Internet of Things (IoT) Development and Testing

Part 1: Fast Start Guide

There and Backplane Again

Scott Allman

Jon Duncan Hagar

2017

Publish - LeanPub

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

Tweet: jonduncanhagar

Linkedin name: jonduncanhagar

Email: jon.d.hagar@gmail

Release Notes
This version is 100% complete for now.

Preface

We 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, we believe teams should not separate development (dev) and testing efforts from each other. Our 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.

We have structured this series of eBooks to address these considerations yet be cost-effective. So rather that one big expensive book, we 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, we plan ongoing updates, so please contact us with ideas, improvements, and what you like. We 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, we hope to keep costs and time minimized. We wish to help teams learn about IoT as this part of the industry grows.

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

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

- What is IoT verification and validation

- How do testers find errors (bugs) in IoT software

- Organizational impacts caused by IoT

- Lifecycle impacts of IoT

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 larger scale, higher risk IoT systems.

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

Finally, we expect that readers have some knowledge of standard or typical development lifecycles including software testing. No book can address all aspects of technology. We 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. We have such libraries, and this is one reason why some people regard us as experts. We do not know everything, but we *do know* when and how to look up materials. A definition of expert that we 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

We would like to thank and recognize our many mentors, colleagues, and teachers, who have taught us so much through the years. There are many ideas in this book from nearly all of them, which we try to cite throughout the book. We are all standing on each other's shoulders. Our most significant acknowledgment goes to our reviewers and Jon's wife, Laura— software–systems geek and head reviewer. Without their assistance, this book would not be possible. With all of these people, the references in this book and hopefully, the book itself, we hope that the IoT device world becomes safer, more secure and fun to use.

We particularly would like to thank the following people who over the years have helped to make us learn critical thinking:

Cem Kaner
James Bach
Lisa Crispin
Becky Fielder
Laura M. Hagar

About the Authors:

Jon Hagar is a senior tester with over 35 years of 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 supported include the domains of embedded, mobile devices, IoT, and PC/IT systems as well as test lab and tool development.

Jon is and has been a member of numerous professional organizations, including Association of Software Test (AST), IEEE, ACM, ISO and Object Modeling Group (OMG). He serves as an author and current project editor of ISO 29119 Software Test Standards as well as a member of IEEE 1012 Verification and Validation Planning standard working group, 1028 review committee, and co-chair on OMG Unified Modeling Language testing profile standard, as well as a voting member on many other standards. He is past member of professional society boards: Denver-QA and American Software Test Qualification Board (ASTQB).

Jon has taught hundreds of classes and tutorials in software engineering, systems engineering, and testing throughout the industry as well as at universities. He has published numerous articles on software reliability, testing, test tools, formal methods, mobile, and embedded systems. Jon is the author of the book: *Software Test Attacks to Break Mobile and Embedded Devices*, as well as a contributor to books on agile testing and test automation. Jon makes presentations regularly in industry working groups and conferences. He also holds a patent on web test technologies.

Scott Allman is a senior software developer with a long history of teaching software testing. His career began in academia where he taught programming languages to Computer Science undergraduates and supported statistical software on mainframe research computers. After about a decade of doing that he spent the next three decades working for many startups, testing software for multi-national projects in Hong Kong and Brazil, and as a consultant he recently designed and developed software for several Internet of Things (IoT) systems and Big Data Analytics projects.

Scott is a frequent and popular presenter at software testing conferences and has spoken on a diverse set of topics: Test Automation Tools, Ethical Issues for Software Testers, Reason and Argument Skills for Software Testers, Adapting Popular Open Source Software Testing Tools, and, of course, Testing and IoT. He is a natural teacher and has taught all courses in the Association for Software Testing's BBST series. He is a member of the Association for Software Testing (AST), Software Quality Assurance of Denver, and many local software organizations in Colorado, USA.

Late in life, he earned an M.A. degree in Philosophy from the University of Colorado where he wrote his thesis on an epistemic issue in software testing, "The Philosophical Limits of Knowledge Claims Based on Formal Methods in Computer Science." He holds a B.A. degree from Indiana University also in Philosophy. He often quips when asked about his philosophy degrees, "When I was learning computer programming as a college student in the 1960's a philosophy degree, especially symbolic logic, was the only computer-science-like degree a guy could get!".

Currently, he is an independent software consultant working on IoT and Big Data projects and lives in Boulder, Colorado, USA.

Introduction – What is the Internet of Things (IoT), V&V, and Testing?

The world of technology continually moves from one hot area to another. We had computers, then personal computers, then the web/.com systems, and more recently mobile/smart devices. Each represents an expansion of software and system functionality along with the growing pains of errors and many company failures. However, for every Microsoft or Google, there are tens or hundreds of companies that did not succeed in each of these market "explosions." This eBook considers IoT from a testing viewpoint:

- How do companies get into the IoT game?

- How do companies improve their IoT game?

- How do people (dev and test) get into the IoT game?

- How, in this new arena, can you leverage your hard-learned lessons from general software testing?

To date, many of these explosions have involved companies that were already technology-based or start-up organizations where the people involved had a high tech background. Additionally, there were influxes of people that had little or no knowledge of technology but showed some ability to learn quickly. These people were often "new hires" who were given the task of sink or swim within these companies. Like the companies, some people succeeded, and some did not.

At the beginning of the technology maturity curve (see link below), the next hot topic we see includes the IoT (Internet of Things). First, let's understand the technology.

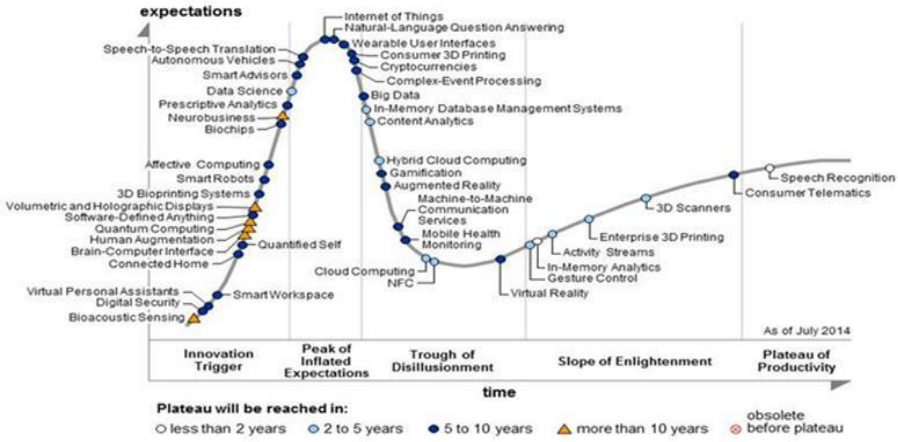

Figure: Reference: Tech Growth Curve - https://media.licdn.com/mpr/mpr/p/7/005/081/13a/317cb86.jpg

What This Book Addresses

This first eBook in the series provides a starting point for information on IoT assessment including:

1. Basic IoT concepts and history
2. Introduction to development ideas
3. Basic Verification & Validation (V&V)/test approaches and concepts
4. Help in startup on IoT as well as optimal skill sets for testers
5. Summary of IoT life cycles
6. The importance of the user interface and data analytics in IoT

Testers should use this eBook with other books, possibly software testing standards and other eBooks in this series. Our current favorites include:

Software Test Attacks to Break Mobile and Embedded Devices (Jon D. Hagar)

A Practitioner's Guide to Software Test Design (Lee Copeland)

How to Break Software Security (James Whittaker), the "How to Break" series

Domain Testing Workbook (Cem Kaner)

Lessons Learned in Software Testing (Kaner, Bach, Pettichord)

How to Break Software A Practical Guide to Testing (James Whittaker)

Audience

Dev-ops-testing is a large subject. Most readers of this eBook will spend periods of time building tester knowledge and skill. With over 70 years' experience between us, we are still learning new testing skills. Further, readers should keep in mind that there is no best or one path to do development (dev) and testing. There are many options for testing; each has positives, negatives, as well as cost and schedule impacts. Software development and testing are technical skills based on knowledge which is practiced over a lifetime.

This eBook is written for organizations and people that are new to testing and IoT. The industry value of the IoT world is expected to be trillions of US dollars in the next years and tens of billions of devices. Almost every traditional industrial company will enter into IoT, and there will be hundreds of new start-ups. The following kinds of organizations and people may find useful information in this eBook:

1. Companies, who have never developed software, but want to expand into IoT within their existing product lines. These might include industrial/medical companies (e.g., medical devices, health monitoring systems, heating control systems, transportation systems, etc.) and consumer companies (e.g., wearables, clothing, home entertainment, etc.). While it is hard to imagine any company that offers technology products could be a novice at software development and testing, the software in IoT devices requires special considerations.

2. Companies who have experience in electronics and limited aspects of software but are looking to expand their software footprint while lacking networked software testing background (e.g., television, audiovisual devices, automotive, etc.).

3. Startup companies looking for a good beginning reference into IoT development and testing.

4. Testers looking to learn more about IoT software testing to enhance their careers.

5. Groups and government officials looking to define IoT conduct, standards, and regulations.

6. Anyone that is interested in IoT testing but lacks test understanding.

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 re-read in detail. We do not intend the book be read in detail from cover to cover, but by jumping around to topics of interest in IoT.

The table of contents can also be used to index into topics that are of interest to readers. We suggest the reader assess if they understand basic concepts of testing found in Chapters 1 and 2. If a reader determines they understand the basics, then use the index or table of contents to find specific topics of interest, e.g., IoT attacks, patterns, security, or environments. We would suggest as testing of an IoT device and system progresses that testers refer back to the book. We also request if a reader finds something missing or lacking that you contact us, as we plan revisions to the eBook rapidly following agile concepts.

A brief history of technology in time and space

The history of humanity is, in part, one of developing technology (see figure below). We started with physical systems that, in the early days, were tools made of stone. Humans evolved these physical systems into more complicated technologies, for example, buildings made of stone and steel in cities and medical devices designed to monitor or heal us. During the most recent technology development, humans have created cyber systems (computers) starting in the 1950s. Cyber computers are thinking machines which in part are programmed in software to help humans with complex activities. However, like the Turing Test (https://en.wikipedia.org/wiki/Turing_test) shows, machine thinking is arguably different than human thinking. Machines imitate or simulate aspects of human thinking, such as math, data analysis, logical constructs, and other activities that may make the computer seem "smart" yet, at this point, machine thinking is different from human thought and awareness. Currently, programmed machines do precisely what the program says, and if the logic is wrong or missing, the unwanted behavior occurs. Such problems are, in part, why with software programs we must do testing--because humans program machines and humans make mistakes that computers blindly follow.

Figure: Evolution of Technology Systems from the Dawn of Man until Today
Reference: Jon Hagar IoT Class Presentation 2016

Today, we see the increased merging of physical and cyber systems in part to create the IoT. IoT becomes possible because of maturing cyber and physical systems including:

- Embedded computer-software devices
- Advanced physical systems
- Cyber IT computer systems
- Integrated communication technologies and networks (the web) – allows for tiny components to easily communicate
- Smart-mobile cyber-physical devices

The IoT is, in part, an extension of an existing product area called embedded software devices that have been around almost since the beginning of computers and software. Technology companies have been using small computers, often called microprocessors or another integrated circuit (IC) chips, for over 40 years. These devices typically had small amounts of storage, computing power, and software. They controlled systems but often did not have a standard computer user interface--a keyboard and screen. The embedded software was often used in the development of "advanced physical systems" where features and functions were provided by the combination of hardware and software. Many times, users of the embedded advanced devices were not aware they were working with software. There were a few books written about how to develop and test them (see reference section). For example, IT security researchers who found the first embedded software virus-worm, when they found the worm, they did not even know the kind of computer-software the worm was targeting.

