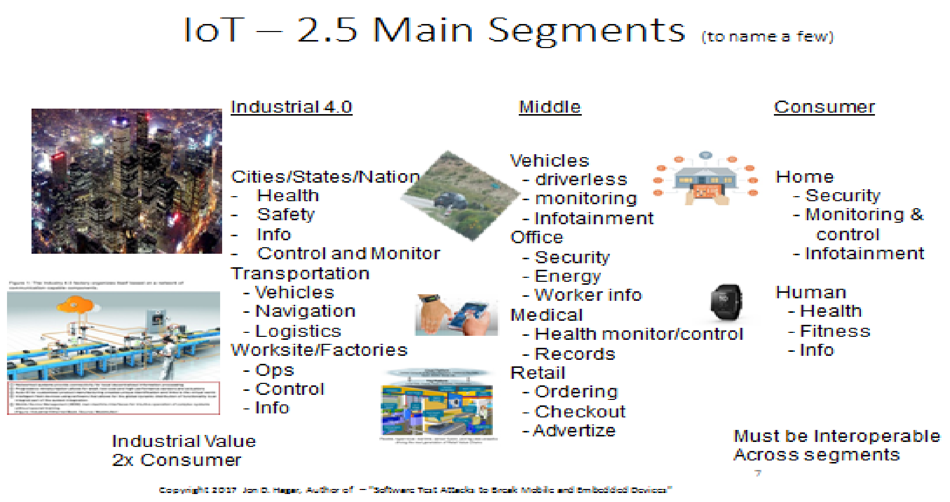
• smart cities (which is made up of IoT and IIoT),

• smart grids (ditto), and

• "smart" anything (for example, cars, buildings, homes, manufacturing, hospitals, and appliances).

Since CPS function in the real world software tester will need to step outside of the smaller (not simpler) cyber test space. The testers of IoT expand into hardware, physical interaction, operations, and all the varied environments found in the universe.

The NIST CPS is one view of IoT/IIoT, and another view is shown below. It is expanded in later sections



IoT – 2.5 Main Segments (to name a few)

Industrial 4.0

Cities/States/Nation
- Health
- Safety
- Info
- Control and Monitor

Transportation
- Vehicles
- Navigation
- Logistics

Worksite/Factories
- Ops
- Control
- Info

Industrial Value
2x Consumer

Middle

Vehicles
- driverless
- monitoring
- Infotainment

Office
- Security
- Energy
- Worker info

Medical
- Health monitor/control
- Records

Retail
- Ordering
- Checkout
- Advertize

Consumer

Home
- Security
- Monitoring & control
- Infotainment

Human
- Health
- Fitness
- Info

Must be Interoperable
Across segments

Copyright 2017 Jon D. Hagar, Author of – "Software Test Attacks to Break Mobile and Embedded Devices"

Figure IoT/IIoT Segments (more detailed mind map later in eBook)

The fact that we are having problems with a classification topology extends into test environments and architectures.

The classification topology ranges from the familiar software structural code testing to what I will call field testing. Knowing when to stop testing is a familiar problem for testers. This problem grows as the usual boundaries of software testing disappear, and we move into the real world.

Many development organizations will likely determine a test strategy and plan which limits or eliminates many of these topologies of testing. This limitation is acceptable to me, as long as the information about the IoT devices defines the topology(s) that the IoT testing addressed. The limit may be done in product specifications, fine print, or user guides, but disclosure of such information will likely become common as IoT/IIoT mature. For example, the industry sees this already where a wearable device disclaims being a medical or safety system. I suspect that such disclaimer will be evaluated and then required by regulators.

What This Book Addresses

This eBook part addresses the area which has variation from IoT project to project and is critical to testing providing useful information. Many software testers will be familiar with setting up attest environment where the software can be executed. The environment can be a PC, mobile device, emulator, mainframe, or another set up which reflects where the software will be run. However, since IoT/IIoT systems are "out in the real world" as well as having many different hardware and operational configuration, the options for setting up the test environment is equally varied. This part examines some of the test environment and architecture options with associated considerations.

This eBook provides a starting point on IoT/IIoT assessment including:

1. Tooling to support the dev, verification, validation, and testing of hardware, software, and the overall system (but not specific vendors and tool names)

2. Dev/test environments and architectures at numerous (but not maybe all) levels

3. Security and privacy assessment considerations

4. Levels of test automation to consider

The ebook is intended to be used with other books on testing, assessment and development. I recommend that the reader have several of the books in the references section as no single book can address all aspects of testing, V&V, and software for IoT.

Audience

Readers benefitting the most from this eBook include:

- Architects including test architects (if there are such things out there)
- Senior testers
- Developers
- Test managers and leads
- Test automation and tool implementers

These people are not limited to software, but will likely include hardware and system staff. Further, I expect many environments, particularly for the test, to be considered “out of scope” from some vendor organization. To solve this problem, I expect companies, governments and industry consortiums to take on the task of “big picture” dev integration and test. We may call these groups Independent Verification and Validation (IV&V), certification bodies (think underwriters laboratory), company/government IoT tech (ITT) labs, or similar.

How these groups are formed, called and paid for is yet to be determined. However, in my experience in cyberspace, I have found these groups. They are called the test IT department in some organization or IV&V in others. They vary from large to small. Some do integration and testing. Some just do firefighting. Whatever these groups are called, the will benefit from this eBook.

How to use this eBook

The eBook can be skimmed or read end-to-end quickly. After an initial reading, sections that were not clear at first can be read in detail. I do not intend the book be read in detail from cover to cover, but by jumping around to topics of interest in IoT

The eBook itself is in some sense written in a front-end-reverse order. What I mean I start with general planning and funding since if you do not have money and time everything else is less important. However, to avoid looking like a waterfall, which most books and standards do, I then jump to V&V/testing, then work "backwards" through a lifecycle ending at architecture, but if you are skipping around, this is okay.

The table of contents can also be used to index into topics that are of interest to readers. I suggest the reader assess if they understand basic concepts found in Chapter 1 and 2. If a reader determines they understand the basics, then use the index or table of contents to find a specific topic of interest. I would suggest as testing of an IoT device and system progress that testers refer back to the book. I also request if a reader finds things missing or lacking that you contact us, as I plan revisions to the eBook rapidly following Agile concepts.

Environments and Architectures

Given that Internet of Things (IoT) devices are predicted to be everywhere, in everything, and used at many different architectural scales, problem testers will face is how to define a test architecture with an associated environment that is sufficient and comprehensive[1]. Currently, IoT testers and companies tend to localize testing considerations just to the edge or device [2]. This shallow focus can leave areas of large-scale IoT systems, e.g., a system-of-systems, factory, city or country, under tested [3, 4].

This situation leaves risks untested since teams do not fully understand the whole picture of device usage, the internet and the broader scope found beyond the device and an ill-informed test strategy. Further, given that the test industry does not fully understand test architecture [5], engineers will struggle to define IoT test architectures and environments completely to do their jobs. The nature of unintended consequence [6] from lack of test knowledge about proper IoT testing, will lead to increased likelihood of failures and challenges in the areas of security, reliability, interoperability, safety, functionality, use, and others [7], which are critical to ultimate customer-user satisfaction.

This paper outlines the need for IoT testing to be done at a larger scale with attention paid to proper architectures and environments. This research considers many elements of a test IoT architecture environment. The paper identifies existing complex test systems as examples of where IoT can start. The solutions to larger scale IoT test architectures should include existing concepts, standards, who conducts the testing, and comprehensive test environments. Solutions must be producible and use resources efficiently if IoT deployment at the larger scale is to be successful. Finally, the paper considers the need for further research to address future changes and implications of IoT testing.

One might expect the phrase "software architecture" "IoT test architectures" or "IoT test environment" to be defined on such websites as Wikipedia [3], IEEE Standards Association [4], or SEVocab [5]. While not without controversy about universal correctness, for me, these websites do not define these phrases or the concepts surrounding them very well.

A general internet search on software architecture yields over 5,000 hits, but the top sites do not offer much help and tend to represent a single author's viewpoint. While individual authors do have valid information, in this eBook I will use the following.

Definition of terms for this eBook

To start refining the definition of the phrase "software architecture" or "software test architecture," I find it is useful to take a step back and consider the historical usage of the word "architecture" from other sources. The oldest historical usage is from civil engineering and building construction. The Wiki website https://en.wikipedia.org/wiki/ gives the following definitions:

Architecture (civil engineering) –

is both the process and the product of planning, designing, and constructing buildings and other physical structures.

Environment – TBD dictionary

Test Environment –

The purpose of the test environment is to allow human testers to exercise new and changed code via either automated checks or non-automated techniques. After the developer accepts the new code and configurations through unit testing in the development environment, the items are moved to one or more test environments. Upon test failure, the test environment can remove the faulty code from the test platforms, contact the responsible developer, and provide detailed test and result logs. If all tests pass, the test environment or a continuous integration framework controlling the tests can automatically promote the code to the next deployment environment.

Different types of testing suggest different types of test environments, some or all of which may be virtualized to allow rapid, parallel testing to take place. For example, automated user interface tests may occur across several virtual operating systems and displays (real or virtual). Performance tests may require a normalized physical baseline hardware configuration so that that performance test results can be compared over time. Availability or durability testing may depend on failure simulators in virtual hardware and virtual networks.

Tests may be serial (one after the other) or parallel (some or all at once) depending on the sophistication of the test environment. A significant goal for agile and other high-productivity software development practices is reducing the time from software design or specification to delivery in production. Highly automated and parallelized test environments are essential contributors to rapid software development.

The test environment definition should be familiar to software testers. It is reasonable, and I will use in this eBook, but the best I find for architecture in Wikipedia is for the civil engineering industry.

The construction industry dates back thousands of years. For me, building architecture can result in functional and elegant structures. As a mathematician, I look for both in a solution space. As software testers, this history should be our starting point for the definition of the phrase, software test architecture.

To support the building construction industry, schools have specialized training for people wanting to become architects. In the practice of architecture, there are hand-offs between architects and other building engineering disciplines, e.g., structural, electrical, heat/air-conditioning, etc. Additionally, as the architecture progresses into the construction phases, it is common for the people doing the actual work to find and correct architectural "issues" in the implementation efforts.

Can you see how the above concepts and ideas might work in software and testing?

In Table III below, I present a definition from individual test practitioners and professor, Dr Nishi Yasuharu from the University of Electro-Communications, Japan. It is taken from conference materials [13].

Table III: Conference Presentation Definition

• Test architecture is a big picture of test design

– Test engineers have to grasp a big picture of test design because test cases increase over 100,000 cases and get much more complicated

– Test techniques and coverages cannot prevent large lacks of test cases though they can prevent small lacks of test cases

– Quality of test design depends more on total balance than the priority of each test case

• Test architecture is just architecture of test design

– In software testing domains, people confuse big pictures of test design and big pictures of test process or test management

» In software development, software architecture is not described in project plans though test architecture is described in test plans

– What kinds of tests you design should be before ordering of test cases

• Test architecture consists of "test viewpoints" and relationships of them

I like parts of Professor Yasuharu's definition as his usage is closer to the definition that I envision, but I think there is room for improvement by being more concise.

For example, if I change aspects of the Wikipedia "construction" industry definition, I like it better. I give these improvements in Table IV.

Table IV

Software Test Architecture *is the process(s) and the product(s) of planning, designing, and constructing tests done with supporting test structures.*

Note: supporting test structures include test: tools, environments, documentation, tooling, viewpoints, and analytics

For IoT systems, an essential test architectural support structure is environments. The website [8] gives the following definition of the computing environment.

Computing Environment (to support testing) - The overall structure within which a user, computer, or program operates.

This paper also uses the terms viewpoint and software user as defined below.

Viewpoint – In systems engineering, a viewpoint is a partitioning or restriction of concerns in a system. Adoption of a viewpoint is used so that issues in those aspects can be addressed separately. A good selection of viewpoints also partitions the design of the system into specific areas of expertise [9].

Software User – Typically in software, humans are the only users who interact with the software system, but in IoT, the user of the software is expanded to include hardware, interfaces and other systems. [4,10]

This eBook uses these terms and definitions.

Classification Models of Basic IoT Device Architectures

While this eBook series has testing as a primary focus, I believe tester (and developers) benefit from understand models of IoT device architectures. Even with a software focus, projects can build and test in a vacuum. Figures below present a few models from which we can view IoT architectures.

