IoT Dev and Testing: Part 3

IoT Test Planning and Strategy: A Guide for Test Leadership

Jon Hagar

Tweet on this book, please

LeanPub

Note: this eBook series is produced following an Agile production process. Early drafts will be less than 100% complete, which means there will be missing sections, to be determined (TBD), and Editorial issues outstanding. I welcome input and comments to:

jon.d.hagar@gmail.com or on LeanPub

Notes on current draft:

Current draft is 90% complete but has not had a final English edit nor independent team review. There are numerous “to be determined” portions noted as TBD, which will be completed in later drafts. Also, some sections may be expanded based on reader feedback (please). Finally, a bug in the table of contents was fixed.

Preface

The purpose of this eBook is to help companies and testers entering the environment of Internet of Things (IoT) testing and evaluation. It is meant as starting guide to help readers understand the most basic aspects of test planning and strategy for this domain.

This eBook contains maps to information excerpted from “Software Test Attacks to Break Mobile and Embedded Devices” by Jon Duncan Hagar, which can be applied to IoT. Other important test references are included since this book is not intended to be a comprehensive guide to planning, but as a starting point for IoT.

I hope this book helps testers to discover the concepts and approaches needed to assess the qualities of software which drive IoT and the overall “goodness” of system devices. However, it provides only introductory knowledge; a tester must still do additional reading and practice the concepts in order to build required skills for this and other domains. The book is intentionally short for ease of use and reference.

Introduction – Defining IoT Test Planning and Strategy

The world of high tech constantly moves from one hot topic to another; from large scale computers, then personal computers, then the web/.com systems, and more recently mobile/smart devices. Each of these represents an expansion of software and system functionality along with the growing pains of errors and even company failures. However, for every Microsoft or Google, there are hundreds of companies that did not succeed in each of these market "explosions." This eBook considers testing from a planning and strategy viewpoint for IoT devices, which are currently undergoing a market explosion.

For teams starting their work, it is optimal to have some kind of test plan and strategy which can range from "we will do no testing outside of what developers and our first users do," to "we will do full testing including independent V&V (IV&V)". The choices within such a range should be determined by factors such as: risk, financial impacts, schedule resources, reputation, team skills, product maturity, basic quality, and others. There is no one "right" or best technology and certainly no one or "best" answer for all challenges. Organizations must decide during the proposal, budgeting, and planning efforts how much and what kind of V&V or testing is "right" for their context. Further, remember that what is "right" is likely to change over time as the IoT product evolves and matures.

At the beginning of the technology maturity curve, the next hot topic includes IoT. Readers should understand the maturity of IoT technologies and devices. IoT is rather immature in many areas and many companies are actively engaged in IoT development and deployment. There is likely to be much volatility and change in IoT, as well as a maturation of each major IoT market segment.

See Tech Growth Curve at – https://media.licdn.com/mpr/mpr/p/7/005/081/13a/317cb86.jpg

It is my belief that test planning should be part of all project management efforts. It has been observed that a Plan (a document or piece of paper) is not as important as the planning effort. What this should mean for everyone is that in most successful human efforts a plan, i.e., some kind of road map that takes us from where teams are, to where teams want to be is needed. Just aimlessly wandering around in the IoT wilderness is likely to be inefficient and ineffective and can consequently sacrifice company viability and profits. So, to improve the chance of IoT success, it is good to do the planning exercise and put forth a plan. As a word of caution, too often "the plan" becomes some kind of "truth" where people become afraid to change it and blindly follow it--even when it does not make sense or ends up wasting resources. A plan should be more like the "Pirate's Code" (see the movie "Pirates of the Caribbean), where the document is more of a "guideline" that can be tailored and changed as needs and context dictates.

What this book addresses

This book provides a starting point of information on IoT assessment including:

1. What is IoT test planning?
2. What is IoT verification and validation (V&V)?
3. How to use V&V/testing to provide information about IoT devices to stakeholders.

4. How to do test planning (important) that results in test plan documents (less important).

5. How to establish test strategies as part of test planning.

6. How to set up IoT test environments (labs or sandboxes).

The book is intended to be used with other books on planning, budgeting, and strategy in testing. I recommend that the reader read or have on hand several of the books found in the reference section because no single book can address all aspects of test planning for IoT. My own reference libraries are large and historic, having hundreds of works, which I refer to as needed.

Audience

The audience for this third book in the IoT Dev-Testing series includes:

1. Corporate level officers and managers who require an introduction to test planning. Although they should also read the short eBook "IoT for Exec's"

2. Development managers who may want to or need to be involved in test planning, particularly as it impacts development staff, project manpower efforts, and planning.

3. Proposal teams and leads.

4. Test managers and leads who do the planning.

5. Testers and support staff who want to understand planning and concepts to further their career.

There is no universal agreement on test concepts with names such as approach, strategy, techniques, and planning. These ideas will be defined for this eBook and attempts are made to try to agree with standards and other books of knowledge, as much as possible. However, readers should keep in mind how terms are used here and that these terms may not match your view.

How to use this book

Like the other books in this series, my intention is that they be used as initial references and not read cover to cover. I recommend specific topics be skimmed in the table of contents and then jumped to within the book. My hope is that some of the figures, tables, and pictures communicate ideas quickly without the need for lots of reading. Keep in mind that this book is not meant as a comprehensive guide book and other reference works must be used in conjunction with it.

I request that if a reader finds things missing or lacking that you contact us, as I plan revisions to the eBook rapidly following Agile concepts. I plan on-going updates and changes, as the IoT industry evolves.

Test Planning Concepts

Planning is something most humans do and many do it pretty well. We build and create things to a plan to, hopefully, make our lives easier. As stated earlier, the plan document is not as important as the

action of planning in order to organize efforts for the success and quality of an IoT device. So, I begin the planning at a high level in this part of the eBook and later drop down to lower levels. In the case of IoT testing, I hope that by doing the planning exercise you will help to get a device that works, has the right kinds of qualities, and is created within a reasonable cost-schedule period.

Figure: Planning

Ref: https://previews.123rf.com/images/stuartphoto/stuartphoto1205/stuartphoto120500882/13564521-Planning-Definition-Magnifier-Shows-Organizing-Strategy-And-Scheme-Stock-Photo.jpg

Quality is value that someone is willing to pay for. There are many classifications of quality (e.g., safety, functionality, performance, reliability, etc.). In IoT, like every other product humans buy and sell in the world, there are many qualities and levels of quality. Testers and development staff will need guidance from executives and managers to decide on the types and levels of qualities for the context they are working in and around. Every IoT device will be different.