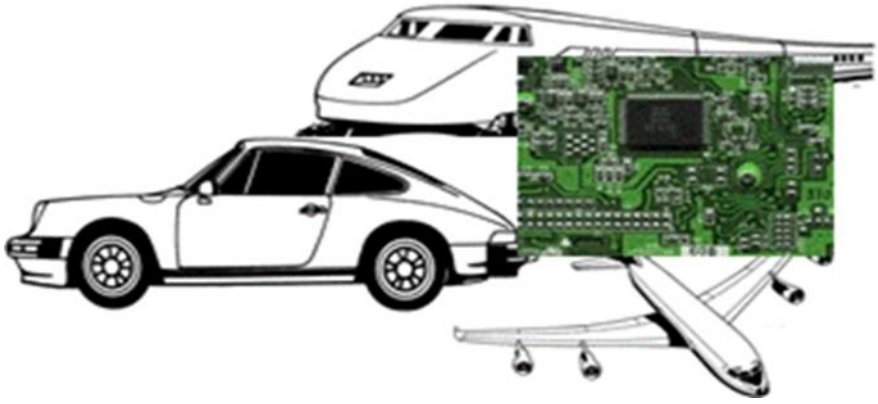

Figure: Example Embedded Software Systems and advanced physical systems moving to IoT
Reference: Jon Hagar IoT Class Presentation 2016

While the embedded and advanced physical devices grew and matured, another segment of the cyber industry developed the first mainframes (1950-1960), and then in the 1970s, the so-called

"personal computer" (PC) appeared. At first, the mainframes, Information Technology (IT), and PCs were the domain of a select few users (Nerds). The user processed information, dealt with errors (bugs) and formed groups (e.g., IEEE).

Figure: 1970 and 1980s IT systems and PCs
Reference: http://www.technected.com/wp-content/uploads/2013/10/computer-racks.jpg

During the 1980s, the PC explosion happened. In this explosion, first workplaces then homes got PCs and then many other computers. More and more people started using computers to do things like writing letters, doing emails, and creating spreadsheets. IT became part of every business and in many homes. We started experiencing more computer software bugs, and testing/QA departments were established in many tech companies. During the 1980s, the first books on software testing appeared and individuals made careers out of software testing. Th books published in the late 1970s (*The Art of Software Testing* by Myers) and in the 1980s (Kaner, Falk, Nguyen published *Testing Computer Software*), books which are still used today.

Later in the 1980s and into the 1990s, the next cyber advancement exploded, this was the rise of computer networks, the internet, and the world wide web. These advancements became the ".com" explosion of the 1990s--and later a bubble that popped. Many companies and people entered into working on cyber systems. Many more users, both at work and home, became "computer fluent." We did business on the web; now we do even more business on the web giving rise to e-commerce. Companies like Amazon and Google became powerhouses, and everyday users went to these sites in cyberspace either at home or from their new "smart" phones. This growth continues. However, many cyber systems had minimal connection to the physical world. Embedded systems continued to do the work of making physical systems "smart" and advanced, yet most physical systems were not connected.

Figure: Mid 1990 to 2000s Web and Communication
Reference: http://www.dsoftsolutions.net/wp-content/uploads/2012/09/computer_networking.jpg

Over the last decade or so, mobile-smart devices using cellular technology now dominate the cyber market. The number of mobile-smart devices now outnumbers classic personal computers. The devices are small; hand carried, battery powered, have more complex user interfaces, and have connectivity to computer networks, e.g., cell systems and the internet. The devices are close "kin" to physical systems since they have sensors and many are now "connected" to other physical systems, e.g., cars, medical devices, etc. This blending continues into the world of IoT.

Figure: 2000s and 2010s Mobile – Smart Cell System
Reference: Jon Hagar IoT Class Presentation 2016

With IoT, we see a merger of the traditional physical and cyber systems. IoT devices have become viable now because IT, communication, data, hardware, and mobile technology have matured. Some chips have processors, memory, sensor suites, and communication nodes. The cost is becoming very low (compared to historical computers). These IoT devices are small and interfaced to what many would consider classic embedded hardware as well as being fully connected to networks. This blending offers new product features and capabilities. The number of IoT devices has driven the need for IPv6 (https://en.wikipedia.org/wiki/IPv6) since the current internet addresses have "run out" (IPv4) and the number of IoT devices connected to the internet is expected to be in the billions or more very soon.

Computer inputs and output evolved in ways that affected software testing. Early inputs were notoriously clumsy and prone to error. No one wants a return to paper tape and punched Hollerith cards. A giant leap came with terminals and their familiar typewriter layout. However, more importantly, terminals are windows to shell scripts and the start of automated testing. For outputs,

computer users got immediate feedback on the screen. With IoT, these trends continue – as long as you are doing black-box testing. You tickle the IoT device just like a consumer, or another IoT device does and observes the behavior. However, for more sophisticated testing, the absence of a keyboard and mouse for outputs, and a large monitor for outputs require new types of hardware/software configuration skills.

IoT market segments

We see three IoT segments: consumer, middle (mixed), and industrial (see Table below). Industrial IoT devices control and communicate about our cities, offices, factories, transportation, utilities, and pretty much everything that makes modern life possible. Consumer devices will be things at home and used by a person. The middle is a mix of consumer and industrial uses. Likely the number of industrial IoT devices and systems will outnumber the consumer market by several orders of magnitude.

Table: IoT market segments

Industrial 4.0	Mixed	Consumer
Government	Vehicles/Robots	Home
- Health	- Driverless	- Security
- Safety	- Monitors	- Control and monitor
- Data analytics	- Infotainment	- Infotainment
Transport/Robots	Office	Human
- Navigation	- Security	- Health
- Optimization	- HVAC	- Fitness
- Logistics	- Worker monitor	- Data analytics
Work Place/Factories/Robots	Retail	Travel
- Ops and control	- Ordering	- Fun
- Data analytics	- Point of sale	- Location
- Automation	- Advertise	Robots

Consumer devices are more visible and have already gained public interest. These devices include automobiles, home monitoring-control, personal medical and wearables, as well as life enhancement systems. Pretty much anything that has in the past been used by consumers that had electronics will move to IoT. Additionally, many new products will be created which were never thought of as "electronic" but become IoT components, e.g., smart shoes, net clothing, smart food packaging, and others yet to be dreamed of items.

There will be many IoT systems that cross the bridge between consumer and industrial to form the middle because they have communications to make the bridge easy. Industry and consumers will be interested and users of these IoT devices. The division between the three is easily blurred. Consumer products generate data and have uses that interest industry. Likewise, the industrial IoT systems will generate information and have uses that consumers want. Companies working in the IoT segments may specialize, but companies will be players in each segment.

Figure: IoT Examples
Reference: Jon Hagar IoT Class Presentation 2016

From a testing perspective, there are differences in testing based on segment (as a starting point). For example, a wearable device like a watch may be "low" risk for bugs being an issue, but the more advanced versions of the watch that might also monitor health data would be a "higher" risk device, thus requiring more and different testing. IoT impact testing may be hard to gauge because of this. Consider the watch, which at first is deemed "low risk" but that start getting used for health data. Should it be retested? Our experience shows us that many companies may say something like "well, the device has been in used in the field, therefore it is proven," but this statement has proven false for other embedded device systems that were thought to be proven in a one use environment, but then found to have major faults when used in another case.

A Sampling of IoT challenges in development and testing
Key Point:

IoT devices face new and old testing challenges

To contemporary software people, few things seem as clunky and impractical as the old IBM punch cards. To computer programmers, punch cards were common and indispensable for working on a computer. To the general public in the 1960s, the cards were nothing short of magical. I can recall people who knew that I was a programmer bringing me stray cards they found lying around campus. These offerings were treated with great reverence when handed to me even when the cards were blank. Besides being just a piece of light cardboard in a standard size, the cards allowed a programmer to intervene on the sleeping computer and make it dance. Of course, you had to wait for your turn in line to use the mainframe computer. Most significantly, it was a straightforward approach to instructing the computer with your wishes. Perhaps the computer operator might drop your cards, but otherwise, when you handed in your deck you could expect a printed response on large sheets of paper called green bar (because it had green bars across it since it was about 22-inches wide).

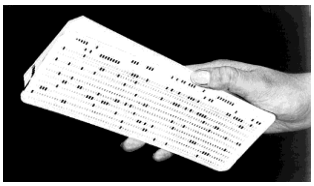

Figure: Old Computers
reference: https://www.concise-courses.com/security/wp- content/uploads/2015/07/mainframe-computers.png and http://www.computerhistory.org/atchm/wp-content/uploads/2015/01/IBM_Punch_Card.png

Output was the second part of the lifecycle of your computer job. What could be simpler than a large sheet of green bar paper with your results? When things did not go as expected all the information you needed to fix the problem was right at your fingertips. Do some debugging and try again? All your observations were in one place and ready for annotation with a device that has been around since the sixteenth century - a pencil.

Finally, the last step was the results of your run. Data read from a printout populated reports and articles. During the 1960s, it was common in newspaper and magazine articles to read phrases like, "the CPU of the computer is the 'brains' which control all the data processing" or "on large reels of magnetic tape enormous data is read and written." The public did not understand what goes on with computers. Today, decision-makers are far more technically savvy. However, they may be as ignorant of IoT as were the decision makers of fifty years ago were of computers. How can they be expected to understand the risks when they do not understand the technology? As a designer of tests, as a reporter of test results, and as an advocate of observations from intervening in the IoT system, you cannot neglect the challenge of explaining what is going on in IoT.

Flash forward to today and IoT development and test. For typical users, IoT will seem like magic for some time to come. Development teams will have integrated development environments (IDEs). We will pull in software reuse items rapidly. Our projects will be primarily reused elements and micro-services that we leverage with language constructs and libraries. We will rely on third parties to help with functionality.

IoT functionality will span hardware, software, communications, and operations while generating large amounts of data, which we must also use. Hardware problems already abound including:

- Many product standards
- Multitudes of device-hardware configurations that change constantly
- Sensor input challenges
- Control output issues
- Battery and power concerns
- Memory usage restrictions
- Processor speeds limitations
- Hardware lifecycle
- Just to name a few

Software factors to address include:

- Security and privacy
- Data processing
- Ubiquitous usability
- Third party impacts
- Hardware limitations fixed in software
    - (An old saying is: "We will fix that hardware problem in software")
- Short life cycles and interfaces to hardware life cycle

Communications challenges include

- Speed of the networks (local, nearby, and global)
- Drop and brownouts
- Multiple communication standards
- Device-to-device traffic
- Interoperability of data
- Timing (hold to download, stale data, latency, sequencing, throughput, etc.)

Operations will have various concerns including:

- Cultures of different organizations and users
- Addressing the items from above on hardware, software, and communication
- Keeping users happy
- Drowning in massive data
- Staying alive to achieve ROI
- Localization
- Globalization
- Security and privacy

Points of failure in all of the above areas will be unseen, unexpected, and will have gremlins lying in wait. Organizations--especially testers, will ultimately have to address many of these to be successful.

Familiar test techniques will serve professional testers. Quick tests apply to applications running on IoT devices. Testers will do domain tests, integration tests, and tests designed to explore risks. The science and philosophy of IoT testing will inherit industry history. If you know the art and engineering of existing testing systems, you have a good start for IoT.

IoT testing, in practice, offers some formidable challenges. Starting to become aware of the challenges and understanding how to overcome them is the primary objective of this book.

Consider, a common IoT device--a thermostat, found in a smart home. Temperature controls adjust the furnace or air conditioning outputs. Rules might be as simple as reacting to ambient heat or cold or cooling things at night, or only operating when the price of electricity is lowest. Suppose you were asked to test an IoT thermostat? What would your first considerations be to outline a

testing strategy? What can you observe? You can see its display or on an app (running on a phone or in a laptop browser). If the thermostat connects to URLs on the network, can you see data coming and going? How can you observe the current state of the thermostat and its environment?