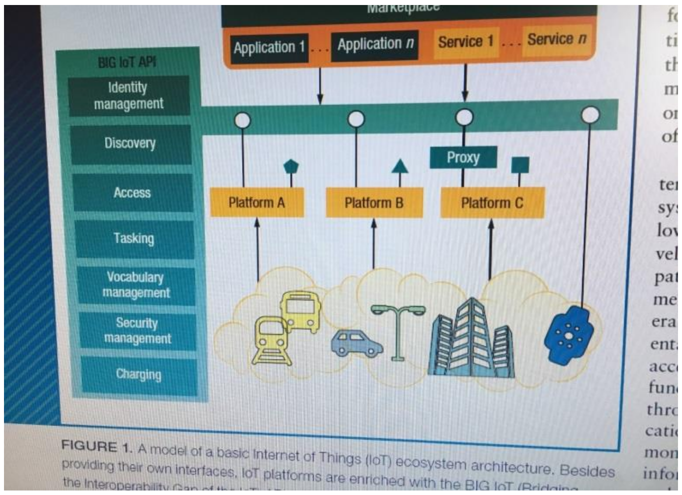
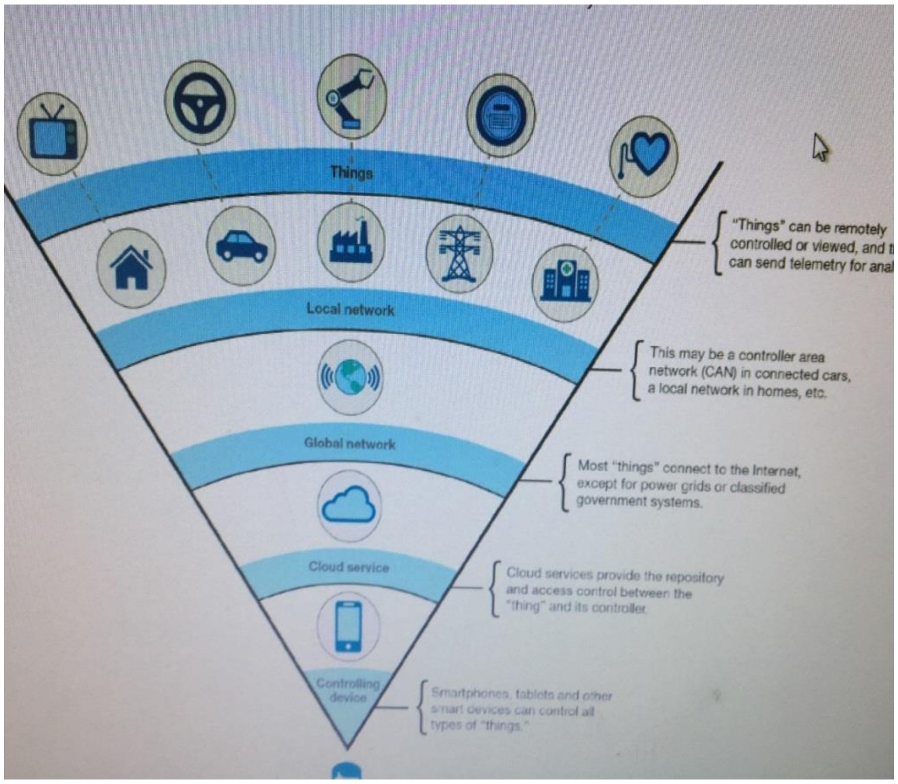
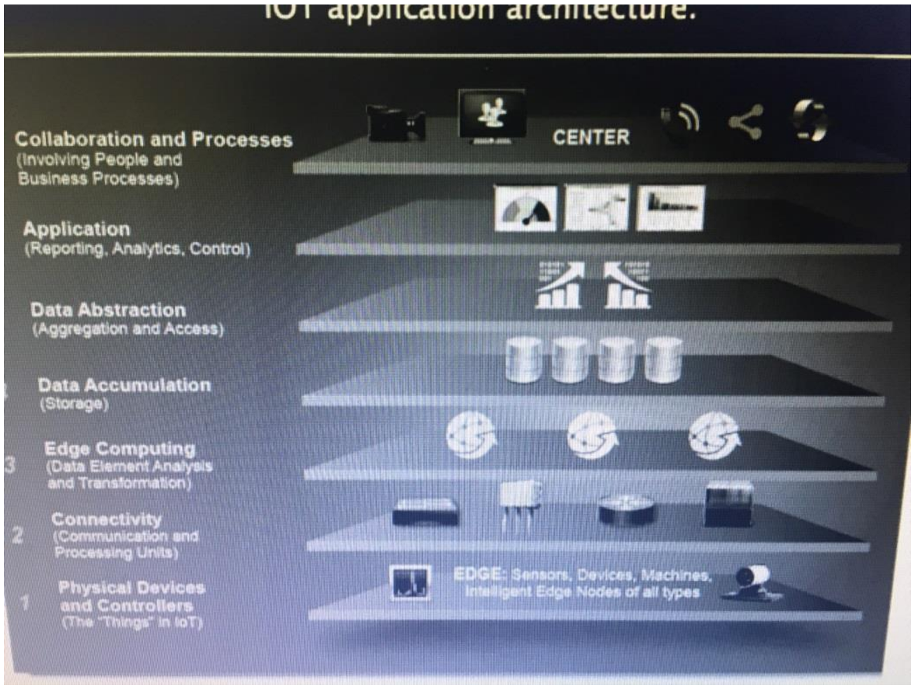


Figure class 1 –tbd ref


Things
"Things" can be remotely controlled or viewed, and t can send telemetry for anal
Local network
This may be a controller area network (CAN) in connected cars, a local network in homes, etc.
Global network
Most "things" connect to the Internet, except for power grids or classified government systems.
Cloud service
Cloud services provide the repository and access control between the "thing" and its controller.
Controlling device
Smartphones, tablets and other smart devices can control all types of "things."
Figure class 2 –tbd ref


IoT application architecture:
CENTER
Collaboration and Processes (Involving People and Business Processes)
Application (Reporting, Analytics, Control)
Data Abstraction (Aggregation and Access)
Data Accumulation (Storage)
3 Edge Computing (Data Element Analysis and Transformation)
2 Connectivity (Communication and Processing Units)
1 Physical Devices and Controllers (The "Things" in IoT)
EDGE: Sensors, Devices, Machines, Intelligent Edge Nodes of all types
Figure class 3 –tbd ref

A mistake software people make with IoT is focusing on development and test on small elements of the architecture. This mistake leaves problems that are from or cross these architectural boundaries undiscovered until the device goes into large scale use.

This book does not address the details of specific IoT/embedded system architecture. I expect readers will create their unique test architectures and environments as part of the test planning activities. If these figures do not make sense or for more information see the noted references.

Classification Map of IoT Device Architectures supporting Software Test

Figure 1 presents an IoT test architecture classification for IoT. It is only one possible classification but includes critical points for production and testing of IoT software systems. The test architecture should reflect aspects of the product's architecture that is experienced by users plus includes test support elements.

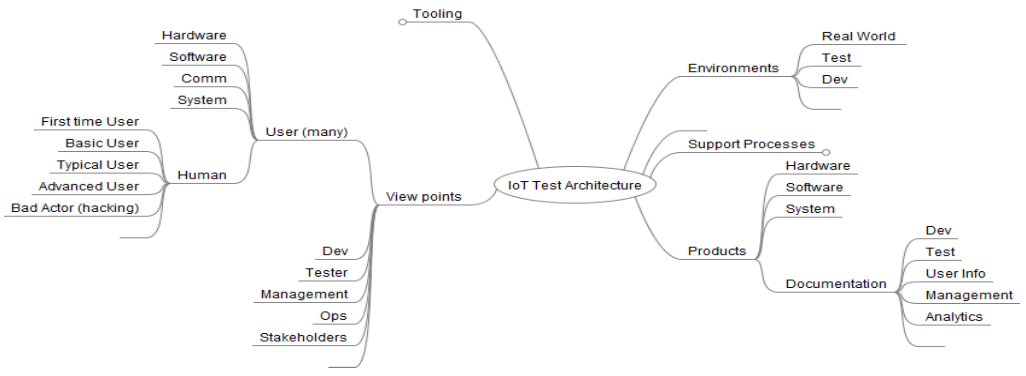


Figure: IoT Test Architecture Map [4]
Note: When a part of the map ends in a clear link, this usage indicates that other classification elements exist but were not included here to manage the size of the map for this paper.

In IoT, architecture product elements include unique hardware and software to produce a system. Further, under products, documentation supports software, hardware and the system. These elements are common to most cyber systems and thus need testing. However, for IoT, the differences are the scale, environments, real-time communications, and specialized hardware all of which need increasingly sophisticated software to operate. Thus, teams must establish viewpoints beyond the classic computer software system.

The viewpoints in this IoT test architecture are made up of users, development (Dev), testers, management, operations (Ops) and other stakeholders. Each of these viewpoints needs to become factors in testing. These factors include levels of testing, use-cases, risks, or other information a tester provides. Testing from the user's viewpoint is essential, but testers must also address the information associated with the other viewpoints, which can include management metrics-reports, stakeholder costs, Ops limitation, and what Dev may not have provided, e.g. structural testing.

The user classification includes human and non-human-users. Many testers experienced in Information Technology (IT) systems tend to focus on just the software and human user interface viewpoints. However, this leaves areas in IoT under tested. In IoT the non-human viewpoints become essential, and this includes hardware, other software, communication (comm) systems, and other systems that need to be integrated and tested. Finally, as the mind map figure shows, testers should not forget that the human user can have differences including viewpoints such as first, basic, typical, advance and a bad actor.

Many pictures see
https://www.google.com/search?q=internet+of+things+architecture+layers&sa=X&tbm=isch&tbo=u&source=univ&ved=0ahUKEwjV4LSd_qDYAhUBwmMKHZdLDpcQ7AkIgQE&biw=1186&bih=646

TBD Scale of the arch can be a problem for IoT

TBD Adopt a layered architecture – device/edge –network-cloud – app - Outside

Some currently widely used IoT reference architectures include:

- Internet of Things – Architecture (IoT-A): The IoT-A reference model and architecture was developed through an EU lighthouse project in 2013. IoT-A was designed to be built upon to develop concrete architectures that are applicable across a range of domains.
- IEEE P2413 - Standard for an Architectural Framework for the Internet of Things (IoT): This ongoing IEEE standardization project aims to identify commonalities across IoT domains, including manufacturing, smart buildings, smart cities, intelligent transport systems, smart grid, and healthcare.
- Industrial Internet Reference Architecture (IIRA) – IIRA was explicitly developed for industrial IoT applications by the Industrial Internet Consortium, which was founded in March 2014 by AT&T, Cisco, General Electric, IBM, and Intel.

Figure below models a more detailed conceptual context diagram for IoT.

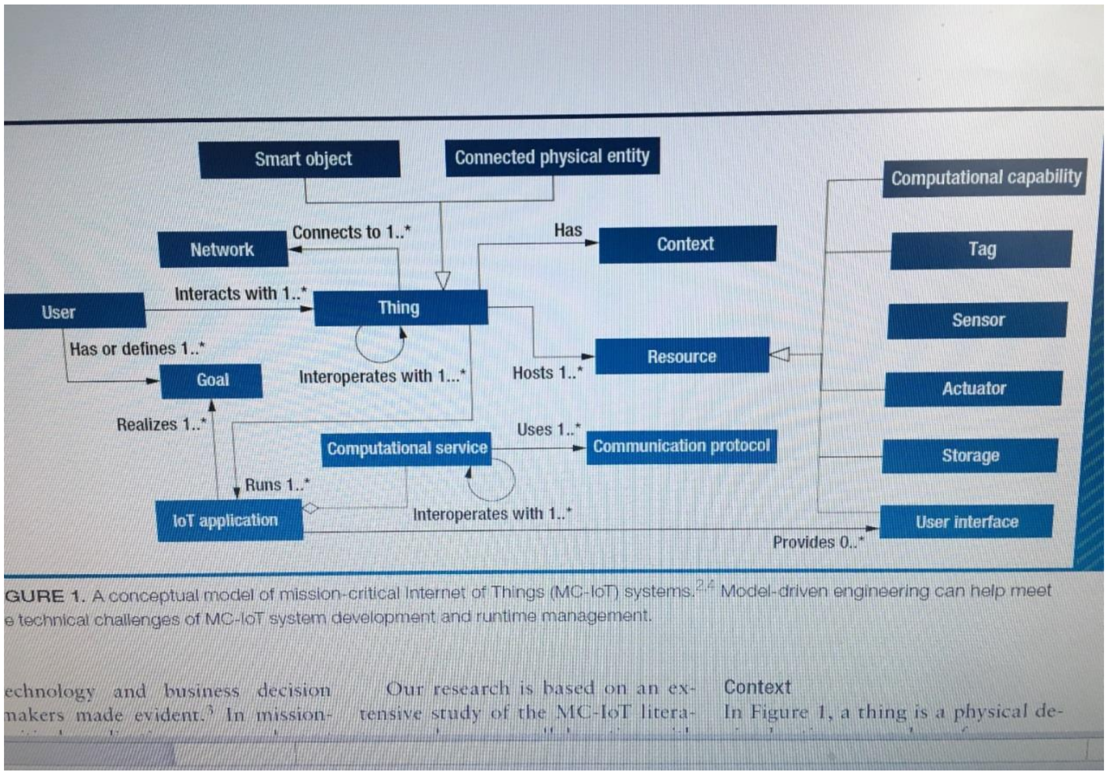

Smart object
Connected physical entity
Computational capability
Network
Connects to 1..*
Has
Context
Tag
User
Interacts with 1..*
Thing
Sensor
Has or defines 1..*
Resource
Goal
Interoperates with 1...*
Hosts 1..*
Actuator
Realizes 1..*
Uses 1..*
Computational service
Communication protocol
Storage
Runs 1..*
IoT application
Interoperates with 1..*
User interface
Provides 0..*
GURE 1. A conceptual model of mission-critical Internet of Things (MC-IoT) systems.2,4 Model-driven engineering can help meet
e technical challenges of MC-IoT system development and runtime management.
echnology and business decision makers made evident.3 In mission-
Our research is based on an ex- tensive study of the MC-IoT litera-
Context
In Figure 1, a thing is a physical de-

Figure: conceptual test model of critical IoT

An architecture should have the following capabilities:

- Managing devices and their data
- Connectivity and communication
- Analytics and applications
- TBD more from the figure

Each of these areas needs to be tested with functional and non-functional quality evaluations. A focus of many traditional test team is on functionality testing. This testing is a first verification checking step this is necessary but not sufficient. Reference architectures also describe mechanisms to address non-functional requirements such as flexibility, reliability, quality of service, interoperability, and integration.