Since IoT devices range from consumer "fun" gaming toys, to home systems, to medical devices, to industrial systems, to large scale systems-of-systems, the qualities and levels will also have a wide range. Gamers will want the device to be fun. Mothers at home will want their home IoT system to help them in some way. Doctors will want medical devices to keep people alive or provide data so the doctor can help the patient(s). Presidents of companies will want their IoT infrastructure to save money. And, the general public will want a smart city to be really "smart" while not crashing under attacks or stress. Given this wider range, no eBook, single plan, or strategy will be "best." Much thinking will be required by smart teams as well as testers for IoT devices to be "good enough."

Good Enough IoT Software and Devices

The developing organizations will want the IoT device to be just "good enough." Being "good enough" can be defined as making a "product that will make our company money, have a positive ROI and keep us viable," but nature of these factors can change over time. For example, the first web pages were raw, but worked well enough for some brick and mortar companies to grow, but now web pages are expected to be "great" and "easy to use on any device." What was once "good enough" to a user of PC based web pages has changed over time. IoT devices and systems will go through the same growing pains. I see them every day and "good enough" is a constant balancing act.

Making money means that the stakeholder (user, customer, other) gets a Return on Investment (ROI) with their investment in IoT. ROI for testing considers two factors: the cost of testing vs. the cost of not testing.

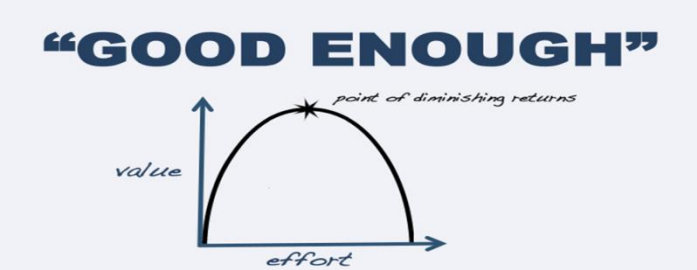

Figure: Good Enough Point Between Value vs Effort (cost and schedule) – A balancing Act
Ref: http://www.pellegrino-riccardi.com/wordpress/wp-content/uploads/2016/11/Good-Enough.png

The cost of testing has a wide range--from almost zero (no testing) to as much as the development costs or more, and such costs change over time, with product usage, and with many other factors. Even teams who say "we do no testing" may mean we do not have a dedicated test team, but developers still have to review, compile, integrate, and run their software at some point to assess it as it goes into a system and is released. This is considered ad hoc developer testing and can consume a third of their time plus or minus some percentage. Better developers do more testing which, in part, is why they are "better" at their jobs. Organizations where the cost of testing is more than the cost of development are few, but these are groups doing software for very high risk systems (e.g., a control system for nuclear devices which needs a lot of V&V/testing or lives are put at risk).

The cost of not testing can be real too. Consider if you must replace defective IoT devices because they were not "good enough." Automobile companies have had to recall hundreds of millions of cars due to faulty software. How much did that cost—not just in dollars but in loss of credibility over how many years, and how does a company regain that credibility.

See Samsung - http://www.samsung.com/us/note7recall/?)

Next, consider the cost of bad press or loss of investors on a prototype IoT device that fails publicly (see VW stock price fall

See http://money.cnn.com/2015/09/24/investing/volkswagen-vw-emissions-scandal-stock/

Finally, consider the cost of going out of business. The high tech world has far more companies that failed than succeeded because often their product was "just not good enough."

Now, most teams want a simple formula for how much V&V/testing vs. development will get them "just good enough." Note: the reader can refer to eBook part 2 for more information on estimation.

If high tech was simple, everyone would be creating products and software while succeeding at every opportunity. The high-tech world can make a lot of money and has demand by many customers

because technology is not simple and because people want technology that works with a return on their individual investment of time, money or effort.

The “good enough” concept of software was coined by James Bach and has some Agile variations as stated in ref http://www.satisfice.com/articles/gooden2.pdf. This article is recommended reading. The concept is important since most organizations use it in some form, though they don’t always call it by this name. For me, “good enough” in test planning has the following characteristics:

1. Functions – the important and critical functions that a user or customer wants to work

2. Works – the time between failures of functions is long enough such that the user or customer gets some value

3. Non-functional qualities – the valued quality items a user or customer expects or pays for

Basically, “good enough” means the user or customer does not put it in the trash or ask for their money back. Users may complain, but if you ask if they want you to take the product away, they will refuse. A “good enough” product provides sufficient value, and the equation of value may change over time. Did you know that the first smart phones were heavy, dropped calls, had limited coverage areas, bad voice tones, and other “features” that today no one tolerates? However, they were once “good enough” so that now everyone has them even though what is “good enough” in them has changed. Now I expect them to be fast, on all the time, meet user expectations, and still be cost effective.

Test Planning Basics

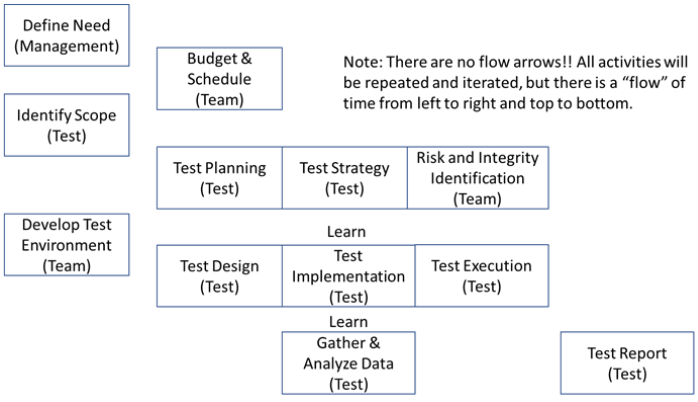

Figure: Test Concept a flow chart
Ref: Jon Hagar Class Production

As the figure depicts, teams should start test planning by defining what is “good enough” for their specific product. Management will determine an acceptance level of product or project risk, which in

turn helps to define levels of test, a schedule and a resulting budget. This balance of cost and schedule to risks helps set a "return on investment" (ROI). Low ROI on testing means less cost and schedule and vice versa. I will expand on this picture in the eBook and the others in this series.