What inputs can be tested? Well, you can press all the buttons and tickle every part of the user interface. How can you make the device think the air temperature is hotter or colder, or spoof the thermostat into thinking it is such and such? How can you fool the thermostat into thinking it is now 2:00 a.m. when actually the time is just before lunch (11:00 a.m.) which is a much more convenient time for you to do the testing? Do you test or simulate what happens with the thermostat must be reprogrammed with codes, temperatures, and security setting when a new owner moves into the house? Do you do the reprogramming as part of a whole smart house environment? Can you hack the device from the outside internet and what things can you do when the hack happens?

Figure: Smart House will have Many IoT Device (10s to 100s to 1000s) and do users want this?
Reference: vhttp://smarthomeenergy.co.uk/sites/smarthomeenergy.co.uk/files/images/smart-home_0.jpg

What user stories need to be dreamt up for testing? Worse do you do these tests for the first release, the next release, or the last release? You will design tests to collect information needed by stakeholders. As you collaborate with them, how do you educate them about the risks of IoT? Since there are always more tests designed that can be built and run, how do you help them decide which tests rise to the top of the pile? How can you translate technical jargon like “Network latency and the unreliability of UDP messages to be dropped silently on embedded processors with no memory or resources to acknowledge as per the standard”?

What tests should be automated and how?

If you get the idea that there is more to the development and testing IoT systems than just addressing the functionality, you begin to understand the total challenges over time of IoT.

IOT examples of development impacts to testing

Development in IoT will be fast-paced, and often with a "start-up" mentality (see book 2 and 3). For years we have heard "testing is dead," "we do not need testing, we are Agile," and "I do not make mistakes" from development staff. We will cover the dance between dev, testing, and ops in the coming sections and other eBooks, but here are some first thoughts on things testers may face:

1. The Hardware is not done, but software needs to go into testing now (how is this possible?)
2. Hardware and software are in a state of flux, but testing needs get started now and be done as soon as dev is done.
3. The project does not need any testing because the effort is a startup and dev wants to stay alive after the first iteration of development, so how does the project do "something" to assess quality?
4. Hardware is done but has a problem that Dev decides to fix at the last minute using the software because it is easy to change, and there is no time for testing. What does test do?
5. The project is using commercial off the shelf (COTS) hardware and software, so effort does not need much (any) testing.
6. The project has no users/stakeholders to talk to, so dev will decide what is "needed" for the device.

Examples of how the IoT device impacts testers

As noted above Dev impacts testers, and some of these are common to most organizations, but what in IoT may be "different from classic IT testing. In this section, we present a few examples that we have encountered to date, and we expect more.

First is testers will need to deal with and test hardware. Many IT testers only deal with "hardware" issues in passing because the basic computer is generic. However, in IoT, each device will have unique hardware in the form of input and output subsystems. Yes, you will have the human user interface (UI) most of are familiar with, but the input sensors will be gathering data often from nonhuman users, e.g., hot, cold, wet, slippery, numbers in, etc. Further, the IoT device will be controlling something on the output channels, e.g., motors, on/off switch, actuators, etc. Testing includes all these input and output devices.

Next issue will be the communication channels and standards. Here we may need to test things like Wi-Fi, Cellular, Bluetooth, Near-field-communication, and many other communication standards. Worse, these communication channels "integrate" the IoT device under test with other systems. Does our testing stop at the interface to outside? Who owns the system of systems testing, e.g., the home, the factory, the car, the city, the world?

Governments are engaging in creating "smart" cities. IoT devices to control roads, traffic, energy usage, communications and many more physical systems that make the city work. Estimations are that money will be saved and citizens happier. However, Ted Koppel's recent book "Lights Out" (http://tedkoppellightsout.com/) on the power grid threat points out that many of our current cyber-physical systems are at risk to hacking including power, water, sewer, to name but a few.

Many of these systems are adding IoT without the industry and government being adequately accountable for these risk. Some of us have been writing and speaking on these cyber-physical IoT risk for years now, but only slowing is our development and testing of these systems coming to the correct levels.

Okay, you say you are working on "low risk" IoT devices, so testing is not a big issue.

Well, take for example a loudspeaker company. They might have been in business for years making consumer speakers. How was much testing of the physical devices done? More than you might think. They historically had to test: electrical, fire hazard, UL listings, speaker life, long duration test at max volume, different environments, etc. These companies learned by product recalls and lost sales, testing was needed to be done over and over. Now, take this company and add software for work with Wi-Fi or Bluetooth to play Spotify and many different smartphone connections. Does their hardware test team understand software testing of IoT devices? Likely they will be learning.

A distorted or garbled song thumping from a speaker will cause you alarm that maybe hacking is afoot. However, what about when hackers are just collecting data on your listening behaviors. Would you even know if your playlist was sent on to a marketing research team without your consent or compensation? Just like these familiar "stamps of approval" protect the consumer from a product that will set the house on fire or irritate the neighbors, are their similar assurances you can provide your customers about the privacy of their information in a connected world?

Figure: Smart Speakers)Wi-Fi speakers connected to smartphone)
Reference: https://upload.wikimedia.org/wikipedia/commons/0/0e/Electrodynamic-loudspeaker.png

What tests can you run to lower the risk of the loudspeaker not working with your favorite archive of music? You will learn within a few minutes if the devices work together. How to save yourself hours of futile integration that won't ever work because drivers are not compatible, or the basic transport of the bits is not going to work?

As a speaker company, you can post the results of simple tests on your product's support web page. It can update in real time. Where there are incompatibilities between devices, this can be published. Perhaps include an area where prospective customers can request tests be run for in use combinations.

In the healthcare field, some medical device manufactures are seeing a 50 percent reduction in mean time to repair their connected devices. They can download new software to fix some problems with a reported saving customer service costs by $2,000 for each problem resolved

remotely. Great! However, consider the hacking of pacemakers (https://threatpost.com/pacemaker-hacking-fears-rise-with-critical-research-report/120174/) how much will this cost?

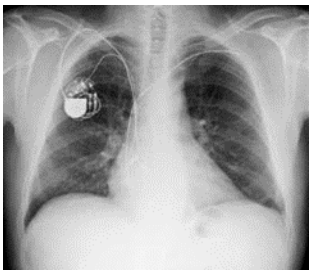

Figure: Smart health IoT device (pacemaker)
reference:
https://www.ibm.com/developerworks/community/blogs/invisiblethread/resource/BLOGS_UPLOADED_IMAGES/Getty_pacemaker.jpg

Next, consider a tire maker that is using IoT to gain valuable insights about the performance of its products in near-real time. The company is using an analytics platform to manage the vast amounts of data gathered directly from sensors embedded in the tires. The system allows the monitoring of the pressure, temperature, and mileage of each tire remotely. By keeping these factors in acceptable ranges, fleet managers can have a significant impact on fuel economy and safety with over $1500 saved per vehicle per year. However, consider the impact of being hacked the way Jeep was (reference https://www.wired.com/2016/08/jeep-hackers-return-high-speed-steering-acceleration-hacks/).

Figure: Smart Tire
Reference:
https://www.ibm.com/developerworks/community/blogs/invisiblethread/resource/BLOGS_UPLOADED_IMAGES/Getty_pacemaker.jpg

It seems like there will be a lot of learning, new testing, and new bugs making into the field.

Key Points:

Where is the “sweet” spot of just the right amount of dev and testing to be successful?
Does success change over time?
Testing is advantageous from the moment it moves a decision maker from less complete information to more complete information. More information always makes testing a success.

Who owns testing the system and system of systems?

The Industry's Rush into IoT systems: Dangers

Any new technology that enters the technology curve encounters various problems in development. For IoT, some of the problems will be new to the IoT space, and others will be familiar with other software technologies. IoT companies and people should start by understanding as many of these as possible because those that do not learn from history are doomed to repeat the historical lesson learned. This section considers likely problems that may be encountered during development of IoT devices.

One of the situations in IoT is existing companies who have never developed software for use in their products are entering the software domain by developing and releasing IoT products. We have seen a variety of scenarios. For example, we have seen companies taking existing staff—qualified or not, and putting them in charge of software development and testing, only to have unhappy users when the product is fielded. Alternatively, we have seen electronic companies start to added processor and software, often using outsource subcontracts, and suddenly new kinds of errors appear in their "new and improved" IoT device. Finally, we see companies hiring new staff to be in charge of software development and testing, only to be surprised when management of these new organizations do not understand the critical nature of some lifecycle efforts. Each of these options can be varied and mixed to the project needs, but care and team skill build are needed for success.

Another phenomenon we have seen is where a company thinks IoT is like developing Information Technology (IT), web or PC software. To be sure, in IoT systems, there will likely be IT, web and PC components and similarities. However, the IoT device itself is a mix of embedded and mobile software environments. IT developers and tester will encounter situations common to the mobile and embedded software which may be new or strange to them. For example, we find problems with the amount of available memory, limitations in battery usage, processor speeds, communication dip outs, and visibility into hardware states. Also, in the mobile/smart devices, we detect bugs related to device interoperability and system integration with unique hardware. Moreover, finally, as in traditional computing systems, one can have problems in basic functionality, non-functional features, and data concerns. All of these are risk areas for IoT systems and must be included in the development, verification, validation (V&V), and testing. Probably the biggest surprise will be the development tools for IoT. Staff should not expect sophisticated IDE's debuggers, and coverage analyzers for the code running on a resource challenged IoT device.

SEEKING FORTUNE AND A COMPETITIVE EDGE

Most of the industry is or will be seeking competitive advantage by having IoT devices. Harvard Business Review has written (https://hbr.org/2016/02/to-predict-the-trajectory-of-the-internet-of-things-look-to-the-software-industry):

"Conducted in September 2014 on early IoT adopters, the survey shows that companies are seeing benefits as they deploy IoT-based initiatives. Among the reasons they most frequently gave for adopting IoT were enhanced customer service (quoted by 51 percent), increased revenue from

services and/or products (44 percent), improved use of assets in the field (38 percent), and acquiring more information to support big data/analytics efforts (35 percent).

Respondents said that they have deployed or plan to use IoT in many areas, including asset tracking, security, fleet management, field force management, energy data management, and condition-based monitoring. Moreover, they give it high marks. For example:

62 percent say IoT somewhat increased or significantly increased their customer responsiveness

58 percent say it increased collaboration within the business

54 percent credit it with increasing market insight

54 percent believe it increased employee productivity."

As the quote indicates, many (most?) companies are entering and rushing into IoT since they believe there is money to made. The money /
"will come from the devices, the data, and analysis. The product space likely to come with a variety of problems.

Challenges which should be assessed by V&V/testing and development of IoT include:

Physical environments where the device will use need to be addressed by development and testing - Hagar quote
IoT has unique aspects of privacy and compliance in regards to regulations – Harvard #1 – 46%
Is security and privacy adequate outside of regulations (future proofing)– Harvard - 28%
Machine-to-machine communication and interaction will be fast and frequent with IoT- Hagar
Managing the numbers of devices and data generated will be a challenge – Harvard - 35%
Interaction and usability will be a key to success of IoT- Jeff Yakmora
COTS/vendor software and hardware reuse will spark development speed while still needing testing - Hagar
Interoperability with different hardware, software, and networks will be a challenge for many years to come -Hagar
Skills of the team to do data analytics (and development) will be issues – Harvard 39%

These factors will change the way we create, products and field IoT products. Users are sophisticated and will expect ease of use, simple solutions that provide more benefits than problems, and fun. Most users do not want to know their devices are "smart" computers.

The concern with these problems may be enough to make companies reluctant to move into the "high tech" IoT space from their more traditional endeavors. Some companies that do not make the transition to IoT, may not remain viable businesses. We still have traditional (analog) Swiss watchmakers, but many Swiss watchmakers did not make moved to digital. They left the market to Asian companies and Swiss companies lost watch market share. Furthermore recently, the traditional watch market share has dropped worldwide as many people do not even have traditional watches, instead, they are using their smartphone and/or wearables to tell time. To stay competitive, there will be companies who move traditional products into the IoT world and many other new companies that will conceive new product in the IoT space. It is likely that a majority of

companies will move into IoT even if they do not have software development and testing experience.