Generally, reference architectures describe:

- Device on-boarding
- Updating device firmware
- Applying new configurations

- Triggering remote operations like disabling, enabling, or decommissioning devices
- Security and privacy
- Performance of system in time and resource use
- Reliability and trustworthiness
- Non-functional qualities necessary to the specific IoT device (project unique analysis)

Some current industry IoT platform-centric reference architectures (vendor specific) include:

- IBM IoT reference architecture
- Intel IoT platform Reference Architecture
- Microsoft Azure IoT Architecture
- Amazon Web Services Pragma Architecture

Ref to https://www.ibm.com/developerworks/library/iot-lp201-iot-architectures/index.html

Finally, environment, tooling, and support processes are major architectural elements of the above figure. However, just having IoT tools and processes alone that are used and followed, does not equate to project success. Details of tooling and support factors must be an on-going research effort and not is documented in the ebook currently. This ebook does explore the impacts to software test environments with consideration of test and the real world.

Classification of IoT Device Environments Needing Test

Figure 2 introduces an example IoT device test environment. The mind map is one of many possible environment taxonomies, but there is no universal standard for IoT test environments, as of now.

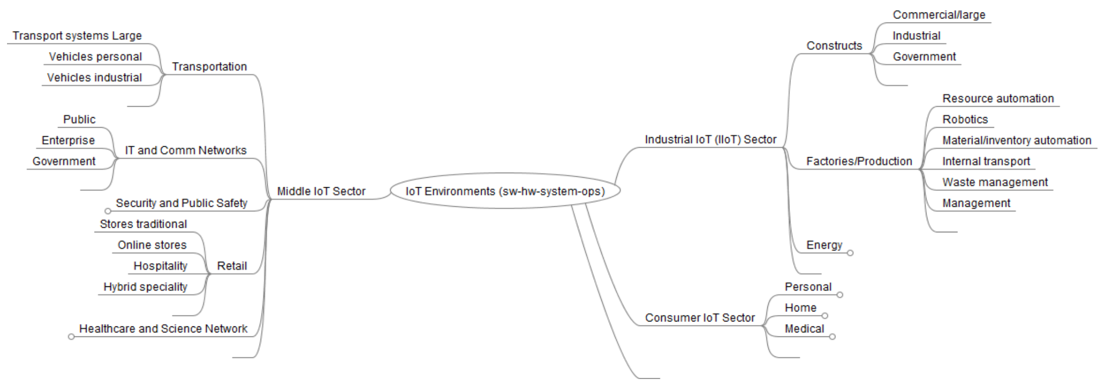


Figure 2 – IoT Device Environments Mind Map [4]
Start in the center with "IoT environments" to read this map, which includes the areas of software (SW), hardware (HW), system, and operations (ops). Testers of IoT devices should think of testing the device within a whole environment. During testing, each of these areas needs to be addressed by the test architecture, planning, environment, design, and implementation.

The map branches into three main divisions which are: industrial IoT (IIOT), Consumer, and what this paper calls the "Middle" sector. Most testers are familiar with IIoT, and Consumer IoT devices [1, 11] as these are in common literature usage today. The Middle IoT sectors are those systems that combine aspects of industrial and consumer. The middle environments find personal consumer, industry, and government users mixed to varying degrees and therefore, crossing the boundaries of consumer and industrial.

The Consumer IoT sector shows that personal devices can be worn, provide information to be aware of, and make life better. The consumer sector also includes devices found in people's homes to control their home, provide entertainment, and even help in managing things such as food inventory. The final parts of the consumer sector are systems that monitor the body, assure exercise, provide reminders to the individual, and possibly provide medications, e.g., smart pills. Many of these aspects have concerns about safety, security, money, and quality of the individual consumer's life.

IIoT systems have already come into use and are growing [1, 3, 11]. IIoT represents potential savings and information gathering, which can benefit an organization such as companies and governments. IIoT has sub-classification entities that include: constructs, factories, and energy. Constructs represent a significant resource that can be used in "large, industrial, or government" settings. Examples of constructs would include buildings, infrastructure, and even cities. Using IIoT, these resources can be built, managed and optimized for resources such as quality, people, cost or schedules. Factories and production facilities have detailed factors such as automation, robotics, inventory control, internal factor/facility transportation routing, waste management, and general management to name a few. Finally, a significant factor in the industrial world is energy using terms such as usage, manage, control and distribution.

The left side of the IoT map is the largest. The middle sector made up of transportation, comm networks, public security and safety, retail, and healthcare. Each of these environments overlaps industrial and consumer to some degree. There are many subsector categories and possibilities in these environments. Each node in the mind map represents an environment that testers may need to address and consider an IoT device. The middle IoT sector of the map has quality factors such as safety, security, financial, functionality usability, and reliability.

This book advocates that in IoT a comprehensive test environment and architecture must be fully considered by testing as well as the lower levels of the device, edge, and app-cloud. The team and tester must address the abundant test architecture elements outlined in the environment map. The testers must focus on the software system or system-of-systems, addressing the integrated cyber-physical system in the “real world” which has been reflected and in the test environment. IoT adopters may be tempted to take the minimal path of simplistic testing, ignoring the more significant architectural and environmental issues. However, this is likely to delay finding many problems until a device is in actual use. Finding problems in the field can substantially increase the cost of software [12].

Individual vendors may be able to “test” much of the right side to the mind map, but parts of the left side of the map will be challenging to address with testing done strictly by vendors. This situation raises the question of who owns the testing of the large-scale integrated IoT environment, e.g., the system-of-systems, a whole city, a factory, or a country?

The chapters and subsections of this eBook explain each classification in more detail as well as providing example devices and locations. This topology classification is not the only mapping possible nor is it complete (does not include all possible IoT/IIoT environments), but is how this eBook is organized in environments and architectures. Many readers and other authors will have other topologies. The different topologies are fine with me, and over time universal classification may become accepted.

Story:

Tbd - high altitude test company environment vs dev TDD emulator and script like LDRA with static support analysis, modelling, and sims – both are environments

Tbd - Picture shows the complexity and scale of IoT/IIoT.

Each of these IoT environments become a top-level IoT test environment. We detail these later in the ebook

Defining IoT Development (Dev) Support Architectures

The tools, processes, and support hardware are necessary for developers (dev) to do their job. In part, IoT developers need the same architecture support elements as any team of system, hardware, and software creators. In section TBD I will provide pointers to reference information, but given that this is a large effort, I will not go into this area in detail. However, dev staff must at some point must move into what I call in this book the test environment and architecture, often very early in the lifecycle.

My experience in the embedded world is that often the dev staff spend as much time in the “test labs” learning about the system, hardware and software the device as the testers do. This eBook has a focus on the test architecture and environment

IoT platform capabilities

The list below should be your starting point of requirements when considering a test platform.

The team (Management, dev, test, etc.) should consider these key IoT platform capabilities:

- Automation

- Modelling and simulation
- Device management
- Data communication protocols
- People factors (ease of use, familiarity, flexibility, etc)
- Data storage
- Rules and analytics (AI, ML, etc)
- Rapid application development and deployment (continuous integration (CI) and test (CT)
- Integration of the whole (system, hardware, software, Ops, etc.)
- Security and privacy
- The resource cost of developing and maintaining the solutions that you build with the IoT platform.

As a starting point, here are current popular general-purpose end-to-end IoT platforms that are suitable for a wide range of applications:

- **IBM Watson IoT**
- Built on top of IBM's Bluemix PaaS platform, IBM Watson IoT is a mature, developer-friendly IoT platform with strengths in real-time analytics and cognitive computing.
- **Amazon Web Services IoT**
- AWS IoT is a highly scalable IoT platform. It includes an SDK, authentication and authorization, device registry, a device gateway that communicates with devices using MQTT, WebSockets or HTTP, and a rules engine that integrates with existing AWS services
- **Microsoft Azure IoT Suite**
- The Microsoft Azure IoT Suite is another comprehensive IoT platform, which includes support for device management and device twins (like device shadowing) by using Azure IoT Hub, standard communication protocols including MQTT, AMQP, and HTTP, secure storage, rules, and analytics engines supporting predictive analysis and data visualization.
- **ThingWorx**
- PTC ThingWorx is an enterprise IoT platform that supports model-driven rapid application development. Features include device management, application modelling, support for standard

protocols including MQTT, AMQP, XMPP, CoAP, and WebSockets and predictive analytics, as well as REST APIs and SDKs to support integration.

- Kaa
- Kaa is a free open source IoT platform that is published under an Apache 2.0 license, which allows the platform to be self-hosted. Kaa includes REST APIs and SDKs for Java, C++, and C, with features for device management, data collection, configuration management, notifications, load balancing, and data analysis.

Ref https://www.ibm.com/developerworks/library/iot-lp101-why-use-iot-platform/index.html

IoT Test Architecture and Environment Viewpoints

Table "Example of IoT Test Architecture and Environment Viewpoints" presents an example of how a test environment can use architecture viewpoints to support IoT testing. The example derives from the mind map under the Middle IoT sector, transportation, vehicle personal, where a specific connected car is considered for testing.

Table Example of IoT Test Architecture and Environment Viewpoints [13, 14]

Environment Example	IoT ViewPoint	Sub Target	Examples
Connected Automobile	User (s) - Non Human	Hardware	Controllers, sensors, motors, batteries
		Software	Onboard App, cloud, third party, Operation system
		Comm	Vendor network, Wifi end to end comm, long duration trip/drops
		System	Safety, Security, availablity, reliability
	Human	First time driver	Security set up, limit usages, non techie
		Basic	Average user, disability, user help files
		Typical	Child, adult, techie
		Advanced	Race Car, expert on snow,
		Bad actor	Hacker, cracker, human using malware
	Dev	Structural tests	Coverage, static analysis
	Tester	Test process, planning, design, techniques, documentation	ISO 29119, ISO 26262
	Management	Information on	Cost, schedule, time to ship
	Ops	Failure management	Help desk, predicitive analysis tires
		Analytics	Machine learning, AI, privacy
	Stakeholder (owner)	Benfit	Information, self-drive
		Resoruce	Cost, schedule, savings

In this small example, a variety of viewpoints are considered each with target categories. The categories are further evolved into samples that a tester would directly cover in testing. Consider the Human is a first-time driver of the auto .i.e., a teenager who is not fully licenced. Many places have limits on who can be in the car, and the teen driver can do.

Additionally, parents may want a security lockout so the teen driver cannot drive at certain times of the day or night. Finally, the tester may want to address if the teen driver is tech-savvy or not because maybe the teen may be able to hack the system. These are only a few of the use cases from this viewpoint that should be addressed in test architecture of what should be a teen driving in the "real world."

Naturally, from this table, the size of the test environment, the viewpoints and architectural considerations of the test space become enormous, and then we run into the problem of test case explosion [13] for IoT. The use of viewpoints combined with test environment starts to organize this problem to make it visible to the development team and management. The next section expands this situation with a sampling of IoT risks.

Below is an introduction of individual IoT environments team should consider.

Environment: Developer level testing and analysis

Developers are working on hardware and software for IoT systems.