From this point, management or perhaps a test manager will determine tasking, such as what to test, where to test, who will test, how to test, and when testing is done. In a cyclic-iterative effort this information can determine the selection of strategy, which in turn refines the planning items. As the interactions continue, some form of documentation should be produced to aid in information retention and to ensure everyone is operating to defined tasks and goals.

When I say documentation, I mean planning games (see reference list for Agile Software Testing books by Crispin and Gregory), something jotted on a white board that I can take a snapshot of, perhaps some mind maps, or maybe a formal plan is written that includes all of these items (e.g., ISO29119 parts 2 and 3). Again, there is no "best" or "right" way to document test plans. Context factors such as cost, schedule, regulations, organization practices and other factors will determine the types and levels of documentation. In my test planning efforts, I have run the full gamut defined above.

A practiced and experienced tester or test manager should be able to determine the nature of documentation and formality of the document(s) fairly quickly. Teams should not look to a "one size fits all" answer in documentation. And, just a quick note: if you are spending all your time planning, you are missing the point of planning.

Further, like the product, test planning will reflect the nature of the organization that is producing it (see the Conway law at https://en.wikipedia.org/wiki/Conway's_law). People doing test planning should understand their organization, communication channels, and know what the team believes is important. Dogmatically demanding that testing "conform" to an ideal or standard when the organization does not support, it will likely result in failure of test planning.

I will say that in every type of test planning and resulting documentation, one should expect the unexpected. I have seen "known unknowns" in planning (things I know that I don't have answers to); these can be expressed as risks. However, you will also have the "unknown unknowns." These are the things that will cost you when they happen and yet, you may have no idea or knowledge of or about them. (The expression of "hindsight is 20/20" comes to mind.) In test planning, I recommend: agility, having a contingency option, and some resource reserves to handle the unknown unknowns. Further, as soon as they become known, a team should conduct another (smaller) planning cycle. Agile teams tend to do this daily (https://www.mountaingoatsoftware.com/agile/scrum/meetings/daily-scrum stand up meetings).

In test planning, it is a good idea to have different levels of planning. In my experience this will often be:

1. Master test plan which has the big picture and changes less. This plan organizes thinking and addresses the whole project. This may be produced only once, though likely it will have updates as time and life cycles pass.

2. Detailed test plan(s) which may be a series of plans addressing some specific phase or strategy of testing. These would be subject to more changes than master plans. Examples would include testing plans for: a sprint, developer testing, integration efforts, a particular release to the public, a test cycle, daily stand-up plan, etc.

Again, the formality of such plans can vary depending on project context. There is no single "right" or "best" way in test plans.

General Test planning outlines by organization classification

In this section, I organize testing around general types of teams. This is learning by example. Of course, there are variations and options within teams doing actual planning since there is no "best" plan for testing. I present these organization-based plan concepts as a quick start set of examples based on common contexts I have seen.

At this point, the reader should refer to the definition section for terms such as verification, validation, testing, checking, assessment, inspections, etc. which are used throughout the test plan outlines. These outlines are not intended to be a table of contents but a list of important test ideas that should be considered. And, while the outlines look similar, they demonstrate how the team's work will be different because of context and team's goals. Further, remember that these outlines should be viewed more as examples to be used as a starting point.

Pure start up, single device, and/or small teams who are trying to stay alive

Some IoT devices will be done by start-up groups where there may only be a few people (perhaps under 10). Such teams will likely be practicing Agile in development and testing.

Testing will likely be done by development and be "just enough" to assess if the device "works" or they will have a single person skilled and focused on testing within an Agile group. This team has low expectations of "good enough" and is trying to "stay alive" with only the most important features working.

Test Plan outline example for Start-up, Single Device, Small Team:

1. Risk assessment exercise (what scares us or should)

2. Our hardware or other's hardware checkout

3. Any third-party software checkout

4. Developer structural testing test driven development (See eBook 4 or Agile Software Testing in reference list)

5. Integration and integrated test (make it work)

6. Small series of system checks against stories/requirements (Acceptance Test Driven Development (ATDD)) and risks, performed in maybe less than a day of effort

7. Deliver (consider if the "customer/user" will be part of a dev-ops test team)

This will be the first or an early development cycle. The device needs to be "good enough" to make it to the next cycle. Testing is minimal and just "sanity checks." Nothing will be too formal. There is minimal or no documentation at this point. Exploratory testing concepts (please refer to my first ebook part 1 on IoT section 4) should be in play.

Mature groups or growing teams with targeting sales

So, the organization has made it through early efforts or has some history, with support of development. The expectation is to actually deliver a device to marketing/sales and/or users. The team may have had several Agile cycles behind it, where increments have been delivered to someone. "Good enough" here has a higher bar than earlier efforts. The risks are increasing now. The device needs work to the point where buyers will continue to buy it and some good press (social media or tech reviewers) may be generated. The organization may be part of a larger company or smaller organization but the bottom line is the device must be "good enough" (e.g., better than first cycles or prototypes) so that people will definitely continue to want it.

I expect the IoT team has some hardware, software, and system skills. While the degrees of skill in each area may vary, the team has the ability to understand where it is strong and weak. Test planning must address weaknesses and leverage strengths during risk-based testing and strategy selection. The test planning can still be "small," but the idea of no testing should be covered in discussions with management and customers so that testing doesn't become a programmatic risk factor.

Test Plan outline example for Maturing Group, Single (or a few) Devices, Growing Team:

1. Risk assessment exercise and assignments to an integrity level (see IEEE 1012 on IEEE web site)

2. Verify the hardware

3. Check out any third-party software

4. Developer structural testing (TDD at higher coverage levels)

5. Integration and integrated test (make it work)

6. Verify hardware and software

7. Validate stories/requirements and device

8. Small scale system V&V

9. Deliver (consider if the user or customer will be part of a dev-ops test team)

Optional strategies to consider for inclusion in a test plan:

1. Expanded experience-based Exploratory cycle, driven by risks and attacks
2. Math-based testing
3. Model-based testing based on development models (if any models exist)
4. V&V of hardware, commercial third party components, and system evaluation

Teams need to consider what "good enough" means to them in this device context. This can vary from a start-up to an existing large company. Questions to ask and answers for "good enough" include:

1. How much failure can be tolerated?
2. What costs can be dealt with and how?
3. How does schedule (time to market) play into the product, risk tolerance, and testing?
4. Who does the testing?
5. Who are my users and customers (stakeholders)?
6. Where do I do the testing? (Is an external lab or third party testing required?)
7. When is testing done?
8. What are my strengths and weaknesses? (What do I have that I can positively leverage?)
9. Can I grow this product and improve sales?