Even with this expansion, companies and users will look for the killer IoT apps and excellent device configuration. There will be market volatility at the same time customers expect "quality" (a value that someone is willing to pay for). Exactly what quality is will change over time and with different products, but users of high tech have increasingly had higher expectations of functional and nonfunctional elements. For example, the early smartphones only partially looked like the smartphones that now dominate the market today and users are quick to post bad reviews in social media regarding their opinions of apps and devices.

Figure: Market Darwinism
Reference: http://blog.heartland.org/wp-content/uploads/2014/03/Evolution-Preview-Image.png

Companies and people looking to start IoT projects should consider marketplace Darwinism. To survive one needs a good idea developed to be just "good enough" to reach sales. Information provided by testing is part of how software companies determine if a software product is good enough. The software industry is littered with good ideas that were not good enough and projects that did not deliver on a good idea. Various industry statistics indicate between 30%, and 60% of new software project have major delivery problems or fail ever to deliver a product. The exact number is not that important for companies looking to move into IoT, what is important is that newbie companies in IoT understand that the likelihood of first time failure in IoT is high and that many of the items in "Common Situation and Potential Solution" of the eBook should be considered. That is not to say that all or any of the ideas in the section must be implemented, but lack of ongoing planning has resulted in many "not good enough" products.

Development, testing, and Operations: Systems, Hardware, Software, and Communication-Integration

One aspect that makes IoT different from traditional IT software focused system is the integrated communications that must take place between unique hardware, specialized software, networks, and operations. In eBook 2 we will talk more about these environments and factors teams must

consider. To be sure, these are all significant areas, each of which could or do have dedicated books and references. In this chapter provide a few lists to get you think about the areas and these factors. You can use these list to look up more information on the internet as well as other references or eBook2.

System efforts can include the following areas:

- Requirements
- Modeling
- Quality characterizes and goals
- Interfaces and integration
- Allocation to hardware, software and/or ops
- Communications
- Data allocations and analysis
- V&V/test
- Management
- Support functions, e.g. CM, QA, measurement, improvement, marketing, etc.

Hardware engineering is needed including

- Lower levels of requirements
  - Electronics
  - Mechanical
  - Packaging
- Design
- Implementation and manufacturing
- V&V/test
- Data allocations and analysis

Software efforts are needed including

- Lower levels of requirements
  - Commercial off the shelf (COTS) software
  - Vendor provided software
- Design
- Implementation programming
- Data allocation and analytics
- V&V/test at component and integration levels

Operations work can include

- Lower levels of requirements
- Design
- Implementation
- Data allocations
- V&V/test

Support functions spanning all areas include

- CM
- QA
- Measurement
- Testing/V&V
- Leadership and management
- Marketing

If you get the feeling that there are many things to do in development, you likely still underestimate the job. The reference list has standard and book measuring the hundreds of pages that readers of

this eBook should probably be familiar with and/or consider for use. If the above concepts are unfamiliar to the reader, more reading on some of the reference lists are in order.

We find companies and teams that have expertise in one or two areas will tend to underestimate and appreciate other areas in IoT. Each group and company will be at different levels of maturity and understanding in these lists.

A key for us is that groups must know where they are strong and where they are weak. We have been called in hardware companies moving in software and software testing who knew they were breaking new ground. We have talked with staff at software companies, who seem to say they thought hardware, system, or ops would be "easy."

It is risky to under or overestimates knowledge and maturity. We do recommend teams moving into IoT do some assessment of these areas. Areas of weakness or overconfidence should be considered risks. Further, as an effort continues, ongoing assessments of progress is not a bad idea.

Startup companies will likely do the best they can within the limited budgets and schedule these groups typically have. The mature large companies will have more resources to pull from and hence likely few weak areas, but overestimation can be a risk in and of itself. Comparison to standards such as ISO can provide a baseline. Note, we are not saying follow or use ISO standards as they exist. We believe standards like ISO, IEEE or other industry reference points must be tailored. A standard is not the "bible" but information useful in triggering thought during the comparison. We have worked with places that only used 10% of a standard while being very successful. We have known of groups that used more of a standard and failed.

Verification, Validation, and Testing Concepts

Many traditional software testers and projects are not familiar with concepts of Verification and Validation (V&V) as defined by standards such as IEEE 1012 and ISO 12207. The lack of familiarity may be because many V&V concepts are related to system and hardware evaluation or because aspects of the software industry have focused on a broad definition of testing and Quality Assurance (QA), which include V&V. However, in IoT, because we feel that product quality assessment activities need to "go beyond" traditional IT software testing as defined in ISO29119, we outline in this section the concepts of V&V, and associated historic industry ideals and IEEE 1012. Readers following these concepts and wanting more details about V&V and industry integrity levels should obtain and reference IEEE 1012, other V&V references and standards such as ISO 29119, but these are only starting points and not addressing "state of the art" which much of IoT is or will be.

V&V are part of building a system including hardware, software, and operations. IEEE 1012 has decades of use in various critical items. V&V provides evaluations and assessment of development products, e.g., requirements, designs, documents and the end deliverable user products. V&V are done throughout the lifecycle of development and test staff with the purpose of helping the organization deliver a product with "good enough" quality. The definition of "good enough" evolves

as a product matures, e.g., an early stakeholder prototype may not be as robust as the mass-produced device. Good enough is determined if the deliverables are correct, complete, accurate, consistent, and testable with the context of a delivery cycle. Verification checks whether the development products of a cycle conform to the previous cycle's activity, e.g., does code verify to design and/or requirements. Validation assesses if a product satisfies user (s) needs.

Key Points

Simple was to remember these are:
Verify – did we build the product right (compared to a reference point)
Validate – did we build the right product (compared to user expectation)

Techniques used include assessment, analysis, evaluation, review, inspection, demonstration and testing of products and processes. V&V is done during and with development, often by development staff and not "at the end" as an afterthought, e.g., you cannot test quality into a product once the product is done.

V&V supports development, management, and other stakeholder interests. Stakeholders can include, customers, third parties (e.g., regulators), and external engineering groups. V&V provides a source of information and data to these parties. Information can include functional and non-functional qualities such as performance, reliability cost, schedule, etc. (refer to ISO 56618). Test information data can include models input data, results, performance numbers, error reports, inspection notes, analytics, etc. Parallel development and V&V allow the feedback of V&V data to development for quick product improvement, e.g., correct errors to avoid technical debt, understand performance issues, etc.

Details of V&V are contained in IEEE 1012. IEEE 1012 provides V&V process, integrity level criteria and usage, and V&V planning information. Readers in critical IoT product domains, e.g., health and safety can benefit from a copy of IEEE 1012, but it may not be required except by contract.

V&V can be applied to devices, networks, systems, hardware, software, data, and systems of systems. All of these are addressed in IEEE 1012. IEEE 1012 does not provide specific information

about V&V/test approaches or techniques. These can be found in references and standards such as software test ISO 29119.

What is IoT Verification

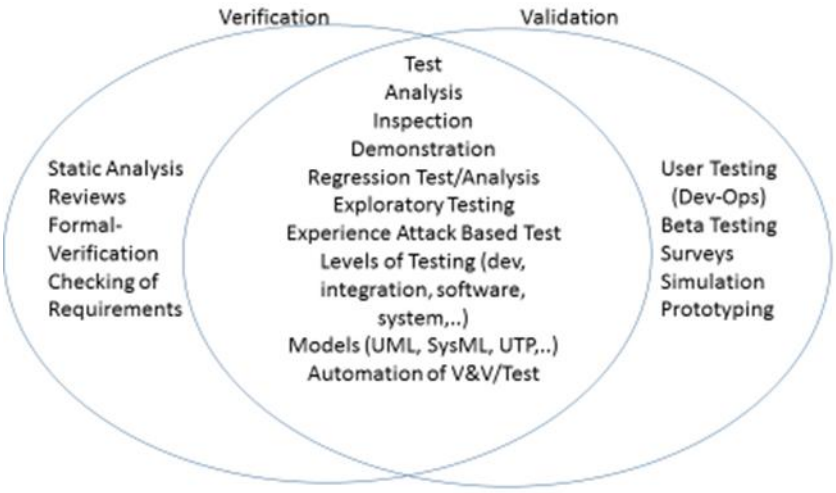

Figure: IoT device Verification and Validation – hardware, software, and system
Reference: Jon Hagar IoT Class 2016

Verification is a development activity which involves testing but also uses other assessment approaches. Verification tries to answer the question "did we build the system-hardware-software right?" Verification uses a variety of concepts to answer this question. Activities can be done by system, hardware, software, and test staff throughout the lifecycle. Concepts employed include (see definition section): inspection, analysis, but testing is often preferred.

Verification takes the artifacts of one lifecycle stage and assesses if they satisfy the information from a previous step. For example, verification will assess if the requirements are met by the software design and implementation (aka checking as noted by James Bach). Verification assumes that source information, e.g., an operations concept, requirement, design, etc. is "correct and complete."

What is IoT Validation

Validation is also a development activity which involves a variety of concept. Validation tries to answer the harder question "did we build the right product for the user." It is often stated that validation is done at the end of a product's lifecycle, but this is a simplification and can have a cost

impact. For us, Validation should be accomplished on each product to be delivered and done throughout the lifecycle as the product matures. So for example, it is a good idea to conduct inspections (peer review with a customer) on requirements to validate the requirements is going to meet the needs. This is why Agile encourages direct user involvement as concepts like stories use cases, and even the code is produced. Further, for some critical requirements, such as the control laws for a self-driving car, validation may need to include modeling and simulation before moving to design. Then the modeling and simulation can continue throughout the car life cycle (disposal).

For more information, please see IEEE 1012.

What Is It Software Testers Do: Testing?

Of all the challenges in software development – from architecture, requirements gathering, development, project management, testing, and support – in our opinion, nothing is more intellectually diverse and demanding than software testing.

Good testers are known for the variety of tasks they do – just like this beautiful summer crop of all kinds of tomatoes.

Figure: Many Kinds of Fruit
Reference: Scott Allman

To see the many skills required to be a successful tester let take a short historical tour. Then we will identify four very different roles software testers play in the development process. Finally, we will show some recent data from three different sources that confirm this picture of the demanding skill known as software testing.

In the Beginning …

Although we have many books and articles from early gatherings of software testers, including some hardbound volumes written with the charming typewriter fonts of days gone by, the most famous early work is Glenford Meyer's 19xx book, "The Art of Software Testing". Here is how he describes software testing:

"Software testing is to find bugs." - Glenford Myers

Like a Zombie that keeps stalking and menacing our profession with the curse, "you – find – bugs" this attitude still haunts us. Software always has bugs. We no more can find them all then can we guarantee "This software is ready to ship because it has no bugs." Yet, that was the early paradigm. To his credit, Meyers goes on to explain a view of software development that is much richer. If you have his book be sure to re-read the interesting chapter on bugs that occurred during testing but were not observed and reported.

Building on this perspective is the prolific author of books in the 1990s, Boris Beizer.

"Design, coding and debugging are done by developers. Testing is done by testers." - Boris Beizer

Alas, he too sets an attitude that later software engineers would be obligated to fight, and we are still fighting. A featured debate at the 2015 annual Conference of the Association of Software Testing featured a debate on the subject. However, with the introduction of the JUnit testing framework in 1999 by Beck and Gamma testing, at least unit testing, became a responsibility shifted to testers. Don't offer you code to be tested until all your unit tests pass.

Next, we see the beginning of basic broadening of a testers scope of tasks. In this quote from agile testing guru, Lisa Crispin is the first hint the software testers should contribute to design decisions and classic quality assurance ("build them in a good way")

"We help the business, and the development team decide the right s/w features to build, and we help build them in a good way so that valuable software is delivered frequently and at a sustainable pace for the team." - Lisa Crispin

The next two quotes, from Lee Copeland and Doug Hoffman, emphasize the role of a tester to provide information. Of course, bug reports are part of this task, but they are not the whole story,

"Software testers do things to create, organize, and distribute information about the quality of the product so that others can make better decisions." - Lee Copeland

"Testers provide [quality] information so others in the organization can make better-informed decisions." - Doug Hoffman

Michael Bolton is one of many testers who realized the value of exploratory testing. It is a process, with many excellent articles to read, where a tester uses their skills to discover and report useful information to stakeholders.