Software – assess low-level functions – see ISO 29119 part 4 structural testing

Hardware – electronic, mechanical shake/bake, acoustic, see other tools on charts

Tooling – Automate as much as possible. Analysis – models for both HW and SW. Peer review inspection

Story: tbd miss match "If" test problem that had to pass dev test but found by IVV

Environment: Software level test with Integration – tbd complete reuse 101

Top down vs bottom up vs mixed

Likely to move to CI with CT to CD – see charts

Build to Software-System level

Building and testing a "full" software system in stages

Tooling

Story: tbd doing a build with the wrong cm'd version of a piece of code that had "extra" unapproved changes that lost a team a week of work look for a real bug

Environment: Hardware level test with integration – tbd complete

Top-down

Bottom-up – most likely

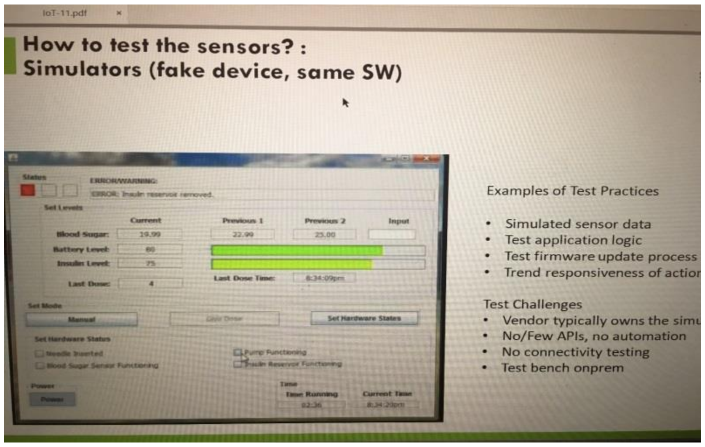
Mixed – prototypes and wire wraps

CI with CT to CD

Tooling

Story: test by installation guy with white glove looking for pic points every missed after retire

Story: tbd of lack of CM with transistor parts which had sub-part changes but did not update the full part revision level because a vendor thought the combination was "equivalent."


IoT-11.pdf

How to test the sensors? :
Simulators (fake device, same SW)

Status

ERROR/WARNING:
ERROR: Insulin reservoir removed.

Set Levels

	Current	Previous 1	Previous 2	Input
Blood Sugar:	19.99	22.99	25.00	
Battery Level:	60			
Insulin Level:	75			
Last Dose:	4	Last Dose Time:	8:34:09pm	

Set Mode
Manual | Give Dose | Set Hardware States

Set Hardware Status
☐ Needle Inserted
☐ Blood Sugar Sensor Functioning
☐ Pump Functioning
☐ Insulin Reservoir Functioning

Power
Power

Time
Time Running: 02:36
Current Time: 8:34:20pm

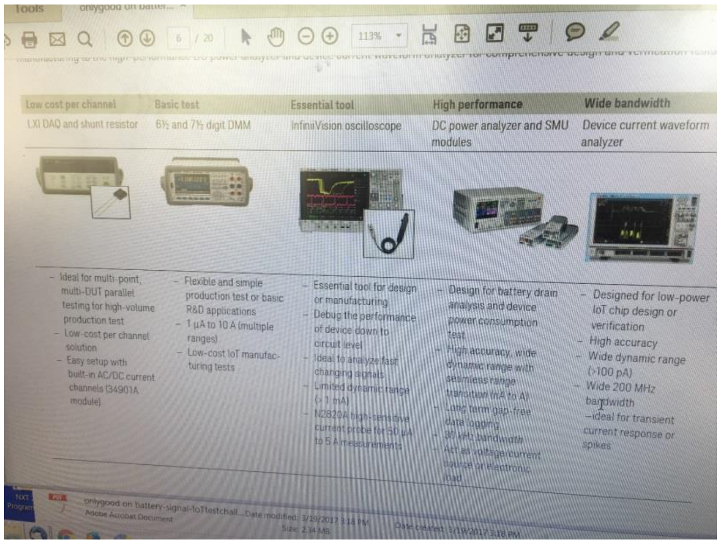
Examples of Test Practices

- Simulated sensor data
- Test application logic
- Test firmware update process
- Trend responsiveness of actio

Test Challenges

- Vendor typically owns the simu
- No/Few APIs, no automation
- No connectivity testing
- Test bench onprem

Figure TBD



Low cost per channel	Basic test	Essential tool	High performance	Wide bandwidth
LXI DAQ and shunt resistor	6½ and 7½ digit DMM	InfiniiVision oscilloscope	DC power analyzer and SMU modules	Device current waveform analyzer
– Ideal for multi-point, multi-DUT parallel testing for high-volume production test – Low-cost per channel solution – Easy setup with built-in AC/DC current channels (34901A module)	– Flexible and simple production test or basic R&D applications – 1 µA to 10 A (multiple ranges) – Low-cost IoT manufacturing tests	– Essential tool for design or manufacturing – Debug the performance of device down to circuit level – Ideal to analyze fast changing signals – Limited dynamic range (> 1 mA) – N2820A high-sensitive current probe for 50 µA to 5 A measurements	– Design for battery drain analysis and device power consumption test – High accuracy, wide dynamic range with seamless range transition (nA to A) – Long term gap-free data logging – 80 kHz bandwidth – Act as voltage/current source or electronic load	– Designed for low-power IoT chip design or verification – High accuracy – Wide dynamic range (>100 pA) – Wide 200 MHz bandwidth –ideal for transient current response or spikes

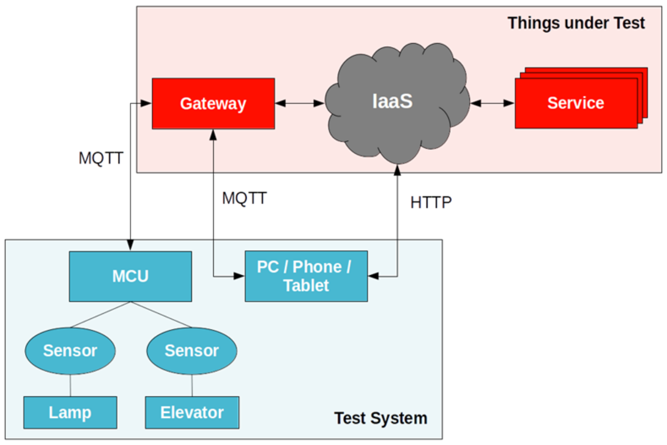
onlygood on battery-signal-IoTtestchall... Date modified: 3/19/2017 3:18 PM
Adobe Acrobat Document Size: 2.34 MB Date created: 3/19/2017 3:18 PM

Figure TBD: hardware test tools

Environment: Full Hardware-Software-System Integration – tbd complete

Do not wait – but plan iteratively

I address integration separate from test since my experience is as the complexity of a system grows, integration becomes a major task area requiring environments and support. Often this effort is not just plug and play.



You must plan Integration

You can test piece parts at a system level in “field” environments early which is good but may be hard

CI – early builds with prototypes for CT to CD

Stay agile

Tooling – labs with a much support system as you can afford

Story: Fly the Sill

Story: Third party HW-SW system where the system was not working at prime’s lab but the vendor lab everything worked fine. We had to fly the vendor team with the scopes to the prime lab to prove it was vendors problem

Environment: System Test– tbd complete

Can start with prototype system integration = get an early picture

Agile

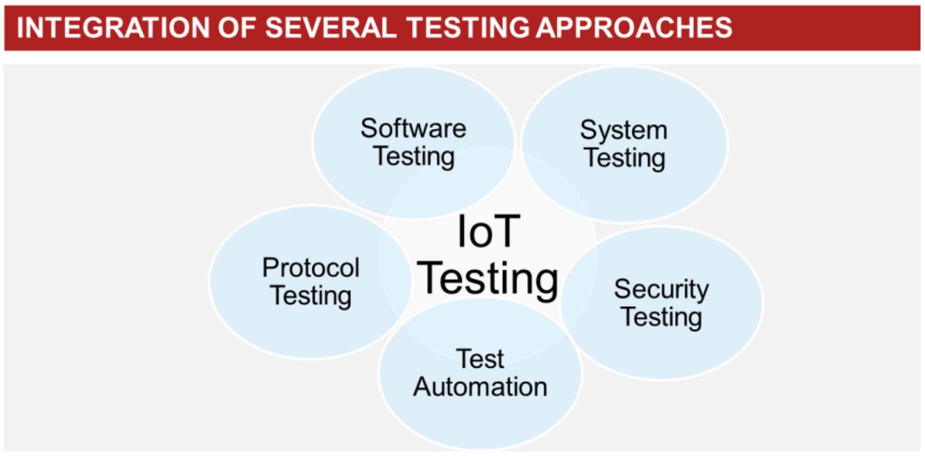
Move into real-world – test track?

Repeat

Tooling =?

Stories –Iron bird

Burning the TVC system up while testing unV&V software on actual hardware that cost 100000



Environment: Simulations and modelling with math **– tbd complete**

Modelling and simulations to solve the problem of limits of real-world – ref the car article too

Not MBT here but below (TBD section)

Data generation – CT or others?

This model is to provide a concept before the hw or sw exists

- Mind maps
- Storyboard
- Logic – UML, SysML, UTP

Simulation - analyzing before hw or sw exists. Good for interfaces and proof of concept

- Simple
- Complex

Test support sims here or after modelling section

Story – lab sims/models vs real world how to find issue via post-mission feedback

Environment: Special case of Sandboxes for security testing– tbd complete

A small step back

Testing a full system in a realistic system environment as possible, but separated from the real world so security testing can take place

A sandbox isolates the environment so “nasty” things can be tried and used

Tools – see charts

Story – virus released?

Environment: Field testing – tbd complete

TBD define field testing per ISO Ried

Make a list below into a Table

TBD Table

Test cities-ref

Testbeds cities

Skills?

Risks?

Story – traffic cam to improve flow see as a threat to privacy?

Auto and self-drive– ref and reuse HOW MUCH test WILL PROVE AUTOMATED CARS ARE SA

Happening now

Probably future book by someone

Regulations

Transition time will be exciting, but in some years in future life will be better

Story: Telsa driver killed

Medical

What is a medical device?

Safety

Security

Risk

Story – VP had update feature disable for fear of hacking

Factories

Robots and automation

Inventory

Speed

Performance

Hazard and risk?

Story: Joke about the dog and man – credit to who?

Cloud

See charts?

Testing many devices using cloud

Emulators to run massively parallel tests

IV&V, independent parties and crowdsource testing

Certification

Regulatory and legal

Who pays?

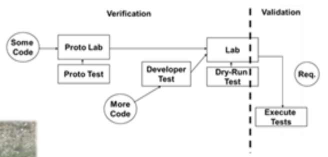
Story: million dollars 18 years find the issue

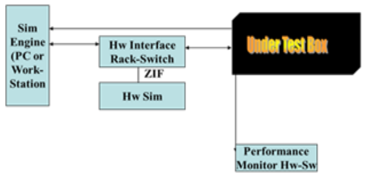
Example of Current Large Scale Test Architecture-Environment Solutions

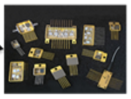
The industry has large-scale test architectures and environments that have proven test history including embedded systems testing, which is a good starting point for IoT since many embedded systems are evolving into IoT. Testers implementing IoT test architectures and environments should look to existing success stories to avoid “reinventing the wheel.” Additionally, these systems historically have scaled to larger implementations [10, 17]. Table 3 presents a list of example historical test architecture-environments.

Table 3 Examples of In-Use Complex Test Architecture-Environments

Example of Complex Test Architecture -Environment	Refer link	Notes
Embedded test labs of AreoSpace	17	Software, Hardware, and system integration facilities
Airbus Iron Bird	18	Airplane with all the parts, but cannot fly
Auto industry high altitude test lab	19	Testing the cars in the real world with a lab support facility
Test bed cities and open evaluation platform	7,15	Real environment, but how are done tests and recorded?
Chaos engineering	8	Testing on live systems.
Embedded test environments approaches	9	Lab with scopes, software,models, switch parts in and out
Device hardware qualification levels	16	e.g. space qual'd parts







IoT needs to build on the existing history of testing in areas such as embedded and cloud software. Table 3 shows examples for aerospace, automotive, governments, and the tech industry. For example, the Airbus Company has a facility they call the Iron Bird [18]. It is a configurable test lab with all the hardware, systems, and software of an airplane. The test architecture system is controlled by simulations and models so that testing can be done in a controlled environment. Reconfigurations to support verification and validation testing can be done quickly from prototype lab to real lab, to hardware-in-the-loop software test labs using architecture concepts shown in the table's figures. This test architecture-environment is very similar to ones seen in Aerospace and automotive industries.

The automotive and aerospace industries typically take test environments beyond the "iron bird" concept and more into testing in the field [19] as shown in the table's figure, which shows a car covered in drapes. This example is a car being tested on the back roads of Colorado for things like cold, high altitude, and bad driving conditions. We know that the system is under test because of the drapes and there is an instruments "chase" van behind the car.

These test approaches are made by independent labs [17, 19, and others]. The test architecture may go by names such as Independent Verification and Validation (IV&V), certification (cert.) labs, e.g., UL for space qualified parts, or IT organization test facilities. The key here is the architecture facilities are responsible for many devices, configurations, and address real-world situations. They are often independent of the vendors who provide the products. IoT systems will need similar approaches.

There will be questions of how such architecture-environments are funded and staffed. For Aerospace in the U.S., IV&V is procured by the customers independent of development [17, 23, 24]. Models, chaos engineering, and model-based testing are often crucial parts of the test architecture in IV&V, as is independent reporting [4, 10]. The scale of IoT may require similar concepts, though expansion and context driven customization of IV&V will be needed.