Mature group with one to more devices or historic teams want to grow market

This is where many of us hope to be or become. This is a group that has products in the market, is doing maintenance, has sales, and may want to expand with new or upgraded IoT products.

The range of IoT devices will cover consumer products, industrial products, government products, and everything in between. Hence the risks and integrity levels (see IEEE 1012) will be vast and varied.

Test plans will need to cover the organization's interests including:

1. Items listed above in the last outline section
2. Does this product have increasing risks or integrity levels?
3. Is this product core business of the company (i.e., if the product fails, you are out of business)?
4. If the product is in maintenance, are the changes small or large (new test, smoke test, and regression testing impacts)?

5. If this is a new product, how can or will this impact the existing product lines?
6. What am I learning from user data (analytics) and competition?
7. What V&V/test improvement areas should I be expanding into as “good enough” changes?

Test Plan outline example for Mature Group, with one to many Devices in Dev, by Historic Teams:

1. Risk and integrity level assessment
2. Verify the hardware
3. V&V any third-party software and hardware
4. Development structural testing (full TDD and analysis)
5. Integration and integrated test (make it work with other device(s) and system(s))
6. Verify system
7. Test and verify software
8. Validate story/requirements and
9. Validate device via simulation, test environment and field testing
10. Correct level of system V&V
11. Product quality V&V
12. Maintenance and regression V&V
13. Deliver product to stakeholder or customers
14. Ops and test with data analytics

Options to consider

1. Expanded Exploratory cycle driven by risks and attacks
2. Model-based testing based on dev models (if any exist)
3. Higher levels of test automation
4. Constant product and process improvement
5. Optimized regression and new feature testing
6. Team skill building, training, new and experienced team grooming

7. Formal data analytics and taxonomies

Teams in this category need to avoid “slipping” backwards and losing market share (refer to Deming at https://en.wikipedia.org/wiki/W._Edwards_Deming). Groups and companies—once successful, can relax. This opens the door to competitors taking market share. Relaxing can be a bad thing.

Organizations in this category often learn that test/V&V are important as these efforts provide the data needed to continue growth. This is not to say checking and mechanical testing cannot be reduced or otherwise optimized, but such changes must be within process improvement and planned.

For a discussion pesticide paradox and problems in automation/regression testing (see Planning Wrap up: regression testing section below)

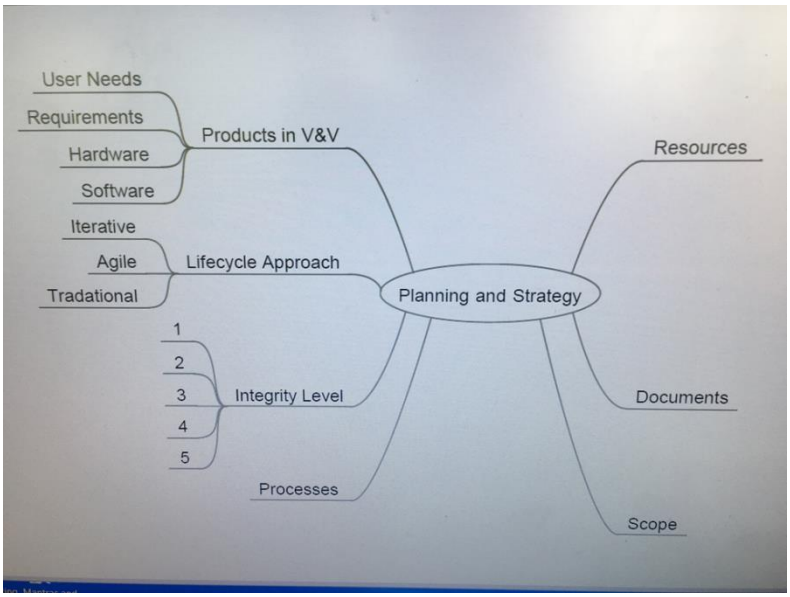

Figure: Test plans element mind map

This figure illustrates some common elements in a mind map form that a tester may want to include in a plan. It is presented as a checklist (i.e., not every element needs to be addressed) but teams may want to consider them during their important test planning activity.

Test planning for procuring organizations: Government, large corporation, and others

(when failure is not an option if it is ever)

Some readers may be wondering why this section on test planning for procuring organizations (such as governments or consuming organizations) as a separate test planning category is necessary. Because much of IoT will happen around names like smart cities, smart schools, smart factories, etc. The procuring organization will be buying and using IoT and all the systems that surround IoT. Therefore,

procurement organizations must have some knowledge before approving spending or spending important dollars on IoT technologies or devices.

I advise that the world needs to develop “smart” IoT consumers. One of the biggest (THE biggest) will be governments. However, companies will be a close second, and even individual consumers may want to be “testing” IoT. Most large organizations, such as governments and companies, already have Information Technology (IT) departments. Many IT groups for years have, as part the procurement and deployment efforts, had test planning efforts. I have seen companies with IT test labs devoted to assessing incoming hardware and software, before it is deployed to the staff. This kind of testing is different from developer testing, but follows many of the same ideas of planning and strategy.

Companies and governments have learned in establishing and running “IT” departments that testing/V&V is part of what these organizations must do in order to be successful. Most of us (as individuals) conduct testing too, for our major procurements. For example, would you buy a car or a house without testing and evaluating it first? IoT should be considered in a similar vein.

What this means for IoT is a new procurement organizational paradigm must emerge. Procuring IoT organizations will need planning and testing just as they do for IT, but the efforts will have to expand to include hardware, systems, and operations as well as the software.