"Software testers investigate the product, learn about it through experimentation, and report on what they have found to help their clients determine whether the product they have got is the product they want." - Michael Bolton

Moreover, finally a quote from a book authored by a trio that includes the sage Cem Kaner, succinctly captures the spirit of software testing:

"You are the headlights of the project." – Cem Kaner, James Bach, and Brett Pettichord

The evolution of "what software testers do" should be seen in the difference between the first and last quotes. Do we just hunt bugs and make trophies of the ones we found? No, we provide all sorts of information to decision makers.

Hidden in these ideas are four very different types of software testers. We will label them:

The Journalist: Objectively gathers, reports observations, and advocate for bug fixes.

The Experimenter: Designs experiments and builds an apparatus to run them.

The Technician: With careful attention to detail runs experiments.

The Trusted Colleague: Supplies timely, valuable information to decision makers.

Of course, individuals will have these traits to different degrees.

The Technician does the work mandated by Meyers and Beizer. Tests are run and re-run as needed. The traditional skills needed are attention to detail and reliability. Tests must be run, observations recorded, and results truthfully reported.

However, where did the tests come from? That is the role of The Experimenter who designs tests to get empirical evidence about the behavior of software. Partly a statistician they know how to design experiments and create samples of data. Furthermore, they know how to "build" the tests either using simple operating system tools or sophisticated test automation frameworks. After all, designs of tests that cannot be run are worthless designs. And, they work with decision makers to understand the questions that must be addressed by the tests. As an example, the BBST course on Test Design is a survey course that covers, at a superficial level, 70 different test techniques. In the toolbox of a good experimenter are dozens of software testing techniques and handy operating system tools to demonstrate software behavior.

Designing, building and running tests is only the beginning. To turn the results data into useful information is primarily a communication task requiring The Journalist. Basic questions all too familiar to every journalist (who, what, when, where, and why) are the heart of this skill. Referring to another of the excellent BBST classes, "Bug Advocacy", we repeat this quote about the class:

“Bug reports are not just neutral technical reports. They are persuasive documents. The key goal of the bug report author is to provide high-quality information, well written, to help stakeholders

make wise decisions about which bugs to fix.”

To elaborate, here is more information about the course, and of course, the skills needed by the Journalist.

- Bug reporting as persuasive writing
- Bug investigation to discover harsher failures and more straightforward replication conditions
- Making bugs reproducible
- Lessons from the psychology of decision-making: bug-handling as a multiple-decision process dominated by heuristics and biases.
- Style and structure of well-written reports

At the beginning of a project, a requirements analyst is just a reporter, a journalist.

All of these roles are subordinate to the role of a software tester as a Trusted Colleague. Excellent relationships with all other members of the software team are the key to your success as a software tester. Your colleagues the architect and developer provide critical technical information as your design and build experiments. Your colleagues in marketing and product/project management steer you towards questions that must be answered. Your fellow software testers are your teachers. You cannot test just by yourself.

Why trusted? Because the information we generate almost always has political and business implications. We speak truth to power. We must have evidence to back up our claims. We must transparently conduct investigations. Without Trust, we cannot do our jobs.

What do employers, trainers, and software testers think it is that software testers do?
With this rough classification of four roles in mind let’s look at how it corresponds to three perspectives on software testing. First, what do employers think it is that software testers do? We collected job descriptions from hundreds of online postings and statistically analyzed them for skills and responsibilities.

Figure: terms from employer data
Reference: Scott Allman

Ignoring generic wishes such as “experience,” “testing,” etc. we see the words “development” and the associated terms “automated” and Linux. It seems that the role of the Experimenter is important here.

The second group we investigated are the trainers and certifiers of software testing. We scanned in many a syllabus, course review, and marketing blurb from these organizations that offer certification and training:

- Quality Assurance Institute
- Association for Software Testing (the BBST classes)
- International Software Testing Qualifications Board (ISTQB)

Notice different terms in the heat map from their data

Again we ignore generic terms. We see a predominance on skills that are associated with The Technician: SDLC, cases, Tester, Defect(s), Quality and support.

Finally, to round out this quick survey about software testing, we turn our attention to software testers themselves. There are dozens of national and international conferences on testing held every year. We scanned in the promotional materials and abstracts of the presentations for many conferences. When conference archives were available, we added those as well. What are the topics of current interest to software testers? What topics do they feel are important?

To summarize, what is it software testers do?

- Design, build OR run experiments
- Communicate observations OR analyses
- A trusted source of information

Using Integrity Levels in IoT Devices to Allocate V&V/Test Approaches

Of consideration in IoT is that not every device and associated software will have the same level of importance to the users. Each device will have varying degrees of critical use. Criticality ranges from minor user annoyance when something fails to people dying where lawyers get involved. It almost goes without saying the amount and type of V&V/testing will, therefore, need to vary. IEEE 1012 Verification and Validation Standard defines different integrity levels (ref) for V&V. We summarize and present basic ideas from IEEE 1012, but for any project needing more details and specific considerations we do recommend obtaining IEEE 1012 which has much more detail on integrity levels, the impact from their use, and how to classify different types of critical software.

Integrity levels provide a numeric ranking system to help in determining the amounts, tasks, rigor, activities criticality and approaches for V&V. The integrity determination is based on rankings of complexity, criticality, risk, safety level, security level, desired performance, reliability, or other system-unique characteristics of systems, software, and/or hardware. The producer, users, and other stakeholders concur on the determination of an integrity level. In IEEE 1012 extensive classifications tables are provided to aid in the determination and then associated activities of V&V based on life cycle stages.

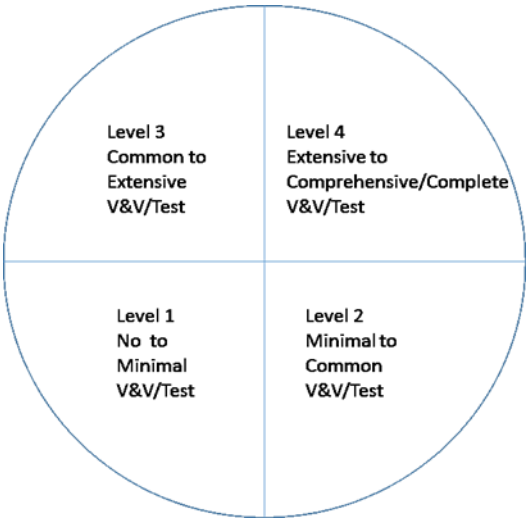

Figure: example division of the component integrity determination.
Reference: Jon Hagar IoT Class 2016

Our four IoT integrity levels are shown in the figure above and summarized in the table below. While based on IEEE 1012, ours is different and simplified for IoT. The table explains each level. The table is a starting point guide and should be tailored to an IoT project context.

Table: V&V/test level classification and examples

Nature of level = 1

Less critical device

Device failure minimally degrades functionality or non-functional criteria

Hardware is simple

Software is simple

Communication interfaces are well defined and based on industry standard

Operations and data analysis is none to minimal

Examples –

New or prototype devices used for demo

IoT component with a backup (first one must be tested)

Consumer "play" toy, such as a toy racing car

Wearable sport band (non-medical)

Nature of level = 2

More critical usage

Failure has low impact or degrades functionality and non-functional criteria (1 – 3 features)

Hardware is moderately complex

Software is moderately complex and interacts with other users

Communications are moderately complex and/or may use several media approaches

Operations and data analysis are moderately complex and can impact business success

Examples –

Loss of money is possible (1000 to 10000s)

Losing part of device mission

Impact business base and future sales (moderate)

Home device that has security implications

Nature of level = 3

Major criticality

Failure has impact or degrades functionality or non-functional criteria (50% of features)

Hardware is complex

Software is complex

Communications are important to success

Operations and data analysis are complex and critical to business success

Examples –

Major loss of money

Losses large part of mission

Impact major business base or sales

Nature of level = 4

Life or sever business impact (go out of business)

Failure has impact or degrades functionality or non-functional criteria (75-100% of items)

Hardware is extremely complex

Software is extremely complex

Communication approaches are critical for success

Operations and data analysis are complex, critical, and large (big data)

Examples –

National news and loss of money

Total Losses of mission

Impact major business base (goes out of business) Self-driving car Pace maker systems integrated with hospital

The following table provides example recommendation of V&V/Test activities that might be associated with integrity levels. These are only examples as many possibilities exist.

Table: Examples of Recommended V&V/test activities

Level 1 Example 1: New prototype IoT device – No written test plan/ Developer testing/ Fast exploratory testing Example 2: IoT toy car - One-page test plan/ Agile testing stories/ Some test automation with exploratory test Level 2: Example 1: Home IoT thermostat – Written test plan with strategy/ Developer testing attacks/ Exploratory testing/ Security Attacks Example 2: Wearable watch – Written test plan/ Developer testing/ Network testing/ Exploratory testing/ Security attacks/ Functional attacks Level 3: Example 1: Home-Personal medical device – Written test plan conforming to standards/ Developer attacks/ Test automation/ Scripted testing/ Conformance testing/ Functional and nonfunctional testing/ Security testing Level 4 Example 1: Self driving Car – Full up V&V and test planning / V&V over full life cycle / Developer attacks/ Test automation/ Scripted testing/ Conformance testing/ Functional and nonfunctional testing/ Security testing/ model analysis and testing/ safety analysis

Example 2: IIoT lighting and traffic control system -– Full up V&V and test planning / V&V over full life cycle / Developer attacks/ Test automation/ Scripted testing/ Conformance testing/ Functional and nonfunctional testing/ Security testing/ Chaos testing in field/ Network analysis and testing

Example 3 Factory IIoT control systems - – Full up V&V and test planning / V&V over full life cycle / Developer attacks/ Test automation/ Scripted testing/ Conformance testing/ Functional and nonfunctional testing/ Security testing/

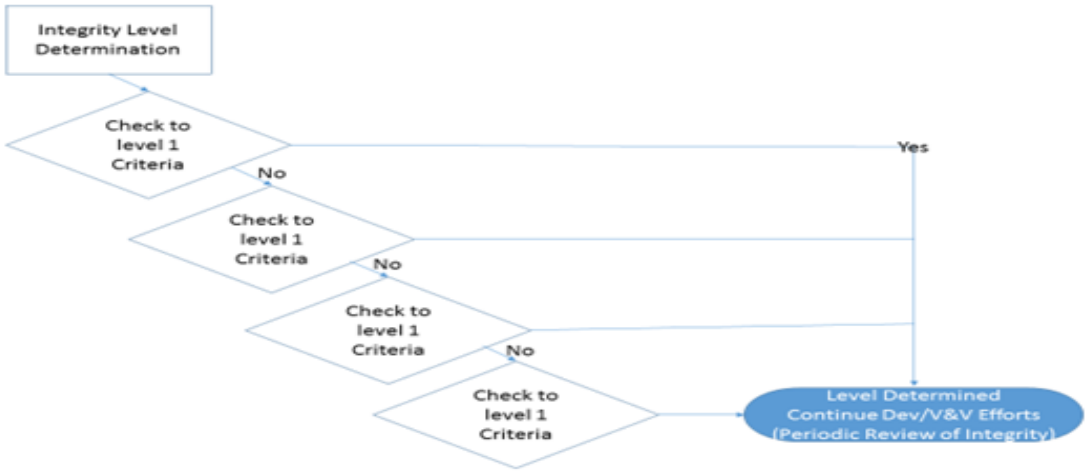

Figure: Determination of Integrity Level
Reference: Jon Hagar IoT Class 2016

We follow IEEE 1012 with 4 levels since this is our reference point, but nothing is preventing more levels or fewer, but for us being less than two makes little sense. The table above is a version modified for this eBook and IoT devices/system. Our integrity classification approach is very simplistic and to be used as an example starting point. For a more rigorous approach, please see IEEE 1012. The figure above shows a schema to determine integrity levels. It can be modified and customized for the local context. If a more rigorous approach, again, refer to IEEE 1012. The determined integrity level will associate with V&V/test efforts and activities.