A Final Environment: IoT Analytics and Statistics – tbd complete

from O&M crosstalk? How data can be used

No tool/vendor specific but link to Forrester Wave™: Streaming Analytics, Q3 ’17

The OPS part of Dev-ops-test, but need a dev and test engineers to “watch” for the bugs so they can be detected, killed, and fixed per dev-ops concept

Story: many devs and most test engineers currently seem less than interested in analysis and AI. Big mistake. It will be a significant future

Test Environment (Labs) Creation –refer to my first book

Update details for IoT from book

Summary table

Project (s) inside of a project

Level of Environment vs V&V criticality – – tbd complete

Move? Parts of my paper here?

No one size fits all on tooling, automation, and environments

Will change over time

Testing the test environment– tbd complete

In my book “Test Attacks to Break Mobile and Embedded Devices” I detailed the need to test the test environment in Chapter tbd. The reasons and key points have not changed. These points are summarized in the table below

TBD Summary figure and table

Test Automation and Tooling: An IoT Necessity – tbd complete

Ref Fraunhofer charts

Why automate with models:

- Better use of resources (time and money)
- Avoid human mistakes in traditional testing over millions of tests
- Support Agile testing in the whole lifecycle

- Run more tests from model->keyword->data-driven with the agile turn around of new tests
- Faster regression testing

Evolve into Automation-Tooling to Survive IoT by Learning to Recognize When Automation Is the Best Choice

IoT testers may start with manual testing, but “just” manual forever may not be a good survival strategy. Testers need to be thinking about situations where the manual approach is not a good option for project test planning. Too often, many testers default to manual testing because it is what they are most familiar with, but given that there are some situations where automated execution must be included in planning, testers need not be trapped in a pattern of thought.

However, even with automation, teams still need thinking and skilled testers. Recognizing the need for automation is a skill worth building along with the factors to make automation success.

In the space industry project, there were several items critical to success. First, the test team needed skilled programmers and testers who understood the keywords and system under test. The project had dedicated people on these areas. Next, in test project planning, management had to allocate a schedule to develop the test system and train people to use the system. This took resources that had to be included in the project-level planning. Projects that do not allocate a schedule and resources for these efforts are more likely to fail during test automation. To assure these allocations, all levels of project management need to support the test execution automation efforts. Finally, the hardware and software efforts needed to be stable and integrated with the test development. This took ongoing communication and iterative planning updates within all project teams.

As software systems grow in complexity and size, test execution automation will become expected and, sometimes, even the only viable option. This situation will be seen in IoT, embedded, and large-scale, complex systems of systems.

While some historical manual testers may be unfamiliar with test execution automation, the growing trend into automation necessitates new skills for manual testers. IoT project test teams need to become aware of this trend, as automation represents not only business opportunities but also increased quality and fewer risks in complex, safety-critical, and mission-dependent projects.

Test teams working into automation of some tests should take the risk of try automation. My personal experience with automation and tooling is that out of every 10 attempts, 2 or 3 failed, 4 or 5 were just okay, and the final 2 or 3 were great successes. I highlight and build on the success, worked to improve the “just okay”, and abandoned the failures. Remember the old saying “Nothing ventured, Nothing gained.”

Automation: Test Tooling Areas – tbd complete

This section offers a review of emerging software test concepts and standards in which teams will find potential value in their improvement efforts including:

- Math-based techniques which apply combinatorial, statistical, Design of Experiments (DOE), or domain-based concepts

- Attack-based testing which focuses on common industry error taxonomies

- Independent model-based testing using tools and standards

- Add more tool categories for the test from where in the book and ?.

It is generally understood that testing is a critical phase and set of activities in system-software projects. Additionally, IoT projects should employ ongoing Verification and Validation (V&V) by Dev and test to help ensure the quality of a system throughout the life cycle. These concepts have been recognized for years as documented in a variety of industry standards such as IEEE 1012 Standard for System and Software Verification and Validation [1] and the new ISO 29119 Software Test Standard [2]. While some embedded projects use these test concepts and standards, opportunities for IoT systems improvement remain. Concepts for system-software test improvements addressed in this automation section include:

-Use of Math and Model Test Techniques to improve the system, hardware and software testing;

-Risk-based exploratory testing concepts;

-Attack-based testing to show the system DOES NOT meet the requirements;

- Execution

-tbd support.

It is common knowledge in the test industry that concepts and techniques beyond basic requirements verification checking exist. Now the less many IoT testers do even have a basic strategy [ref picture stat] The next sections of this eBook survey and provide references to useful automated test and V&V concepts, standards, and techniques that teams should have within their arsenal to support robust testing programs. While these are recommendations, there is no one "best" concept for IoT, so teams must be proactive in their improvement efforts by working test architecture, strategies and plans.

Math-based Concepts for System and Software Level Testing

An underutilized set of testing techniques for system and software testing are based in mathematical concepts. The table below provides examples of math-based test support techniques, which software testers should know about and be able to put into practice.

Table – Math-based Testing and V&V Techniques			
General Technique Concept	**Tool Examples** *(Note 1)*	**Examples of where a technique can be used**	**Specific sub-technique examples**

Combinatorial Testing	ACT [4], Hexawise[5]	Medical, Automotive, Aerospace, Information Tech, avionics, controls, User interfaces	Pairwise, orthogonal arrays, 3-way, and up to 6-way pairing are now available
	rdExpert [6]		
	PICT[7]		
Design of Experiments	DOE ProXL[8]	Hardware, systems, and software testing where there are "unknowns" needing to be evaluated	Taguchi [12]
(DOE)	DOE++ [9]		
Random Testing	Random number generator feature used from most systems or languages	Chip makers, manufacturing quality control in hardware selection	Testing with randomly generated numbers includes: fuzzing and use in model-based simulations
Statistical Sampling	SAS [10]	Most sciences, engineering experiments, hardware testing, and manufacturing	Numerous statistical methods are included with most statistical tools
Software Black box Domain Testing	Mostly used in manual test design, though some tools are now coming available [11]	All environments and types of software tests. These are “classic” test techniques, but still underused	Equivalence Class, Boundary Value Analysis, decision tables *(Note 2)*

Note 1: no tool listed here should be taken as a recommendation by the author, and it should be recognized there are numerous tool vendors. These are listed only as examples for the reader’s reference.

Note 2: while domain-based test techniques go back 40 years, most software testers do not fully understand the depth and full usage. See [11] for an excellent new resource to expand usage and understanding.

To support the justification by IoT projects of increasing usage, there is evidence coming from a variety of environments where math-based testing techniques have been used successfully including software, hardware, electronics, manufacturing, and systems [13] development. Areas supported include: interoperability, device configuration assessment to address different combinations, tbd

Software situations where math-based testing can be of assistance include significant amounts of data, a variety of combinations, where sampling heuristics are needed, and places where the “variables” of testing are unknown. The work by Kuhn, Kaner and others [4, 11, 12] are shedding insight and providing education on these situations. Almost any type of system with software can benefit from math-based testing. The reason these techniques help is that math supports numbers and computers process numbers. Additionally, project management and customers can facilitate the use of math concepts through support in regulations, contracts and other resources.

Beyond basic Automation: **Attack and Risk-based Software Test Planning with Exploration Concepts – add tooling for attacks**

The focus of scripted (written) test procedures to verify requirements is typically driven by showing compliance with contracts or regulatory considerations. Many software test teams focus on this form of verification exclusively because they are only given enough schedule and budget--often at the end of the life cycle, to get minimal efforts done. This can result in errors and information being missed [3]. Successful software test teams [2, 14] have learned that verification checking was necessary but not sufficient. These teams used approaches based on risk analysis with exploratory testing and attacks in their attempts to break the software. Example attacks as listed in the table below have been applied to critical mobile and embedded software-systems in the environments of avionics, medical, aerospace, automotive, and others.

To focus attack-based testing, test teams should consider risk-based testing as defined in ISO 29119. In risk-based testing, areas that are at higher risk are subject to earlier and more testing, which can include experience-based error guessing software test attacks. Attacks are patterns of testing that leverage techniques in attempts to break (find errors) in the software. Attack-based testing was popularized by Dr James Whittaker in his series of "how to break" books [14, 15, 16]. A more recent book by me has focused attacks on mobile and embedded software systems in their respective environments [3].

Table: Example Software Attacks for Exploratory Testing

Table: Example Software Attacks for Exploratory Testing

Software Test Attack Type	**Attack Finds**	**Notes on the Attack**
Developer level attacks	Code and data structure problems	Almost a quarter of errors in mobile and embedded can be found by structural testing
Control system attacks	Hardware and software control system errors	Many critical errors in mobile and embedded are centered in the control logic, for example, analogue-to-digital and digital-to-analogue computation problems
Hardware-software attacks	Hardware and software interface issues	The software should be tested to work with any unique hardware
Communication attacks	Digital communications problems	The software communicates with hardware, network, and other software with complex interfaces that should be tested
Time attacks	Time, performance, sequence, and scenario errors	System software can have critical timing and performance factors that testing can provide valuable information about

Smart/Mobile attacks	Issues specific to smart device configurations including cloud issues	Cloud-hybrid computing comprises a majority of the new software systems being deployed
Security test hacking attacks	Software errors that can expose devices to security threats	Security of devices or systems is increasing in importance, and attacks include, for example, GPS and identity spoofing
Generic functional verification attacks	Requirements and interoperability errors	Basic checks that testers should conduct on systems and software
Static code analysis attacks	Hard to find errors that classic testing often misses	Can often be done by the development group but sometimes the test group must run this analysis

Notes: Details on this table and these test attacks are contained in [3]

Risk-based testing with attacks has been used by teams to conduct rapid exploratory testing to optimize error finding in conjunction with requirements verification [3]. The balancing of these test and V&V efforts takes critical thinking [17] and, there is no "one size fits all" set of attacks or explorations. When the risks justify it, a highly skilled tester is needed to conduct the attacks and testing. Lack of highly skilled testers may be one reason that teams opt for purely scripted verification checking and as a result miss critical errors [18,19]. The mix and match of concepts and techniques driven by risk-based testing may allow better optimization than the "just do minimum checking" that many groups engage in.

Model-based Testing with automation for IoT **– tbd complete**

System, software, and test, model engineering have been growing in use and interest [20]. Specific industry segments have active efforts in model-based engineering and testing, such as telecom, finance, automotive, and aerospace. Additionally, many European countries are actively participating in updates to modelling standards, and their support is ongoing, e.g., the Unified Modeling Language (UML) test profile (UTP). While interest and use are growing, model-based testing is still immature in general usage. Thus, for groups seeking to improve project activities, another test area to consider, is the use of model information to support testing.

Model-based testing can support:

1. Generation of test cases from models into test automated execution engines directly using scripts or through the use of keywords;

2. Improved understanding of the system and risks;

3. Use of models to support simulations to drive test environments;

4. Verification via compares between development and test models;

5. Generation of test result oracles or judges;

6. Support for independent testing such as Independent V&V (IV&V); and

7. Models analysis.

These features can offer many improvements in the testing process and V&V life cycle as shown in the figure below.

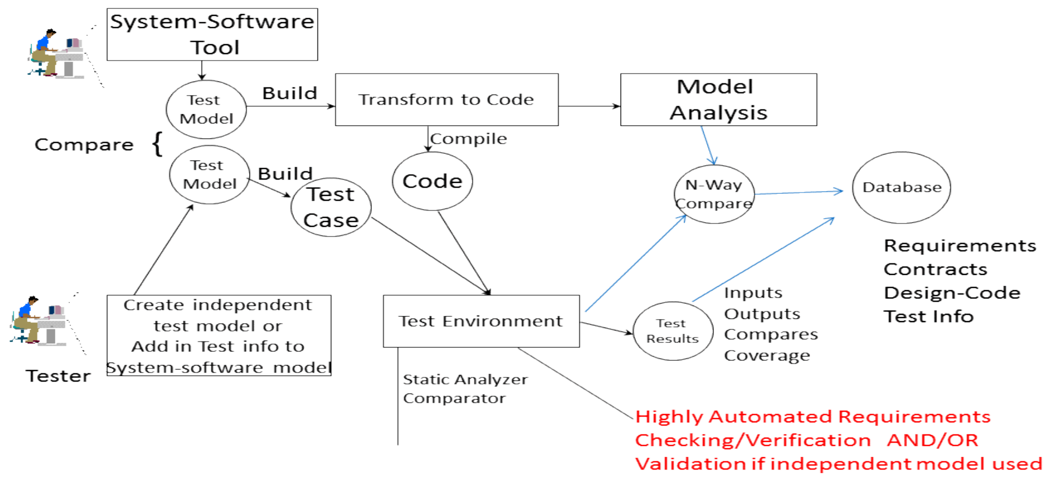


Figure - An Example Test Flow with Modeling

Many of these advantages have been used on a customized basis in the past. Others are being developed with supporting tools and processes [21]. Further, independent test and development models of a system can assist in validation and verification by comparison of the models and analysis to find errors. The rigour to produce the model aids in the removal of errors, and then the models can be used to generate test information including data, execution scripts, and in some cases test oracles.