Many procuring organizations may think that all they need to do is expand their IT department--and this can be a good start, but what if they expect “smart?” Then, they will need more than just the traditional IT department skill set. Possible new skills which may be needed include:

1. Operations understanding

2. Data analytics staff

3. Specialized hardware practices (not just for computers but perhaps for telecommunications or radio frequency analysis too)

4. Human factors and IoT usage experts

5. Psychology

6. Expanded and new software knowledge

7. Testing teams with IoT understanding

Test Plan outline for Procuring Organizations: Governments, Companies and Others can include:

1. Risk assessment and integrity levels exercises

2. Verify the hardware within the new “smart” system-of-systems

3. V&V integrated software elements especially long duration runs (see Software Test Attacks to Break Mobile and Embedded Devices)

4. V&V of any needed software wrappers (containers which isolate a software element) locally created as part of Agile

5. Integration and integrated test (make it work in system-of-systems)

6. Verify system and system-of-systems

a. Test and verify software

b. Validate requirements and device

c. Correctly scale system V&V

7. Product quality V&V

8. Maintenance and regression V&V

9. Ops and test with data analytics

Options to consider for Procuring Organizations: Governments, Companies and Others

1. Expanded exploratory cycle driven by risks and attacks

2. Model and simulation based testing based on development models (if any exist)

3. Higher levels of test automation

4. Constant product improvement

5. Team skill building, training, new and experienced team grooming

Note: The above may seem onerous for the individual consumer, and likely most consumers will have ad hoc testing plans--like the ones they use for buying cars or houses. However, I feel that some consumers may want to follow some of the ideas of these eBooks (a future guide may be created) since IoT at the consumer level will be more complex than current home “automation” systems, may be expensive and time consuming.

Possible impact of data and analytics to IoT test planning

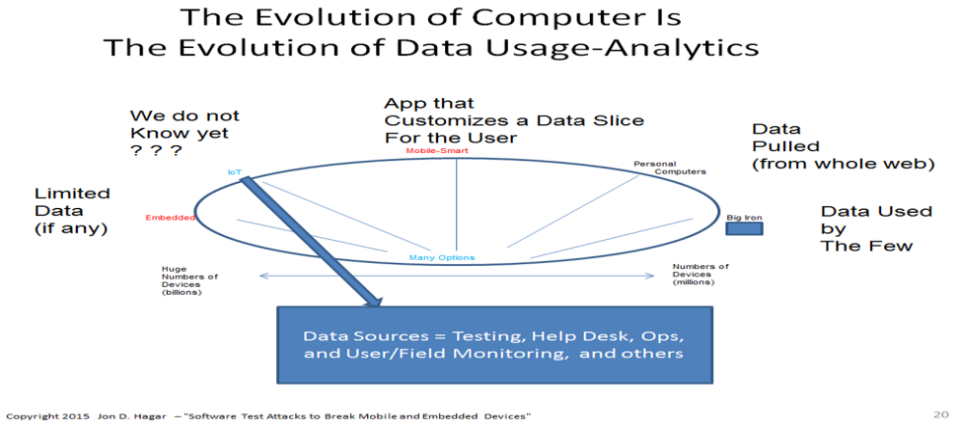

Figure: Data evolves into IOT
Figure: Jon Hagar Class Productions

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

Computers 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's changed all this, which is when PC users were able to pull and search large amounts of data. However, the data was often general and much work was needed to "pull" information of interest from any compilation. Web pages started helping with this while at the same time the number of web sites and information on them expanded, and so the product would get hundreds of millions of "hits" (web references), some of which did not pertain directly to what is being searching for. Again, not what many users really wanted. And this trend continues to this day, which means it takes manpower to review and extrapolate data to find sometimes the simplest of things in the data analysis.

Around the same time as the PCs and the web, another domain of computerized industry was emerging, called embedded software systems (which came out of the rocket industry). These started to appear in the 1960s-70s, expanded into the 1980s and 1990s. These computers were typically not connected to the internet, had limited user interfaces (hence no data flows to users), and they had many resource limitations. These devices started to run our machines, health, factories, industries, and cities. During the 2000s and today, these embedded devices slowly started to get networked with machine-to-machine (M2M) communication (including mobile-to-mobile). In many cases, the user interface expanded. The world even saw the first virus (Stuxnet) infect them (Note: the researcher that found the Stuxnet virus did not--at first--understand the type of computer it was targeting). The data inflow and outflow was often limited (or none) on such embedded devices, but today this has changed as embedded systems have moved into the IoT domain.

IoT is projected to generate huge amounts of data from what was once embedded devices, plus all the new IoT devices that will be invented and marketed. Petabytes and beyond of data are projected. This data will be generated during operational use of these IoT device systems. The data will be used by many interested parties. The data users may think of first is system marketing or controls, yet one of the interested parties should be the testing staff. Testers should become involved in the operational use of IoT devices and the collection of analytic data from the operations data. Benefits to test planning can include:

1. Improved field test planning based on error taxonomies reported automatically by devices
2. Tasking some test activities to the user in a more realistic operational setting
3. Mining of help desk data to aide test planning (e.g., use cases, problem reports, etc.)
4. Analyzing in real time, fast data to improve on-going test planning
5. Mining social media for IoT device problems
6. Using analytic data to improve patterns of attack and test models
7. Changing test plans over time with data driven feedback

To use such huge amounts of data, analytics and big data mining will be needed by the test team. This will become an effort that must be included in the master planning. Planning will need to define schedule and budget, skills, tools and types of analysis that are needed to accompany any IoT product.

Test management will need to advocate for this type of analytics. I expect executives and management to see test as less important compared to consumer/sales data, at first. They will see the operations and marketing people as obvious. However, lessons learned from the embedded systems and rocket industry showed that testers needed to look at the post flight data to see if it matched historical test case data. When a mismatch was found, it implied that new test use cases were needed and changes were rippling in to their test models and test planning. Lessons were learned from data analytics as the software evolved. IoT must do the same.

The organization and testers that master using data analytics to drive future test planning will likely be successful more often. In the production of test patterns for the book “Software Test Attacks to Break Mobile and Embedded Devices,” I produced mind maps and taxonomies of historic errors that escaped testing and but were seen in the field. In surveys of practicing testers, all of whom had error database logs, over 90% did not do any error taxonomy or data analytic analysis to improve test planning. The test organizations or individuals that learn to use big data and analytics will benefit.

Product and development lifecycle impacts on test planning – Dev-Ops and Agile

So much of the world, including hardware, is evolving towards Agile or at least groups say they want to be Agile. This can include the concepts from Dev-Ops (ref). My suspicion is that IoT efforts will be confronted with Dev-Ops and Agile and sometimes aspects of traditional waterfall cycles. These will

form “hybrid” efforts where the IoT goals will be to put together a product as soon as possible (ASAP). Teams will be faced with time to market and trying to keep funding sources happy. This will put pressure on development and testers to answer the following question:

What is or will be the minimum viable “good enough” IoT product?