It is possible for larger IoT system to have different components of the system to have different integrity levels, and therefore different V&V approaches and efforts. However, some care must be taken when working with sub-contractor and vendors that the flow down of integrity levels is thought out. It is best to apply to integrity analysis recursively to each element and sub-component of the IoT device and/or system. This analysis may take more effort for devices that are found to be of higher integrity.

For systems with multiple elements, analysist tasks should be used to establish integrity levels for the associated subsystems and vendors. This analysis may impact planning, requirements, functions, hardware, and software. Software and hardware elements or components should be treated as a function of the IoT system. Thus IEEE 1012 requires a system to assume at least the integrity level of the highest integrity level component it contains. For the elements contained within a system, it possible to for a lower level component to assume a smaller integrity level if the recursive analysis determines this is possible. When working with COTS, selection of the COTS element should use integrity levels as one of the selection criteria.

IEEE 1012 views Integrity levels as characterizing and defining the likelihood of system problems resulting from:

1. failure to meet the trust and validity expectation of users
2. fault resulting in system failures
3. unverified quality characteristics

As integrity levels increase documentation and formality of information control, e.g., test and results, will like grow in importance. High integrity levels often imply that legal and regulatory actions will become possible. Records feed such formal system and offer protection to the producing organization. Likewise, the involvement of vendors or contract items implies increased levels of documentation and formality. Added documentation does not say that Agile ideals such as "working with the customer over contract adherence" or focus on code should not be considered. However, for those of us involved in legal actions have documentation to back us up proved decisive.

Once integrity levels are determined, tailoring of actual V&V plans, process, and activities can be done. These activities are outside of IEEE 1012 and this first eBook but are discussed in other eBooks of this series. There are many tasks, processes, techniques and resulting documentation which are possible for IoT. Generally speaking, the higher the risk and integrity level, the more V&V/test activities to be considered in planning and implementation. The integrity levels define some minimal activities, but these are not hard and fast formula which give "the answer." Context and critical thinking are needed.

Further, integrity levels and risk should be evaluated and updated during the product development lifecycle and the products life. For example, the story of wearable fitness devices that "morph" into being used in medical decision making should be considered. Teams should expect (maybe even hope) that the market and usage of the IoT devices will expand and change. Such movement can change risks and integrity levels.

Changes to risks and integrity may mean more V&V/test activities become necessary. Level of integrity must be determined during project planning starting at proposal time and then continue until a device is retired from use. The assessment of integrity throughout the life of a product is needed because the uses of an IoT device may change over time. It likely is not enough to say,

"well the product has been used for x months/years, so, therefore, it is proven." For example, software and a device that is being used in a new environment and use case may encounter new errors. Such cases have been seen when a car was put in an extremely cold environment and encounter use cases which errors that had never been tested. Claiming "prior proven use" will likely not provide shelter. New V&V/test addressing the new cases and even previous lifecycle stages may be necessary.

Key Point: You need agreement with stakeholders

Whether lowering or rising integrity levels, the change should be agreed with stakeholders. The interplay of components, hardware and/or software, should be considered in changes since these are part of the "system." IoT systems are software and communication intensive system. We have learned that couple and cohesion impacts in the system can have ripple effects whereby lower integrity components impact higher integrity functions. IoT interconnection must be considered in establishing and change integrity levels.

Determining integrity levels applies to products provided by vendors or as COTS items where a new usage scenario may be tempted to assume a product is "good enough" and stable because it has been in use under a set of scenarios. Industry usage has seen historic COTS product placed in a new environment and usage scenario that then experience problems because the integrity level in effect changed, but additional assessment V&V/test under the new usage was not done.

The selection of integrity levels and risks should be done by skilled staff with the support of stakeholders as needed of both developed and vendor supplied components. It may be possible for low integrity levels of simple IoT devices to have "no V&V/test," but the user may be very unforgiving if a device fails to function and worse of "bad" things happen.

Risk-based development and testing

Closely associated with integrity level is the idea of risk. Lower integrity levels imply the IoT product overall has a lower risk. However, every product comes with risk and V&V/testing deal with risk. We thus advise that what your integrity level, that testers conduct risk-based exercises and maybe even risk-based testing.

Many authors and standard on project management, development, and testing talk about risk analysis. Risk analysis is a big subject with many books and classes [ISO 29119 - 1, 2 & hagar]. In this section, we want to give quick start readers a beginning, but it should not be an end to risk analysis and associated risk-based testing

The SEvocab [3] defines risk as
Reference: https://pascal.computer.org/sev_display/index.action

(1) an uncertain event or condition that, if it occurs, has a positive or negative effect on a project's objectives (A Guide to the Project Management Body of Knowledge (PMBOK(R) Guide) -- Fourth Edition) (2) combination of the probability of an abnormal event or failure and the consequence(s) of that event or failure to a system's components, operators, users, or environment. (IEEE 829-2008 IEEE Standard for Software and System Test Documentation, 3.1.30)

We recommend teams conduct a risk assessment, even for a level 1 integrity device. Assessments include management risks, development risks, and product risks. Risk play into integrity levels (ref that section in this eBook). Many teams and most managers tend to focus on management and development process risk, which are those things that may impact cost and schedule. Often managers and teams overlook product quality(s), which is a test concern, until the point that quality (poor quality) impact cost or schedule, but then it may be "too late."

Indeed, in IoT, many start-up efforts only care about getting a product to market (schedule) for as little cost as possible since they usually have minimal money. This rush is to be expected in many IoT projects. However, assuming the startup has just enough quality and functionality to clear the start gate, many IoT product will quickly worry about quality risks, which lead them to have testers.

A tester starts risk analysis with information collection, which can be done using interviews, history assessment, experience, review of taxonomies, independent checks, workshops, checklists, organized brainstorming, and/or customer interviews. The knowledge coming from these efforts should be captured and recorded. Capture can be done using note cards, spreadsheets, tables, text files, tools or even on whiteboards with pictures. There are risk tools and systems to help in doing formal risk analysis, but most of us keep it as simple as possible for the context at hand.

A risk statement creates a written definition of the risk to aid understanding by stakeholders and possibly drive test efforts. Classically, risk descriptions capture a single condition followed by details of the potential consequence (potential problem or risk). One way to do this is with a statement structure of:

<if condition>, then <consequence(s)> + <time factor>

If Condition — a single phrase citing a single key circumstance or situation that can cause the concern, doubt, case, or uncertainty. The "if" should not have an "and" since this can indicate multiple risks.

Consequence — a single phrase or sentence describing the key, an adverse outcome of a condition. A consequence can have "and" statements.

Time factor — a single phrase or sentence that captures a time factor or implication of a time factor of when a risk can or may occur. Time factors are optional.

Note on the above: For more detail and to reference, see "Software Test Attacks to Break Embedded and Mobile Devices" by Jon Hagar.

Examples:

Testers should refine these common areas of IoT risk during risk analysis.

1. Safety—when the well–being of humans is threatened.
2. Security and privacy—data or information can be exposed.
3. Hazard—damage to equipment or the environment is possible. Hazards can include hardware within and outside of the device.
4. Communications—loss of information or control caused by communication channels, interfaces, and protocols. Comm issues can include dropouts, bad data input/output, internal and external comm to the device, slow comm lines, etc.
5. Business Impact—bad computations generate wrong information and ends up impacting profit.
6. Regulations and Legal —the product could result in harm, not compliant with standards, or be at odds with government regulations, leading to legal actions.
7. External environment factors—impacts from hardware inputs for devices and electronics, which are susceptible to influence (noise) either systematic or random, including outside communication lines and characteristics; the "real world" (weather or conditions in the real world such as wet roads or rain); and even human operations.
8. The impact of input and output noise—input sensors or outputs to devices and electronics that are susceptible to noise influences.
9. Complexity—the size of the system or some aspect of the system makes missed cases likely.
10. Compatibility and interoperability – the ability to integrated with other systems, since often IoT will be in a system of systems.
11. Quality factors not met – There are many qualities of a device or system which if not met, may be a risk (see other sections and books for discussion of qualities).

Testers should use these lists as a checklist to trigger thinking leading to risk statements.

Many projects in larger historic organization will have formal risk analysis process (ISO ref). Testers should support and build on these classic events. Our experience is that many items that tester deem as a risk will not make it to management level formal risk process. This is fine. We have often kept and worked our "lower level" development and test risk lists and analysis within our teams. Such actions are done until such a time as the lower level risk is addressed or bubbles up to a formal management-level risk list.

Story Sidebar: On one effort, we were dealing with a hardware-software vendor that was to supply a chip-computer system, software, and communication-interface protocols. One of the testers had personal experience with the vendor and warned that there would likely be both hardware and software issues based on another historic project. Management said they trusted the vendor and had other risks that were higher, the no project level risk was assigned to hardware and software from the vendor. However, the development and test team tracked this lower level risk. In the team sprint planning, they asked the vendor for a prototype drop of this partial system. The vendor committed to delivery date. The actual delivery was 5 months late. The first warning bell was sounded.

When the IoT hardware and software was delivered, the development and test team has an immediate sprint priority (no waiting) to receive, inspect, integrate, and communicate with the system. This happened in just a few hours:

1. Hardware plug was wired upside down and backward – vendor said to ship it back
2. Development and test team did not ship the system back but wired a "test harness" to allow the integration to continue
3. The test team then found a bug in the software (actually a series of bugs) upon startup
4. Management was notified and put the vendor/product on the project risk list
5. Testing continued with new drops and hardware and software
6. Bugs continued
7. By the time product rollout came, the vendor was the number one project risk and almost a ship date, but the development and test team had work the vendor risk as a high priority, so the shipment date was met. Yee haw

Moral of the story: test often, focuses on risk, communicate to management with data.

We recommend that groups work on risk identification early and often. Somebody with test knowledge should support some formal efforts. Inside of a test group, the developers and testers should consider risk. Risk analysis can be done as part of Agile team efforts if Agile is being followed or as informal team communication. We recommend the team effort because one gets a

comprehensive set of risks, but individual tester can do test risk analysis by themselves, although it may not be as effective.

Internally, development and test teams should focus on technical risk. We can use the risk as the story told to focus and assign integrity levels of V&V/testing. As the project matures and more information becomes available, risks will change in priority. Testing may not address all risks since usually the number of risk and area to test exceed budget and schedule, but as a standard like ISO 29119 details, risk-based testing can manage the test planning process. The resulting risk statements get captured in a table such as the Table below.

Table: Example Risk Statement Table

Risk Statement for Smart Diaper	**Impact**	**Likelihood**	**Test/Note**
If the hardware from vendor does not have proven wet sensors working by sprint 10, software design may be unknown	High	Medium	Conduct A2D and D2A Wet Sensor tests
If the WiFi communication protocol cannot handle sensor data rates, data dropouts may occur (wet user)	Medium	Medium	Performance test comm

Once risk statements have been defined, they should be prioritized and integrity levels assigned (see IEEE 1012, ISO 29119, and "Software Test Attacks to Break Mobile and Embedded Devices). Setting priority for cases and risks can be hard. The literature talks about many methods, but maybe the easiest is just to decide how many tests you have the budget and time for during this attack effort, and then sort the testing into buckets: "test," "maybe test if there is time" and "don't test," Teams can balance budget, schedule, and risk in the test plan.

Once dev and test teams start risk analysis and RBT, the number of risks will grow. Teams should review materials with stakeholders. Periodic risk review is a good practice. However, stakeholder must know they cannot get every risk thoroughly tested within limited budgets and schedules. Using risks to do the planning and test design is a learned skill that will benefit IoT teams. Good testers think risk and run scared.