The expense and efforts involved in model-based testing may not be justified in all contexts. The concept and expense of having both a development model and test model have potential issues in the form of a variation of the N-version programming problem [22]. However, the use of test models has shown to be able to find errors and serve as an oracle [23]. For organizations dealing with critical software, complicated software where risks are high, or software that may have long-term regression

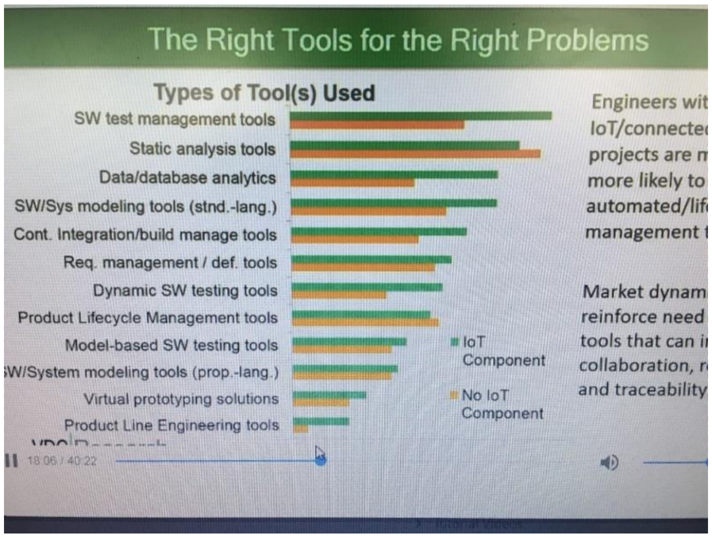
problems, model-based testing may offer viable improvements. It has been applied with specialized support tools in automotive, finance, and aerospace [23].

To be successful, model-based testing must be conducted in the right environments (computer-aided development or labs with test automation tools), by skilled staff (people who know modelling languages and testing), and using model-based test tools. These support resources are becoming more widely available. Model-based testing will likely grow as tester skills in modelling improves [21].

Test execution automation – pointer to category concept– tbd complete

Test Support areas – tools categories– tbd complete

Automation: IoT Dev Tooling Examples– tbd complete



Engineers wit
IoT/connecte
projects are m
more likely to
automated/lif
management t

Market dynam
reinforce need
tools that can i
collaboration, r
and traceability

TBD intro to the table:

Table of High-Level Tooling Concepts– tbd complete

Developer level testing tooling
Test Lab tooling – software
Test Lab – Hardware tooling
Hardware tools, probes, connectors, brass board, etc
CM

Lifecycle support
Management-planning-control
Reporting
Integration lab tooling
Modelling tools
Security lab – Sandbox tooling
Sniffers
Fuzz
A long list from Hagar web site
Spoofing
Cloud lab tooling
Crowd lab tooling
Real world tooling

Automation: IoT Dev Tooling Examples– tbd complete

See website

Tools will be one way to go

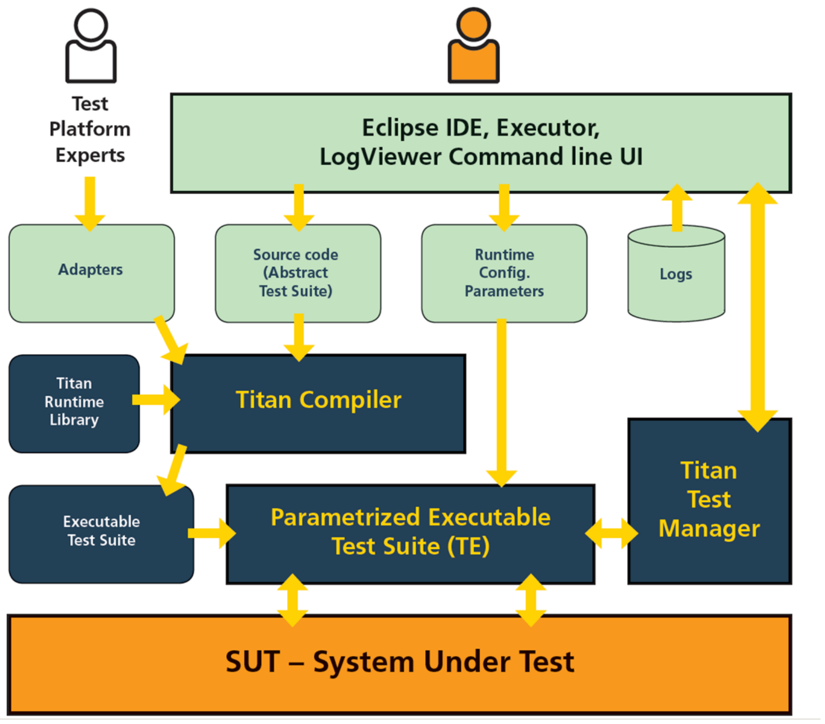
Pointer to the web site

- Best Programming Languages for IoT
  - Assembler
  - C
  - B#

    - JavaScript
    - Python
- Top IoT Development Tools
    - Node-RED
    - Arduino
    - Eclipse
- Best IoT Platforms
    - DeviceHub
    - ThingSpeak
    - mangoOH

ref: https://qubit-labs.com/top-iot-programming-languages-tools/

- **open source** test tools (Eclipse)


Test Platform Experts
Eclipse IDE, Executor, LogViewer Command line UI
Adapters
Source code (Abstract Test Suite)
Runtime Config. Parameters
Logs
Titan Runtime Library
Titan Compiler
Titan Test Manager
Executable Test Suite
Parametrized Executable Test Suite (TE)
SUT – System Under Test

Automation: Lessons learn

There are many lessons to learn in test automation of IoT systems. Probably the first and most important is that automation is only part of the story for IoT testing as you still need thinking testers who can do rapid manual testing and use tools when needed. Automation does not always save money but can help to yield IoT product that delights customers. Many managers believe they can get rid of test staff and just let the computer do the work, thus saving money, but computers, even ones with AI, need thinking creative humans to drive the test of hardware, software, and the system.

I will not try to list all the lesson learned here. I will give you, the reader, some pointers to one of my favourite source of lessons, that being Dot Gramm. Some of her wisdom is contained in the figures below. She also has to great reference books, which I wholeheartedly recommend you get and read at some point. These are TBD ref and websites.

tbd

Sample Risks Caused by Insufficient Test Architectures and Environments

One of the goals of this research was to start examining the scope of the IoT test problem, which is large and has many risks. Table 2 lists some of the top level risks for IoT.

Table 2 IoT Risk Sampling [1, 4, 7, 11, 15, 16]

Risk Area Example		**Risk Area Example**
The reactive and always-on nature of the devices		Lack of realized benefits promised by IoT vendors
Heterogeneity and diversity at the same time across many systems and devices		Waste (cost or schedule) caused by failures seen in the field
Power/battery usage limitations		Errors and failures impact happiness and quality of life
The massively distributed, highly dynamic, and migratory nature of devices		Disruption in society caused by devices (story of traffic lights)
The need for software fault-tolerant and recovery		Lack of resources (cost and schedule) for test environments
Fragmentation of the market place (many vendors)		Lack of responsibility for quality across the system or system of systems
Configuration management of devices to maintain consistency and qualities		Interoperability and integration across devices and sectors
Current approaches in testing and test architectures do not scale given billions of devices		Software quality characteristics not met, e.g. High availability, relability, safety, security, usability, and functionality
Lack of universal product Comm standards		Hardware quality characteristics not met

All of these taken together may impact the success of IoT adoption, if not tested during IoT efforts. Many implementers and adopters of IoT have not considered the scope of the risk and test space problem. IoT is in the early adoption phases. However, vendors, companies, government, and individual users are already using IoT, without addressing many of these risks. The scale and risk problems outlined here are just a small sample and beginning.

Future Research Needs

Research opportunities exist in IoT test architecture and the associated environments. These include:

- Specialized IoT IV&V facilities at the full system/system-of-system for IoT [4]
- Data analytics with real-time testing in the field and self-healing systems
- Government departments tasked with the focus on IoT
- Privacy and security regulations and standards [15]
- Model-Based Testing (MBT) and simulation-driven testbeds [4, 25, 26, 27]
- Predictive Maintenance [28]
- Distribution and heterogeneous systems on the Internet [3, 7]
- Industry and government test labs with independence (cost and schedule) [29, 30]
- Testing to address the fragmentation of IoT [31]
- IoT Test support tools
- Support process standards, e.g., ISO and IEEE [13, 14, 23, 32]

This list is not complete and will evolve as IoT changes. The list will notably change as the IoT challenges, threats, and opportunities immerge. Unfortunately, many of these will emerge during failures, product recalls, and other "disasters" that may doom some IoT projects.

Summary

The research for this paper has indicated the need for more work in IoT test architecture and environments. An IoT classification example for test architecture and environments was outlined including demonstrating viewpoint usage in testing. Further existing and successful test facilities from the embedded software device world were presented to show where IoT testing may need to go. Some projects will make IoT devices just "good enough" to continue in the market, but likely only a few will truly be "good enough" at these system-level IoT architectures and environments. The lack of being "good enough" will lead to many IoT projects learning by failures in the field. The risks outlined in this paper are only part of the IoT problem. IoT, IIoT and IoT of everything are being rushed, implemented and fielded before many of these test architecture and environment issues are even considered, let alone solved, and resources will be wasted.

Test Architecture basis, Strategy, and Standards: Implications, Conclusions and Recommendations

Risk-based, math-based, experience-based (attack and exploration), requirements-based verification and model-based testing are all viable bases for starting the test and V&V process and activities. Groups that use several of these bases in combination may be able to increase the test information that they provide thus, providing more value to a project or company.

Combinations of bases and standards should be considered in a strategy and set of plans because complete testing and V&V of a complex software system are not possible [2]. Each basis samples or provides aspects of the system. IEEE 1012 and ISO 29119 offer guidance on how to approach testing and V&V using different integrity levels of the software under test and employ a variety of techniques within a set of test/V&V processes. A combination of standards and bases is implied since each of these are engineering heuristics and thus subject to missing needed test information, such as embedded errors. However, and while implied, hard supportive use data is hard to find or does not exist currently.

The standards presented here are new or contain new information and so do not have the use history data to indicate optimal strategies and plans. Project and groups using the standards should consider presenting to industry what is a viable, optimal usage, and what needs to evolve in the standards. Changes to standards will take time and groups such as IEEE and ISO have ongoing maintenance plans for the standards.

Risk-based, math-based, experience-based, requirements-based verification and model-based testing do have industry use history, but are often used individually and not in combination. Thus combination strategies and V&V plans cannot be defined optimally based on industry experience and statistically significant data. Further, the error finding the ability of combinations has minimal reporting [3] in error taxonomies or industry error data, thus highlighting the need for more research and reporting.

The test/V&V bases and standards were presented in this paper to increase awareness of them since some are new and others historically underused on projects. As the use and application of combinations increases, it will be of value to industry researchers and standards groups if the actual project data and usage experiences become publically reported. Those of us working in standards and research need more concrete, large industry project data, and not simple graduate student projects. However, many projects do not take the time to report data or are not permitted to release such information. This hampers the evolution of standards and understanding of how to create optimal test/V&V strategies and plans.

Appendix A: Tools pointer to websites - tbd

Mine -

Techwell or tbd site??

Hardware

Appendix B: How to grow your tester skills

How to Stay Happy in Your Testing Career

Jon Hagar, GST LLC
jon.d.hagar@gmail.com

Testers tend to focus on skills, learning, providing information, and getting the job done on time. They want careers, reasonable compensation, and the ability to find work as needed. Of course, these are important considerations, but I find that they miss an area that surveys only occasionally touch on, which is job satisfaction.

I extend the satisfaction theme even further. I believe a tester should have fun, find their jobs challenging, and look forward to an exciting career. I say this because when I talk about my life's path in the test, people ask how I did it.

The main point in life is to have fun. This does not mean every workplace task must make you happy; there will be stress and maybe even some pain, but the big picture of how you work should result in happiness. If it does not, consider changing something.

How do you change your future test path?

The high-tech world of computers and software means those of us working in it must always be learning, building skills through practice, and looking to the future. The tools, ideas, and systems of forty years ago (when I started) were vastly different from today, and the systems of tomorrow will be even more different. Are you at risk of being caught in a skills-practice gap?

To keep my career and skills on the track, I have always had a career plan covering what I want to do this year and what I'd like to be doing in the next few years. I work on learning and building skills by going to conferences, reading, and taking risks to practice new ideas.