For this eBook and IoT, the things I would list as possible answers include:

1. Certainly, the communications must work. (Bluetooth failed to work properly and has been viewed negatively in industry and with many users.)

2. The hardware must be functional and have reasonable reliability (a quality).

3. The software must work functionally plus have a few non-functional quality characteristics (ref and see below).

4. The software can be updated to fix issues not met by item 3.

5. Time to market must be met.

6. Costs are minimized for developers or products and to consumers.

To meet these challenges, an Agile effort will want to stay flexible usually with small sprints and then watch the product unfold.

Some Agile writers have said that “in Agile we do not need test teams” (since testing is automated and owned by Dev). This view has faded as discussed by authors such as Crispin and Gregory in Agile testing (reference: Agile Testing – Crispin and Gregory).

To complicate this, one of the industry’s testing gurus, James Whittaker, has stated that “testing is dead.” So many IoT teams, particularly start-ups, have heard “no testing is needed.” I think he should have phrased it as “<traditional, end-of-life-cycle, scripted, unthinking> testing is dead.” I agree with the revised statement.

I believe these all can be and are true, while at the same time advocating that a tester’s role exists in many IoT efforts. The old ways of many testers (e.g., late in the lifecycle testing for days, weeks, and maybe longer, using scripts, and manual brute force) is dead or at least for IoT, needs major changes. Additionally, for some early testing cycles (as noted above) for start-ups, indeed “not testing” may be the way to go.

Testing will integrate into the lifecycle with development and during ops, thus providing critical data (information) to these other efforts. Testers in IoT will need to be skilled in planning at a master and detailed level. The plans will need to address strategies (see the strategy section below) and fit within IoT product costs and schedules. As I have been saying, there is no “best way” for testing, so the planning skill that test managers and senior people need will be greater than ever before.

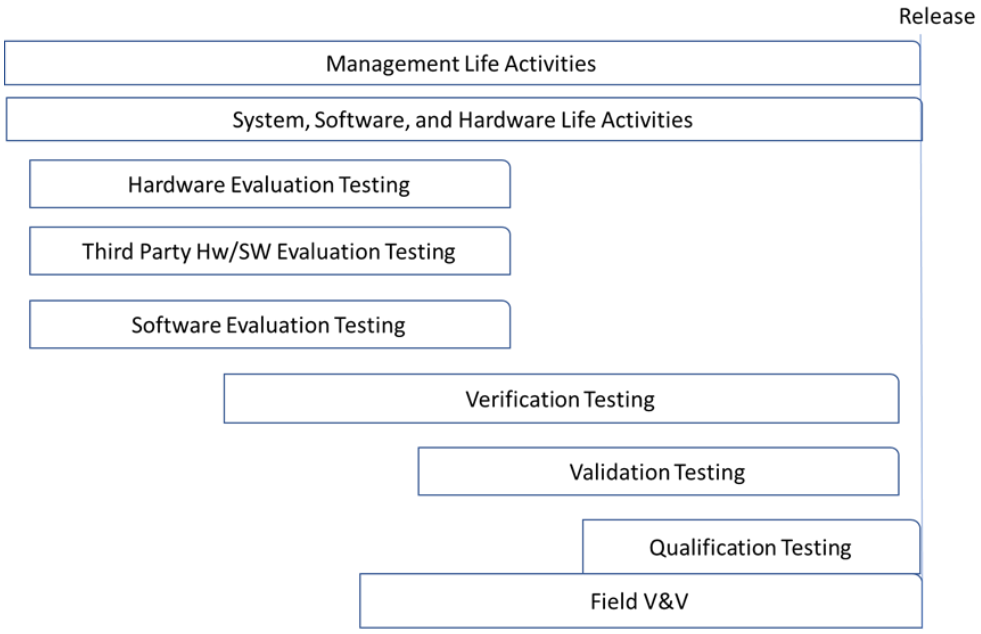

Figure: Example Schedule for Planning IoT Testing (Simplified))

Test planning strategy – the traditional life cycle for hardware and software

This section is applicable to hardware and some software organizations.

In traditional testing, I plan testing while dev builds hardware or software. dev starts with requirements (needs), creates design, and does an implementation maybe with integration. During these efforts, test analyzes requirements and design, makes detailed test plans, creates test designs and implementations (usually manual), and so after an implementation exists I run tests. Everything is good. Of course, there can be different levels of tests e.g., component, integration, and system as I test from the bottom up. This all worked, sometimes well for things like hardware, and other times not so well for things like software that can change quickly.

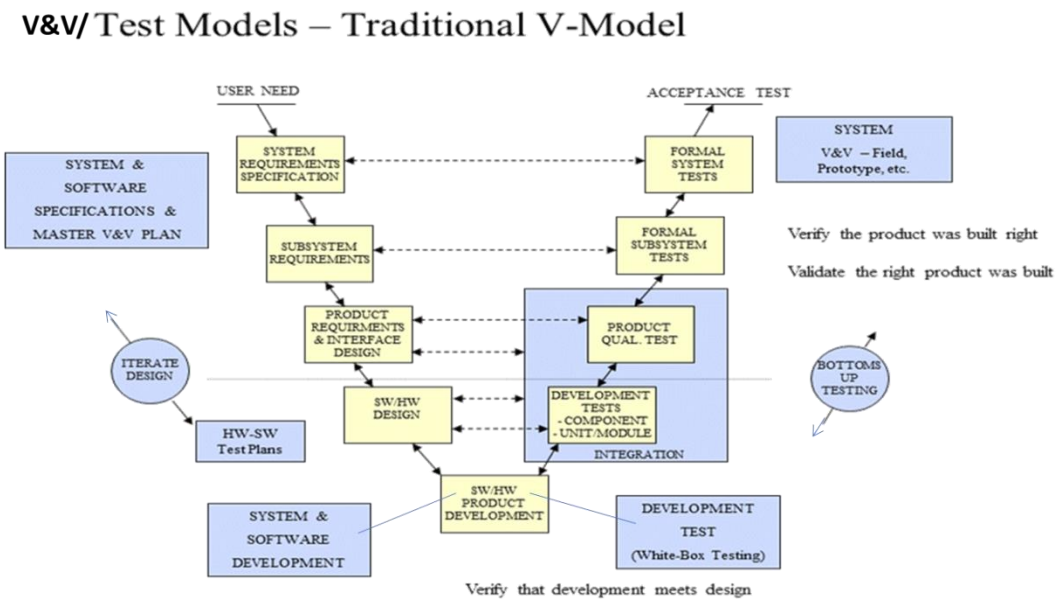

Figure Historic V model of V&V/Test

Hardware, mechanical, physical, electronics, etc. did all right with the traditional model, often called waterfall or V (see figure). The traditional models worked well when teams understood that even the original paper that used the name “waterfall” assumed iteration and cycles of change. However, teams revised as they built. Things worked less well, when teams tried to make things perfect without change and integration. Teams produced working software this way too, but the software industry has statistics that many (50% plus or minus 10%) software projects failed to one degree or another when they tried to follow a “pure” (if there is such a thing) waterfall model.