Note:
If you are interested in a more detailed treatment of risk and failure analysis, see Jon Duncan Hagar's book "Software Test Attacks to Break Mobile and Embedded Devices"

Organizational Ability Levels Entering into IoT Dev and Testing

We find a different organization and people active in the IoT world. Teams range from startup organization with a great idea and maybe little experience to big "mega" corporations with tons of experience. In this short section, we outline these. In later books, we will expand our thinking for the organization.

We have this breakout because how the organize proceeds with IoT development will be different depending on the abilities of the organization and the context of the IoT effort. We suggest you consider our breakout of ability levels and find one that seems close to your organization, while recognizing the classification represent what a continuum is, and your organization will not precisely match one of our breakouts but be somewhere in the continuum.

We hope this helps.

"Newbie" Companies- Level 1

We expect like in the days of the web and .com world there will be people setting up new companies hoping to be the IoT version of Google or FaceBook. There certainly will be many of this startup, and a few might be the "Google."

We do not pretend to be experts in startup companies, so what we write here we have gathered from friends, conversations, and some research. If you have different or added experience becoming and growing a startup, please feel free to contact us. We all get to learn.

Here are some of the items a Newbie company should consider to do more research and work on:

1) Money – Get funding via the Crowd (ref) network or self.
2) Stay alive on the first version - Get enough working product to stay alive, and this may mean no formal testing, but a lot of developer testing and maybe some fast exploratory attacks should be considered.
3) Hardware – Use as much generic of the shelf hardware as possible and find a good source manufacturing (example: China and see book 2).
4) Software – You can do it yourself but go Agile (see reference books).
5) Production version – define what "good enough" is for your customers and users but expect changes once you go live.
6) Assumed Risk - Expect problems, so some testing can help these from being "big ones" but assuming risk will part of the game.
7) Schedule – Most every experimental project takes 2 or 3 times longer than you first expect, so again be Agile.

Probably the most significant area is funding, followed by "staying alive." We cannot say what your level of assumed risk will be nor who your stakeholders are. We know several people who have played the so-called "silicon valley lottery" (moving from startup to startup hoping to hit it rich). They are always hoping to "hit it big." We know of a couple who did pretty well, but it is a gamble.

We would not risk more time, money, or reputation that we can lose. We also keep in mind that "good enough" is always different and changes over time.

Good luck.

Companies with some experience and success moving into IoT- level 2
We see and know of many companies moving into IoT and software which are not traditional software IT companies. They may have an IT department, but they did little or no software development for sale. Further, we know of great software companies who have little hardware or operations experience. Most companies are or will be moving into IoT and so will have different areas of expertise and areas to work. We outline areas here.

Company with Hardware (electronics) experience
Companies that understand hardware production, and maybe even have had some classic embedded software experience in their product will find the following IoT differences:

1. Communication with devices and to the network
2. Huge data volume to do analytics on
3. Ongoing operations and ability to update software on the fly
4. Security and privacy
5. More complex graphical user interface that at the same time must be simple to work
6. Many protocols and standards which are fluid
7. Lack of experienced software staff

Company with Software experience
Companies that have software experience but view hardware as a "generic" problem, e.g., PC hardware which tends to be very similar compare the unique type of IoT hardware will find the following differences:

1. Unique hardware with sensors and controls which interact with the real world
2. Real-time performance issues
3. More/new security and privacy vulnerabilities
4. Hardware life cycles and upgrade issues
5. Hardware retirement and disposal concerns
6. Many protocols and standards which are fluid
7. Lack of experienced hardware development staff

Company with Hardware-Software Experience
Companies with hardware and software experience may be the best position for IoT, but even they will find some new differences, including:

1. Environments IoT will function in change rapidly
2. Communications to devices and networks (unless they have been working the mobile space)
3. Scales of hardware from the small (coin size) to large (city size) requiring systems thinking
4. More security and privacy issues
5. Resource limitations, such as batteries, memory size, performance, and others

Note: the hardware and software experienced companies will like encounter these difference once they get over the shock of their initial learning.

Company with Systems experience
Companies that have done systems and system engineering have the advantage of thinking about the "big picture, " and often these companies can deal with both the hardware and software, but difference this type of organization can include:

1. Having outsourced hardware or software, rather than doing things in-house
2. Integration, which may be familiar concept, but maybe new in the consumer or industrial aspects of the device where they have not played before
3. Agile may be new since many system companies are "older" so they "follow" waterfall ideas

Government organization
IoT will be in cities, counties, states, people's bodies, transportation systems, and you name it. We can pretty much assume government regulation will continue to expand to cover IoT. Many leading IoT industries, e.g., transportation and medical device, are already highly regulated, and this trend is likely to continue.

We have heard regulators say "well the regulations that we have now, work." We are not sure this is true. Many regulated industries seem very slow in how things happen. IoT will be fast. Many regulations are based on technology that is year or decades old. IoT will change technology very fast. It seems like regulation will take time to be worked while at the same time safety and hazard issues will occur followed by a public demand to "fix" things.

We have worked consulting with a government organization. Some seem to be "progressive" in getting ahead of technologies. Other seem mystified as if they were seeing magic when we talked about development and test technologies which had been in use for years. We are worried in a rush to plug holes in safety, hazards, security, etc. poor and restrictive regulations will do more harm than good.

We hope government readers will think about what these eBooks say, and feel free to contact us. The world is changing. Societies need to keep up.

Companies with Consumer products, but minimal software, electronics, and/or systems experience

 => All products become IoT Consumer t

Figure: Consumer Products Move to IoT or Die Trying (Internet of Everything)
Reference: http://www.ecvinternational.com/consumers/image/shopping-1.jpg

Finally, some companies have long histories of consumer products. Here we think things like clothes, food, shoes, and products you find in a supermarket or any place in big box stores. Many of these companies at first will think IoT does not apply to them, but many will need to think again as IoT products in their business space appear.

For example, one of us submitted an idea to a IoT contest regarding diapers. How can diapers be IoT? Well, read on.

We were challenged by some people who said, we do not see how IoT can be used outside of the tech space. They were holding a small newborn baby.

Well, we what is one of the problems parents deal with in babies?

Answer: Diapers, specifically, when to change them. How is this currently done? 1) baby cries; 2) the smell test; 3) the finger test, with items 2 and 3 being done after 1 happens.

Would it be nice if the parent got a notice before 1 say the baby needs changing?

Can IoT do this? The short answer is yes it can with a wetness sensor and notification to an App.

However, is this idea a viable business space? Are we restricting our thinking? Again the answer is likely yes because when we have done this as an exercise in class with most engineers missing a part of the market space in consumer diaper market.

Can you think of it?

A big market is now adult diaper. In fact, it may be even more critical for adults to know they need changing because many people using these products do not or won't notify that change is needed. These situations lead to things like bed sores and UTIs. These are much more expensive to fix than changing a diaper.

So our guess is many companies will at first ignore IoT since it does not fit their business model. Some of the companies will get surprised by the Newbie level 1 startups. They will then buy the newbies. They will then have problems of learning about hardware and software (see organization sections above).

Worse at the risk of being "skunked" by some other company many traditional consumer companies need to avoid what happened in the watch industry some years back.

Swizz Watch Story: Short version

Years ago a Swizz watch company invented the digital watch. It did not fit the "mental model" of what a watch was or should be. They let the right go to other companies and countries. Before the digital watch, a large percentage of the watch industry was Switz. Now decades later most people use and have digital watches.

Moral of the story: consumer companies with significant market share need to be aware of any new tech even when it does not fit their local mental model. IoT may be one of these techs.

Highly Experienced Company level 3

There are a few, usually big companies, that have all the bases of level 2 covered. You know their names. They have moved and will move into IoT. We have worked with some of these types of companies. We recommend they look at the list of level 2, and ask themselves where they can improve. Our experience is every company can get better. They do this by growing their people, forming relationships with other companies, or acquiring small companies. We expect acquiring companies to go on forever like it always has, and we hope people will continue to grow their skills like some of have. These eBooks offer some help on growth in IoT.

The process, people, and product improvement have many books and reference (TBD list of them?). We recommend people and organizations take some time, review the above lists and create their improvement or career plans. We have always used plans to work on areas for improvement. We continue this today with IoT plans. As we have said, these books are part of our learning and improvement. Feel free to send us suggestions.

Getting started with IoT

In this section, we outline some of the vital items we think may make IoT a success. Indeed there are more keys to being successful with IoT. We define 4 here for this book. As our eBooks mature and we get feedback, we will add more.

IoT Key 1: Ubiquitous user interface needed for IoT devices with constant communications
IoT can learn from Web design. The Google splash page is an excellent example of keep simple it is (keep it simple sir = KISS). There is underlying complexity which can be used, but the primary start point is very minimalistic. Contrast this with some IoT Apps we seen which have more options than can comfortably fit on a smart phone's small screen

First, we need to consider what we mean by the user in IoT. For us, users include:

1. Human interfaces (many different kinds of humans possibly)
2. Communication (comm) interfaces
3. Hardware
4. Security
5. Other pieces of software
6. Other systems
7. Testers

There are two audiences to this section.

First, developer decisions. Include testers early. Do it in a later phase? Standards will help.

Second, tester validation tests.

Give them a list. - tbd

Jon saw a talk by Ken M (Japanese) professor with fame in IoT space.

We should suggest tests for the developers to do so that developers can be sure the device can be configured internationally, age, tech smarts, social factors, diminished capacities.

If the devices are truly smart, then they will accommodate these differences our house will be the gateway. Most folks do not understand routers. Optimally, consumer devices should just plug in, and it works.

On the development side, do not say, "Users are going to configure the device it is easy." Nope, most user can configure even smart device UIs. Smart of us means the device "thinks" of many of the things a user is going to need and want to provide these without copious amounts of user action.

We see stories of the current infotainments system of cars (circa 2016). Many consumer reviews state the UIs are not easy to set up, understand, or find things. They are not "intuitive." Other UIs in cars are a joy to use (or so it is reported).

For example, the router in your house. How easy or hard to configure? Does it use system default passwords, which most people use? These cases may not work for IoT when you have 10s or 100s of devices in your house or usage area.

However, it gets worse as most of you when you hear, "UI," think human. In IoT user don't have to be human as the list above indicates. Our successful IoT UI needs to work with other machines in

the IoT network. It may need to seamlessly switch between different comm networks as their availability comes and goes. Our UI many need to work with sensor and actuators without the human having to worry about calibrations, environments, and other factors that will impact the hardware.

The human user does not want to care about all the things going on under the IoT hood. They just want the device and systems to work.

In one of our home automation attempts with IoT, it took us hours to get a partial configuration of speakers, smartphones, and TV. Moreover, even then the total configuration of the system was a little clunky. One has to push several buttons to get speakers and TV to change source devices. Further, Wi-Fi songs appear to drop out during play mods if two smartphone devices by different manufacture try to control volume. Apparently, the UI needs work, but it is cool when it does play.

Finally, our guess is the IoT UI will likely include voice input/output controls driven by adaptive AI to help the different levels and types of users. Visualization will continue as an option, but for many IoT devices, we will start to interact with them the way we do with humans. This may mean the "smarts" of these systems will need to understand the primary and nuanced aspect of human communication such as tone of voice, accents, body language, etc. These will take time to perfect, but with systems like Siri and Alexa, we see the beginning of good voice control UI.

IoT Key 2: Learning from Data Analytics to Drive IoT Dev and Test

Key Point: IoT will be all about the data (base)

I have been talking about how data and big data will be generated from IoT in massive amounts. So far, the literature and interest seem to be from marketing people and business managers who see the data as a key to generating more sales and money. Many testers and dev people see less than interested in the data analytics message.

In general data, analytics is a growth area in high tech (see tech curve in first eBook section). Many high tech companies, e.g., IBM, Microsoft, Google, and others, are investing in data analytics. So, it curious that many dev/test professionals seem less than interested.