Specifically, I learned languages designed to support testing. Next, I learned Artificial Intelligence (AI) where I used neural networks to do data analytics on error reports. This led me to interest in taxonomies and understanding patterns to improve software testing. In turn, I wrote a STAR conference award-winning paper on this subject, which then led me to publish books in testing embedded, mobile and IoT systems. This was my plan and path. Yours will be different, but I have always found fun challenges in software testing. You can too.

Here are things you get or work on to improve your IoT testing:

Become more artistic

Learn more about engineering concepts and areas, e.g. civil engineering, medical (all branches and subfields), electronics, mechanical, psychology, biology, etc.

Establish a sizeable personal reference library of books and materials (I have over 500 books and large reference sources)

Work building skills in new areas that you are not familiar with

Become an entrepreneur

Work at being more of a risk taker

Have a career plan, near-term and life long

Work on test skill areas (see Appendix A)

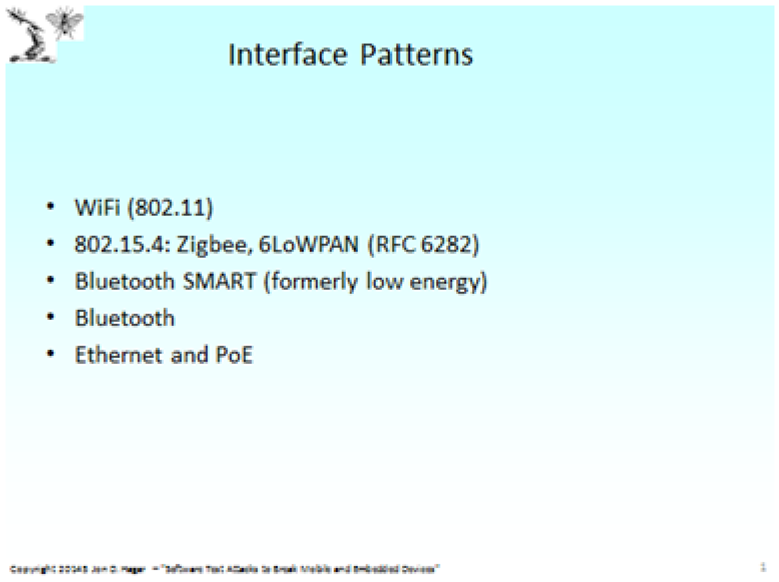
Have some fun!

Appendix C: Automation Situations the Newbie Tester or Company May Face

- Should I keep/do this appendix?

Appendix D: IoT 2016 Example Supporting Standards

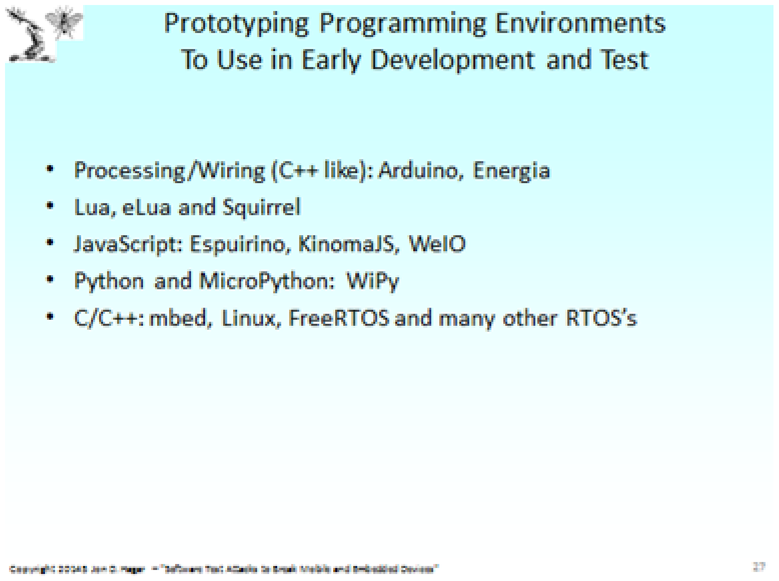
Currently power point. Need to turn into tables and add words. Chart titles need rework into table names….. These will evolve rapidly.



Interface Patterns

- WiFi (802.11)
- 802.15.4: Zigbee, 6LoWPAN (RFC 6282)
- Bluetooth SMART (formerly low energy)
- Bluetooth
- Ethernet and PoE

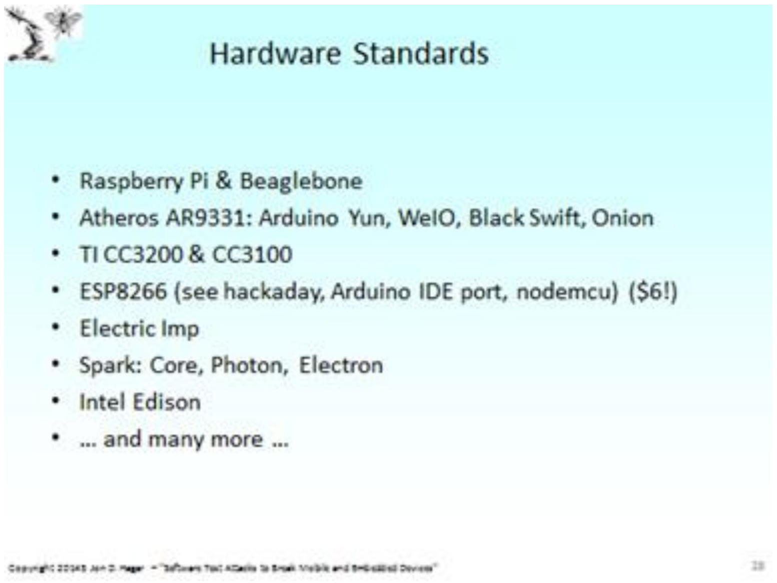
1



Prototyping Programming Environments To Use in Early Development and Test

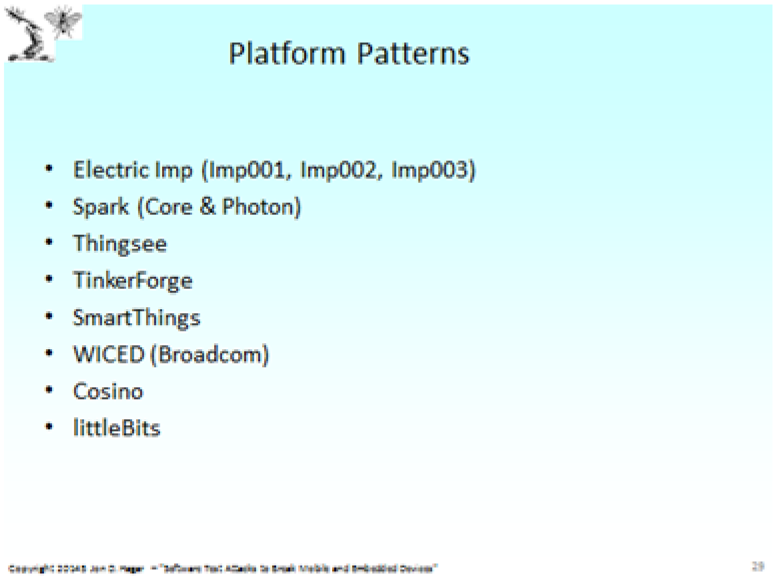
- Processing/Wiring (C++ like): Arduino, Energia
- Lua, eLua and Squirrel
- JavaScript: Espuirino, KinomaJS, WeIO
- Python and MicroPython: WiPy
- C/C++: mbed, Linux, FreeRTOS and many other RTOS's

27



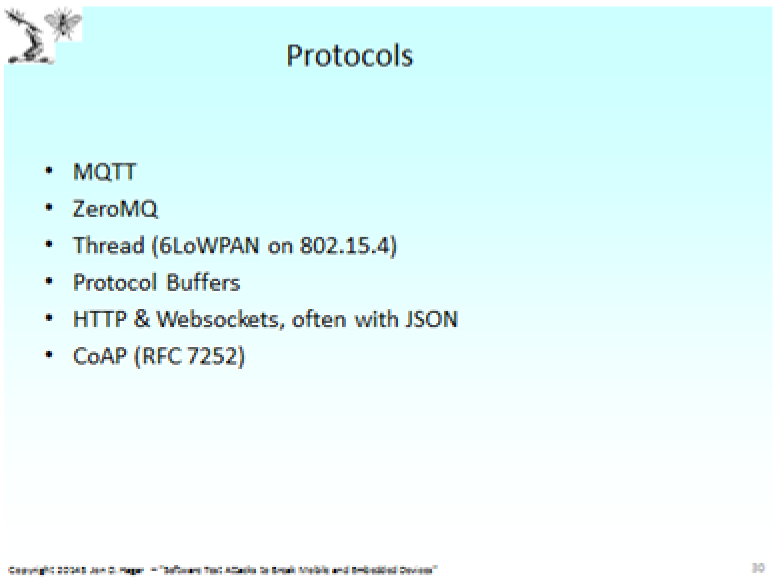
Hardware Standards

- Raspberry Pi & Beaglebone
- Atheros AR9331: Arduino Yun, WeIO, Black Swift, Onion
- TI CC3200 & CC3100
- ESP8266 (see hackaday, Arduino IDE port, nodemcu) ($6!)
- Electric Imp
- Spark: Core, Photon, Electron
- Intel Edison
- … and many more …



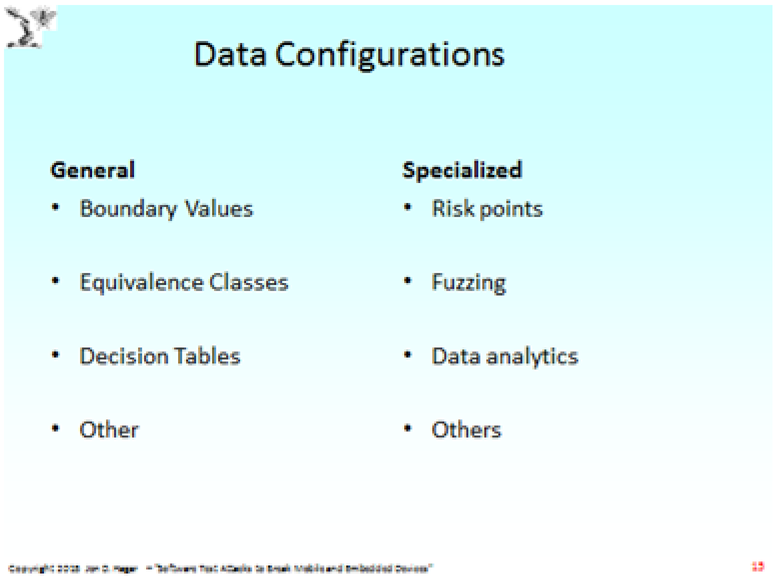
Platform Patterns

- Electric Imp (Imp001, Imp002, Imp003)
- Spark (Core & Photon)
- Thingsee
- TinkerForge
- SmartThings
- WICED (Broadcom)
- Cosino
- littleBits



Protocols

- MQTT
- ZeroMQ
- Thread (6LoWPAN on 802.15.4)
- Protocol Buffers
- HTTP & Websockets, often with JSON
- CoAP (RFC 7252)



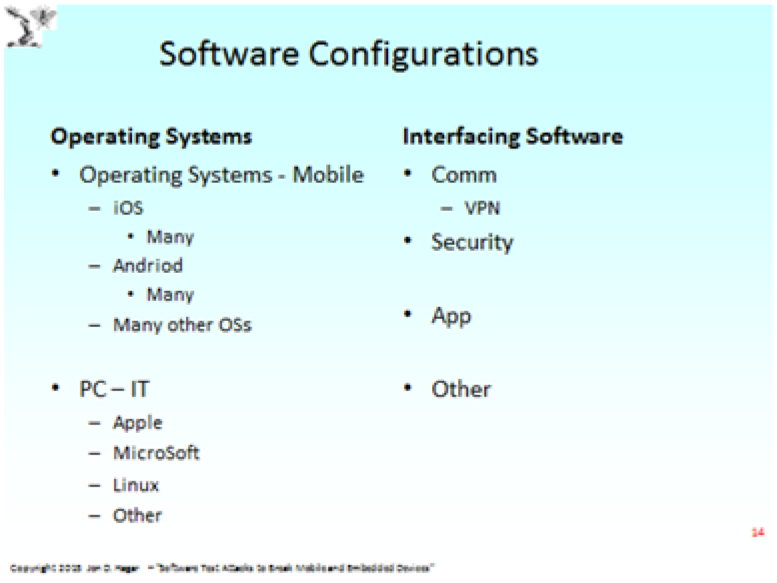
Data Configurations

General

- Boundary Values
- Equivalence Classes
- Decision Tables
- Other

Specialized

- Risk points
- Fuzzing
- Data analytics
- Others



Software Configurations

Operating Systems

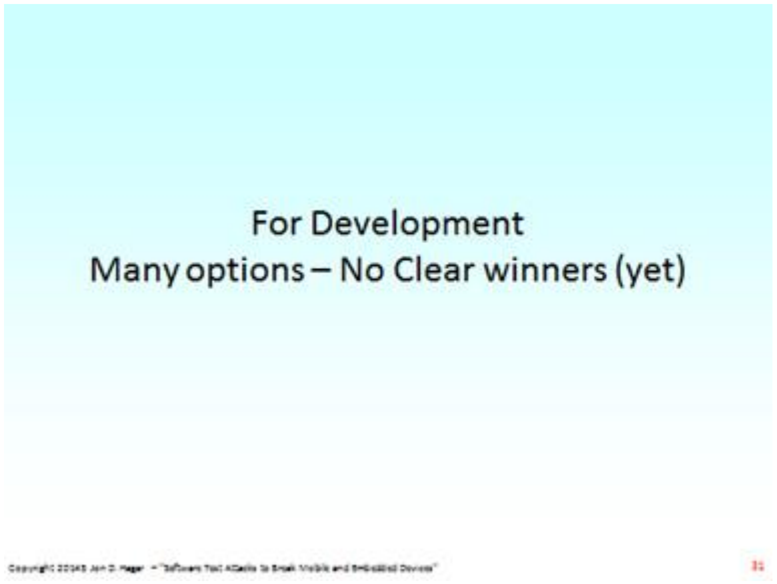
- Operating Systems - Mobile
  - iOS
    - Many
  - Andriod
    - Many
  - Many other OSs
- PC – IT
  - Apple
  - MicroSoft
  - Linux
  - Other

Interfacing Software

- Comm
  - VPN
- Security
- App
- Other

14

Copyright 2013 Jon D. Hagar – "Software Test Attacks to Break Mobile and Embedded Devices"



For Development
Many options – No Clear winners (yet)

Copyright 2013 Jon D. Hagar – "Software Test Attacks to Break Mobile and Embedded Devices"

31

Should I have a process standards section: IEEE and ISO

References for Additional Learn

Glossary of Definitions

In this book, I have followed—as much as possible, common usage and definitions from the following sources.