Given that many hardware and system companies still follow more or less of a “traditional” approach, teams moving into IoT should expect and learn how to work with traditional cycles, with the knowledge that iteration still should be possible. It is also possible to mix traditional and Agile models and techniques together, although some work in scheduling and planning may be necessary.

IoT teams will likely not produce all of their own hardware or software. Teams will use mostly vendors by buying “Commercial off the shelf” (COTS) components. True, some IoT teams will create a specialized sensor or combination of components for their IoT device, but likely a majority of the hardware (and software) will COTS. Outside vendors can and will use different lifecycles than whatever your project is using locally unless there is contractual obligation to do otherwise.

Planning for COTS hardware and/or software testing/integration/V&V

So, what about all these COTS when it comes to test planning? Should you trust COTS?

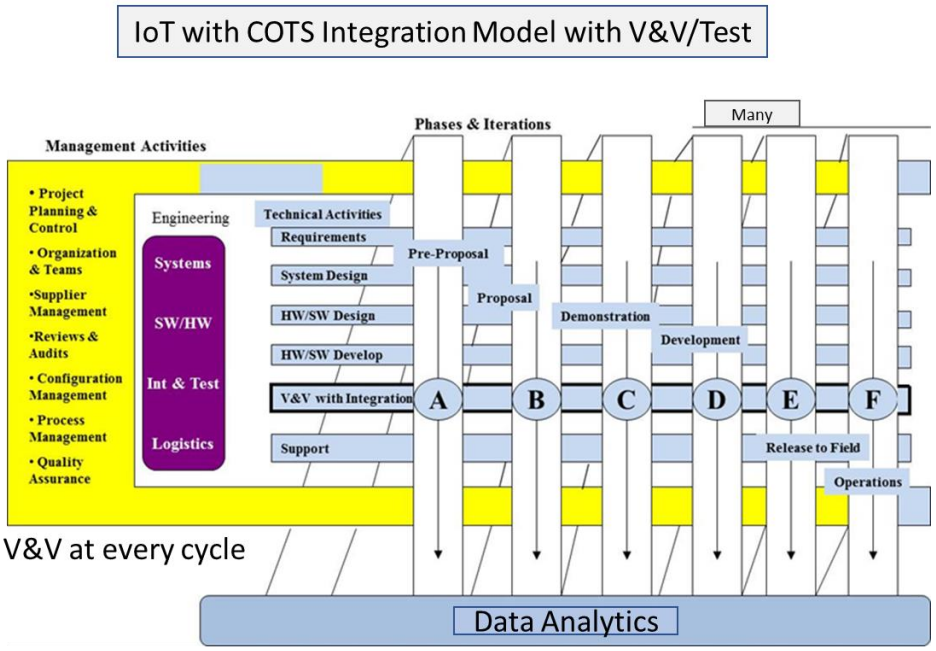

Figure IoT with COTS Integration model picture
Ref: Jon Hagar Class Production from historic sources

When it comes to COTS, teams should trust but verify (an old Ronald Regan statement). To do this, teams should consider a screw model (see the figure) (and if you are not careful, you will be screwed over). In the above "coil" model, I iterate and repeat. The screw model has built in interactions of development and testing. A modification of it can work for teams working hybrids of Agile and Traditional lifecycle production. Items A to F in the model are V&V/testing activities. What these are varies depending on the device, risks, and integrity level.

So a team with testing help would do activities similar to this example.

1. Define a need (story or requirements)
2. Define qualities
3. Define risk
4. Conduct a COTS selection e.g., trade study, competition, or decision

a. As part of the decision process, testing should be done

5. Obtain the COTS items and integrate
6. Conduct integration testing as early as possible
7. Team conducts Dev
8. dev and integration testing

9. Continue Dev-test cycles including regression planning

10. Repeat items 7-9

11. Final integration testing

12. Final V&V testing

13. Final system testing

14. Release

15. Repeat in Maintenance

Basically, in the screw model of Dev-testing the team "moves testing to the left." Team should include planning for new or updated COTS items as they arrive. Remember regression, smoke and new testing are likely (see eBook 4) but watch for the pesticide paradox (https://testwithnishi.wordpress.com/2015/01/03/pesticide-paradox-in-software-testing/).

Here are the kinds of tests to put in the master test plan and strategy when dealing with the more traditional life cycles with COTS, hardware, and even some software.

Note: detailed test patterns will be in eBook part 4

At this point, for more details on planning, I point to the following references (in no particular order of preference):

1. ISO 29119 Software Testing part 1 and 2

2. Systematic Software Testing

3. Testing Computer Software

4. Agile testing

5. Software Test Attacks to Break Mobile and Embedded Software

6. Testing Embedded Software

7. Software Testing (a guide to the TMap Approach)

8. ISTQB syllabi

9. IEEE 1012 V&V

10. Finally, do your own "Google" search on Test Planning and you will find millions of references.

These references cover a large representation of views on testing and do not necessarily all agree with each other, but again, testing is hard and there is no single "best."

Further, in software test planning, the use of COTS that has historic use doesn't mean that new issues and limitations will not occur in the software. Since most IoT software will be heavily based in COTS software e.g., OS, interface protocols, hardware drivers, etc. The planning must address some COTS.

Does this mean I must test fully COTS hardware and software?

Certainly not, since this isn't possible, but some check/V&V of COTS is a good idea (see section in this ebook on COTS)

Can this be done in one or few tests during planning?

Within the side of accepting risks, the answer is yes, but the less test/V&V, the more risk is assumed.