Readers should Google or see the other eBook in this series for more information on data analytics

My recommendation is that testers should be data consumers. I have used data to improve my testing for years. I have data mind maps of errors. I have used real-world embedded device telemetry to verify my test plans and strategies. I have seen test labs which generated terabytes and then petabytes of data which testers and analysts had to leverage.

To become data consumers, testers will need new skills and knowledge. Not every tester will need to be a statistician, but we will need to learn how to use analytic tools and ask questions of the tool operators. This may change testers from being just "runners of tests" to software engineers. Some testers will make the transition to engineering during IoT, and some will remain tester. You need to decide for yourself. The history of computers in part is the evolution of data usage-analytics.

The nature of both computers and the data they generate has changed over time. In the early days, 1950-1970, computer-generated limited data that was used by a few "nerds" and programmers. It was almost priesthood. General human use of computer-generated data was limited and supplied by these priests.

Computer evolved into the "personal computer" (PC). With the PC more people had access to data. While being more available, data was often limited to what was on that PC and that set of users. The internet and web in the 1990 changed all this. Quickly PC users were able to pull and search large amounts of data, but the data was often general, and much work was needed to "pull" the information of interest. Web pages started helping this, but at the same time the number of websites and information expanded, and so we would get "Google" hits of 100s of millions of web references. Again, not what many users wanted.

At the same time of the computer, the PCs, and the web, there was another domain of industry that was computerized. This domain environment gets called embedded software systems. These started to appear in the 70s, expanded into the 80s and 90s. These computers were typically not connected to the internet, had limited user interfaces and hence data flows to users and had many resource limitations. These devices started to run our machines, health, factories, industries, and cities. During the 2000s and until today, these embedded devices slowly started to get networked with machine to machine (M2M) communication. In many cases, the user interface expanded. We even saw the first virus (Stuxnet) infect them (side note: the researcher that found the virus did not at first understand the type of computer it was targeting). The data inflow and outflow were often limited (or none) on such embedded device, but today this has changed as we move embedded into IoT.

IoT is projected to generate vast amounts of data from what was once embedded devices, plus all the new IoT devices that will appear. Petabytes and beyond are projected. This data will be generated during operational use of these IoT device systems. The data will be used by many interested parties. One of those interested parties should be the testing staff. Testers should become involved in the operational use of IoT devices and the collection with analytics of the operation data. Benefits to testers will include:

1. Improved field based error taxonomies reported automatically by devices
2. Tasking some test activities to the user in the operational setting
3. Mining of help desk for data to aid testing, e.g., use cases, problem reports, etc.
4. Analyzing in real time Fast data
5. Mining social media for IoT device problems
6. Using data to improve patterns of attack and test models

To use such huge amounts of data, analytics and big data mining will be needed.

One area compared to embedded and even traditional IT systems will be the amount of data IoT can and will generate. Most teams think of data from a marketing viewpoint. A few may think about

using data for prediction. We hope testers and developers will think of data analytics as aids to doing better jobs:

1. Test Taxonomies
2. AI and self-organizing data analytics predefined for tester use
3. Users as tester (generated data as part of Dev-ops)
4. Causes of error and prevention of future errors (process improvement and root cause analysis)

We expect more from tester data analytics.

IoT Key 3: Unique and Specialized Hardware (working to be a system)
As mentioned above, for us a difference in IoT is the unique and specialized hardware. Whether it is the components of a car, sensor, and controllers in a house, or the elements that make a smart "city," we will see many everyday objects become smart.

Those of testers of IT and computer software system pretty much dealt with generic hardware. Sure, there were differences in computer platforms, peripherals, and communications, but IoT will be orders of magnitude different in these sensors, controllers, and attachments.

A tester who worked with embedded devices will be more familiar with the unique and specialized hardware. However, embedded teams that move into IoT will get all the problems of embedded plus all the problems of IT systems and higher levels of integration complexity that most of us have seen before.

Many of us have seen large system and systems of systems, pass tests labs, only to fail in the field. I expect the uniqueness of hardware and the system to grow complexity and hence the likelihood of failures.

Many have said will live and work around such failures, but here the idea of "good enough" (see book 3) will come into play. The tester will need to balance good enough and failure for software, unique hardware, and systems.

IoT Key 4: V&V/Test for IoT
I think there will be a struggle on what is the right amount of V&V/Testing for IoT. The other books of this series will explore this key more. There is no best or right level of V&V/testing.

Thinking testers will be needed. Standards may be a start for some teams, but thinking will still be needed. Experienced testers who already can think will evolve quickly for IoT teams. Some tester who have minimal critical thinking skill may struggle.

Context will continue to be king for software, testing, and IoT systems. Management will always want to reduce testing. Some testers will be viewed as "second class" team members.

I always view my job as education for management and other stakeholders. I also never felt like a second-class player. In fact, as I moved around I found most disciplines felt they were "undervalued."

My interest for testing as intellectual challenge meant that pay and recognition were less important to me than doing the best job I could, learning about testing, and communicating this to the other stakeholders. For IoT, the key of V&V/testing will be such communication.

Summary

This is the first in a series of eBooks on IoT V&V/testing. We summarize some areas such as development and operations because what makes IoT different from some other software systems, is the need to integrate hardware, software, systems, communication, and operations.

The IoT industry is growing and evolving rapidly. We plan the same for our eBook. We do seek comments and feedback about what is working in these books and what needs to evolve. We seek Agile customer readers. The other eBooks are (will be):

Part 2 – Dev-Ops

Part 3 – Test Planning

Part 4 – Test techniques, tours, and attacks

Part 5 – Test environments and tools

We offer this first of the eBooks in the IoT series free of charge. Our goal is to keep the book's low cost and segmented to help readers. Our expectation is reader will use smart devices to access these eBooks using the parts they need but not reading the whole book. Given the nature of IoT and smart devices the access of such references is evolving, so tell us what you like and what you want.

We hope you have fun reading and learning. We try to make the access to the information short and easy to help learning. However, learning is only the first step. To build a skill, you must practice the ideas in these eBooks. We expect each idea will be evolved and customized for the local context of your IoT project. Again, please feel free to share what you learn and change. We will update this into future eBook revisions and give citations to you where proper.

We all learn from each other. We are still learning and building skill for IoT devices. We all must remain Agile and context-driven to handle the billions of IoT devices which will range from the simple (minimal testing) to the whole world (more complicated testing)

Please have fun

Appendix A: References for Additional Learning

The following books and references were used in preparing this eBook and/or are of good general reading. Further, for IoT software systems, we recommend a familiarity with many works, but certainly, these are not the only good references. Finally, many people will believe that the mixing of diverse viewpoints, e.g., process standards such as ISO and work by people such as James Bach, should not be done and is ill-advised. However, we treasure diversity and open thinking whatever school or viewpoint one might have. Our industry is young, and we are all still learning. To close out any set of ideas from any book, reference or standard may be limiting, when we do not need limits. You should be free to think, please.

- Software Test Attacks to Break Mobile and Embedded Devices
- Software Test Domain workbook
- Agile Testing: A Practical Guide for Testers and Agile Teams
- A Practical Guide to Testing Wireless Smartphone Applications
- A Practitioner's Guide to Software Test Design
- Black Box Testing. Techniques for Functional Testing of Software and Systems
- Embedded Systems and Software Validation
- Experiences in Test Automation
- How to Break Software series – 3 books by James Whittaker
- Lessons Learned in Software Testing
- The Art of Software Testing
- Testing Embedded Software
- Testing Computer Software
- Testing Safety-Related Software
- Safeware: System Safety and Computers
- Systematic Software Testing
- Testing Complex and Embedded Systems

- Works and website by James Bach
- ISTQB syllabus (download for the web)
- ISO/IEC/IEEE 29119 software test standard
- IEEE 1012 Verification and Validation standard

Risk References

[1] http://en.wikipedia.org/wiki/Risk_analysis_(engineering)

[2] http://www.hcra.harvard.edu/

[3] http://sevocab.wikispaces.com/

[5] http://en.wikipedia.org/wiki/Failure_mode_and_effects_analysis

[6] http://en.wikipedia.org/wiki/Brainstorming

[7] Myers, Glenford, 1979, The Art of Software Testing, Wiley

[8] Leveson, Nancy 1995, Safeware: System Safety and Computers, Addison Wesley

[9] http://en.wikipedia.org/wiki/Failure_mode,_effects,_and_criticality_analysis

Glossary of Definitions

In this book, we 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
- Definitions in “Software Test Attacks to Break Mobile and Embedded Devices” by Jon Duncan Hagar

If you do not find a term in this list, refer to one of the sources listed above, 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.

Disclaimer: It should be noted that in general the software industry and software testing often uses terms slightly differently or uses different words that mean the same thing as another term. We do not seek to solve this problem, but we do acknowledge it. We provide a set of definitions for our

series of eBooks that help readers know how we are using a word locally within the eBook and not that our usage is universally right, though we do follow many industry common definitions as much as we can.

Definition, Reference Links, and Acronyms

A2D - Analog to digital

D2A - - Digital to Analog

Checking - The basic activities of confirming that software meets its requirements (see verification).

Chaos Engineering and Testing - http://principlesofchaos.org/

COTS - Commercial off the shelf (can be hardware or software); throughout this book I have used "off-the-shelf."

Exploratory testing - Software testing which simultaneously learns, designs tests and executes them, see http://en.wikipedia.org/wiki/Exploratory_test

Failure - Termination of the ability of a product to perform a required function or its inability to perform within previously specified limits.

Fault - When an error in software manifests itself.

Field testing - Full–system test done at an operational site or in the real world.

Interrupts - Hardware–based signal generated to the software for action http://en.wikipedia.org/wiki/Interrupt

IV&V - Independent Verification and Validation (see below)

Model - A representation of a real-world process, device, software or concept, which can be logical, physical, and/or mental.

Noise - In the physics world and analog electronics, noise is mostly an unwanted random addition to a signal picked up by sensors or electronics, which can impact software processing

Tours - A logically ordered sequence of test activities. For example, stories, techniques, or attacks, which are centered around a theme or concept. For example, a world tour, an error tour, or a hacking tour.

Modeling (Models) – use of a language or math to define a representation of aspects of hardware, software, and/or a system. Remember: all models are wrong in some way, but many models are useful

Sand Box - test environment of used in security - privacy testing which is "cut off" from the real world so damage cannot be done if a virus is used in a test environment. It is a place to "play" with testing.

Quality – 1) Value that someone is willing to pay for. 2) attributes of a product, e.g., performance, reliability, safety, security, etc.

V&V – see below

V&V/Test Approaches Defined:

Analysis – use of math, modeling, and human thought (see dictionary too) to provide information about a product. The analysis is often done before actual products existing to assess artifacts such as requirement, model elements, design, risks and plans for completeness and correctness.

There are many subforms of analysis that teams should be aware of:

Formal analysis (verification)

-Quality control (QC) of production lines in manufacturing

-Math modeling

-Simulation

Demonstration – is testing but done on the deliverable product(s) pretty much as they will be used by consumers. Demonstration is often used where products used in testing have undergone some alteration to support testing which may impact test results. For example, special hardware or software "instrumentation" can impact test results such as timing performance or even outputs. Demonstration can be used on a complex system to confirm the results of earlier testing. Demonstration is not necessary for every product, but if planning and risk analysis indicate test results may be questionable, demonstration is used. System using demonstration include complex IoT systems, smart cities, automotive systems, and medical systems, where each of these come with a higher level of risk and criticality.

Inspection – is the visual or human assessment of a product. Inspection is often used for hardware, for example looking for workmanship defect during receiving or before a final delivery (think kicking the tires of your new car to see if they fall off). Some assessment can only be done by humans, but humans have problems of bias and more substantial variation between humans doing assessments. Use of inspection should be limited to those areas where other concepts can NOT be employed.

Test – See test above