- SEvoc — http://pascal.computer.org/sev_display/index.action
- IEEE — ISO/IEC/IEEE 24765:2010 Systems and software engineering (Vocabulary)
- Definitions in the *How to Break* book series by James Whittaker.

If you do not find a term in this list, refer to one of the sources listed here or one of the references given throughout the book, or you can do an Internet search for the term. (Google can be your best friend in helping you to find things.)

Definitions/Abbreviations

DOE	**Design of Experiments**
FPGA	**Field Programmable Gate Array**
IEEE	**Institute of Electrical and Electronics Engineers**
ISO	**International Standards Organization**
IV&V	**Independent Verification and Validation**
V&V	**Verification and Validation**
UML	**Unified Modeling Language**
UTP	**UML testing profile**

References – tbd fix and finalize

•Handbook of Visual Display Technology, Springer ("all-in-one"), § 11 "Display Metrology" by K. Blankenbach →

•DFF "Automotive Display Measurement Specification" (add-on to "OEM Display Spec"), www.displayforum.de

•SID (SOCIETY FOR INFORMATION DISPLAY) International Committee for Display Metrology (ICDM): INFORMATION DISPLAY MEASUREMENTS STANDARD → (IDMS), free download: www.icdm-sid.org

•**Software**: - RADIANT IMAGING: Color Calculator - PIXPERAN: Response Time, Motion Blur (visual assessment) - Test patterns, test software etc. by BUROSCH, Display Lab,…

ThingML (Internet of Things Modeling Language) = find a link

Check https://blog.testproject.io/2017/10/11/open-source-test-automation-frameworks

[1] – Schools of Software Testing: A Debate with Rex Black, Cem Kaner, J.D., Ph.D. July 15th, 2016, http://kaner.com/?p=437

[2] – The Art of Software Testing – Myers, 1979, John Wiley and Sons

[3] – Wikipedia web site, Landing Page, https://www.wikipedia.org/

[4] - Software and Systems Engineering Vocabulary, https://pascal.computer.org/sev_display/index.action

[5] - IEEE Standards Dictionary Online, http://dictionary.ieee.org

[6] - ISO/IEC 29119, Wikipedia web site, https://en.wikipedia.org/wiki/ISO/IEC_29119

[7] - State Board of Licensure for Architects, Professional Engineers and Professional Land Surveyors, DORA, State of Colorado, https://www.colorado.gov/pacific/dora/AES

[8] - Manifesto for Agile Software Development, web site, http://agilemanifesto.org/

[9] The mythical software engineer, Michael Dunn -December 07, 2016, EDN, http://www.edn.com/electronics-blogs/benchtalk/4443132/The-mythical-software-engineer

[10] - Stop 29119 web site, http://www.ipetitions.com/petition/stop29119

[11] - DO-178B, Wikpedia web site, https://en.wikipedia.org/wiki/DO-178B

[12] - General Principles of Software Validation; Final Guidance for Industry and FDA Staff, FDA, http://www.fda.gov/MedicalDevices/DeviceRegulationandGuidance/ GuidanceDocuments/ucm085281.htm

[13] - Software Test Architecture Design focusing on Test Viewpoints, SOFTEC 2012 2012/7/12, Nishi, Yasuharu, The University of Electro-Communications, Japan, http://qualab.jp/materials/SOFTEC2012-2.pdf

[14] - IEEE Certificates Program, IEEE web site, https://www.ieee.org/education_careers/education/certificates/index.html?utm_source=mm_link&utm_campaign=certificates&utm_medium=ec&utm_term=certificates%20program

[15] -Certifying Software Testers Worldwide Landing Page, ISTQB web site, http://www.istqb.org/

IEEE paper arch 2018

References –

1-"Presentation on System of Systems Engineering Challenges inIoT based Systems", François Coallier, IEEE 12th International Conference on ?Systems of Systems Engineering (SoSE), 2017-06-21

2 – The Internet of Things in Action: Testing Anki's OVERDRIVE Racing Game, Jane Fraser, May 2016, STAReast Conference Presentation

3-NIST's Network-of-Things Model Builds Foundation to Help Define the Internet of Things, July 28, 2016, https://www.nist.gov/news-events/news/2016/07/nists-network-things-model-builds-foundation-help-define-internet-things

4 – eBook: IoT Development and Testing, Jon Hagar, 2017, https://leanpub.com/iotdevelopmentandtestingpart1

5 – Defining the Phrase "Software Test Architecture", Jon Hagar, 4th International Workshop on

Software Test Architecture, 2017, IEEE
6 - https://en.wikipedia.org/wiki/Unintended_consequences
7 - A Roadmap to the Programmable World Software Challenges in the IoT Era, Antero Taivalsaari, Tommi Mikkonen IEEE SOFTWARE, Jan/Feb 2017, P72-79
8 - https://en.oxforddictionaries.com/definition/environment
9 - Concepts for Automating Systems Integration, Edward J. Barkmeyer ea (2003), NIST publication 2003
10 – Software Test Attacks to Break Mobile and Embedded Devices, Jon Hagar, CRC press, 2013
11– IIoT http://internetofthingsagenda.techtarget.com/definition/Industrial-Internet-of-Things-IIoT
12 Software Engineering Economics, - Boehm, B. W., Prentice-Hall, Englewood Cliffs, NJ, 1981.
13 – ISO 29119 Software Testing Standard, 2013
14 – ISO 26262 Functional Safety Standard, 2011
15 - Privacy and Data Security: Minimizing Reputational and Legal Risks, Tatiana Melnik, Melnik Legal, PLCC, proceedings Better Software Conference west, 2015 – quot regs exist and are expanding
16 -Enabling IoT Ecosystems through Platform Interoperability, Arne Bröring, Stefan Schmid, Corina-Kim Schindhelm, Abdelmajid Khelil, Sebastian Käbisch, Denis Kramer, Danh Le Phuoc, Jelena Mitic and Darko Anicic, Ernest Teniente, IEEE Software, Jan/Feb 2017, P54-59
17 – Industrial Experiences in Establishing Laboratories and Software Models to Effectively Execute Software Test, Jon Hagar, Quality Week Conference, 1999
18 - http://www.aircraft.airbus.com/innovation/proven-concepts/in-design/iron-bird/
19 - http://www.sgs.com/en/news/2016/05/sgs-expands-testing-services-with-new-rocky-mountain-test-center
20 - City of things: An integrated and multi-technology testbed for IoT smart city experiments Published in: Smart Cities Conference (ISC2), 2016 IEEE International by Steven Latre, Philip Leroux, Tanguy Coenen
21-– http://principlesofchaos.org/
22 - Challenges in the Qualification of Electronic Components and Systems, Vidyu Challa, Peter Rundle, Michael Pecht, IEEE Transactions on Device and Materials Reliability, Volume: 13, Issue: 1, March 2013
23 – IEEE 1012 IEEE Standard for System, Software, and Hardware Verification and Validation
24 - Cyber-physical systems, IoT, and smart cities "global city teams Challenge, Sokwoo Rhee (sokwoo.rhee@nist.gov), Embedded system conference 2015
25 - UML TESTING profile 2 OMG's new standard for model-based testing, Marc-Florian Wendland, SDL-Forum, Berlin, 12th October, 2015
26 - Model-Driven Engineering for Mission-Critical IoT Systems, Federico Ciccozzi, Ivica Crnkovic, Chalmers , Davide Di Ruscio, Ivano Malavolta, Patrizio Pelliccione, Romina Spalazzese, IEEE SOFTWARE, Jan/Feb 2017, P46-52
27 - Model-Based Software Engineering to Tame the IoT Jungle, Brice Morin, Nicolas Harrand, and Franck Fleurey, IEEE Software, Jan/Feb 2017, P30-36
28 - https://www.sap.com/products/predictive-maintenance.html]
29 https://www.computerworld.com.my/resource/applications/malaysian-software-testing-board-opens-us33-million-q-lab-in-malaysia/]
30 - "Design and Deployment of an IoT Application-Oriented Testbed", Laura Belli, Simone Cirani, Luca Davoli, Andrea Gorrieri, Mirko Mancin, Marco Picone, and Gianluigi Ferrari, University of Parma IEEE

COMPUTER 0018- 9 162 / 15/ 2015 , p 32-41
31 - https://nextm2m.com/fragmented-iot-ecosystems-are-destined-for-connectivity-interoperability-and-security-problems/
32- ISO 12207 - ISO JTC 1/SC 7/WG 7 ISO/IEC/IEEE 12207-2: Systems and software engineering — Lifecycle management — Part 6: System integration engineering 2016-12-20(WD), ISO JTC 1/SC 7/WG 7

References

1. IEEE 1012, Standard for System and Software Verification and Validation- http://standards.ieee.org/findstds/standard/1012-2012.html, IEEE press, 2012

2. ISO 29119, Software Test Standard - http://www.softwaretestingstandard.org/

3. Hagar, J. Software Test Attacks to Break Mobile and Embedded Devices, CRC press, 2013

4. Kuhn, Kacker, Lei, Introduction to Combinatorial Testing, CRC press, 2013 (includes the tool ACTS)

5. Tool: Hexawise - app.hexawise.com/

6. Tool: rdExpert – www.phadkeassociates.com/

7. Tool: PICT – msdn.microsoft.com/en-us/library/cc150619.aspx

8. Reagan, Kiemele, Tool: DOE Pro XL - Design for Six Sigma, Air Academy Associates, self publish, 2000

9. DOE++ - www.reliasoft.com/

10. SAS - www.sas.com/

11. Kaner, Hoffman, Padmanabhan, The Domain Testing Workbook, self publish, 2013

12. Bailey, Design of Comparative Experiments. Cambridge University Press, 2008

13. Kacker, Kuhn, Hagar, Wissink, "Introducing Combinatorial Testing to a Large System-Software Organization," scheduled-2014, IEEE Software

14. Whittaker, James 2003, How to Break Software, Pearson Addison Wesley

15. Whittaker, James and Thompson, Herbert, How to Break Software Security, Pearson Addison Wesley, 2004

16. Andrews, Whittaker, How to Break Web Software, Pearson Addison Wesley, 2006

17. Levy, Tools of Critical Thinking: Metathoughts for Psychology, 1996

18. Bach, Bolton, “Testing vs. Checking,” www.developsense.com/blog/2009/08/testing-vs-checking/

19. Hagar, “Why didn’t testing find the embedded GM Truck fire system error?”- www.breakingembeddedsoftware.wordpress.com/

20. OMG UTP 1.2, www.omg.org/spec/UTP/1.2/

21. Baker, Dai, Grabowski, Schieferdecker, Williams, “Model-Driven Testing:Using the UML Testing Profile,” 2008

22. Green, Hagar, “Testing Critical Software: Practical Experiences,” IFAC Conference 1995

23. Boden, Hagar, “How to Build a 20-Year Successful Independent Verification and Validation (IV&V) Program for the Next Millennium,” Quality Week Conference 1999

24. Port, Nakao, Katahira, Motes, Challenges of COTS IV & V, Springer press, 2005

Contact Information

Jon D. Hagar, Grand Software Testing

Jon.d.hagar@gmail.com

Acknowledgments