When working with COTS vendors who follow traditional life cycles, IoT teams should understand how the lifecycle works. Late changes to hardware may not be possible. Changes to some COTS elements may not be possible, ever.

This leads us to the idea of architecture, wrappers, (refer to eBook 2) and solving COTS component limitations in implementations the IoT dev team has control over.

Hardware test planning

The following is an outline of some examples of tests that should be included in a hardware focused test plan. The detailed selection may be better suited for a lower level test plan, than a master test plan. Hardware basis tests can include

1. Functionality tests
2. Form, fit and function tests
3. Test to failure
4. Stress test
5. Environmental tests (heat, humid, salt air, cold, etc.)
6. Battery test
7. Power test
8. Material test (strength, color, feel, etc.)
9. Noise and vibe test
10. Long duration test
11. Destructive testing

12. Production-manufacturing variance (6 sigma) sample verification

13. Quality checks – see ISO quality characteristic standard

14. Reliability and failure rate

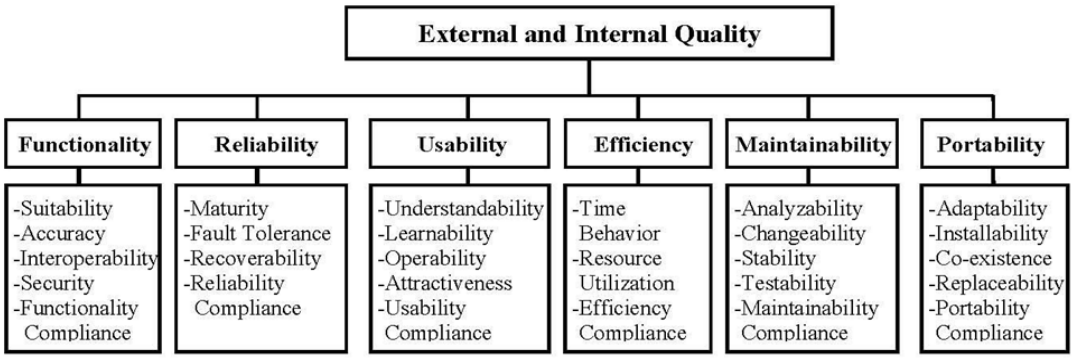

Figure: ISO Quality Blocks

Ref: http://3.bp.blogspot.com/--aCkq4zWzg8/TiSNIKWkJhI/AAAAAAAAAAQ/Mi_kmv6Q2vs/s1600/ISO-IEC-9126.png

The questions to answer in hardware testing planning include:

1. Should the tests be done in a specialized organization lab?

a. What environments with hardware, software and system tooling need to be in place?

2. Should the test be done by a third party have abilities?

3. Should the tests be done by the vendor and how do you trust/verify/witness?

4. Should the test be done using simulations/simulators?

5. Should the test be done in a controlled environment (e.g. test track)?

6. Should the test be done in the field (and how do you get results data)?

7. Should testing be redone on new revisions of the hardware?

8. Which areas of hardware need to be testing including: mechanical, electrical, packaging, form-fit-function, integration, etc.

If you get the feeling hardware testing is hard, good. IoT teams moving from software dev into IoT hardware-software-system Dev, may need to get some new skilled people.

Software tests planning

Software testing planning has been extensively written in the literature. There are whole books on the subject. I am not going to try to cover all that information and I already list the reference books above to help in your learning.

Since IoT testing can range from zero too infinite these items cited are example and starting points. Each will need heavy tailoring and customization for your IoT planning. If you are doing near zero, follow the Agile ideal of doing what is most important first and work from there until time or money run out. If you are more critical and formal, think about looking at the IEEE and ISO references, but even these will need a lot thinking as they are not intended to be used "straight out of the box." If you are in the middle, you will likely want to mix many of the reference above (remember I said high tech is not easy).

Teams should not over promise in test planning. You should set expectations low if they are "near zero" levels of testing or even so called middle levels of test effort. Further, even if more testing/V&V is possible, teams in the planning must make clear that all testing is sampling to provide information for those making decisions. Many managers and executives do not understand that complete testing in provable impossible. Our planning needs to make this point clear as testers are the educators and information providers to our organizations. Remember, test/V&V cannot show there are no errors or issues, but only provide some indications of IoT product qualities under specific point situations.

Key points in software test planning include:

1. Test at many levels from the lowest to the highest (structural and functional)
2. Planning needs to address strategies
3. Planning needs to fit with resources (skill, budget, people, tools, environments, etc.)
4. Planning needs to be flexible
5. Quality factors that need to be addressed (functional, non-functional)
6. Integrity and risks should be factored in planning
7. Hardware-software integration and interface efforts

Software test planning

In system testing I integrate and bring the IoT hardware, software, and maybe other systems into an integrated system or system-of-systems. This is the big picture.

The ideal in system testing is that the testing of the lower levels and components including integration testing has been completed before I do the whole system. Small project may be able to jump to system testing with fewer levels of testing, but the more complex an IoT device-systems, skipping to many early

levels is not advised. I advise this because if a lower level problem has been missed and appears at a system level, it can be hard to tell where the problem is coming from.

I have seen team chase “unverified failures” (faults that could not be repeated on every test) because of software faults that were interacting with hardware to make it look like the hardware had a problem and vice versa.

I will present this section and test list assuming earlier and lower levels of testing have been done.

System test planning includes but are not limited:

1. Normal day tours
2. Off normal day tours
3. Stress testing
4. System quality verification

a. Repeat selected hardware and software tests

5. System validation with users/customer or surrogates in the loop
6. Simulation of the systems testing
7. Field testing
8. Demonstration (a system as it is to be delivered)
9. Alpha/beta testing (define where)
10. Crowd testing
11. Quality (s) testing
12. Usability testing

Item 6 may be needed for many IoT devices because the real world systems an IoT device is intended to be used in, may not be available or difficult to implement. For example, there are hundreds (millions) of home system that might be in use. I can do system testing with all of them. So items like 7 and 8 become problematic. Now using system testing item 9 and 10 may help but here you run into combinatorics problems (see ref in eBook 4 on combinatorial testing).

System IoT testing will like be taking “bests guess” on some of these.

However, some system testers will be only doing system testing e.g., a so called smart city testbed. These are organization doing “procurement testing” (ref above). Groups doing this kind of testing my go

by names such as IV&V (ref), testbed, agencies, etc. These groups will not be doing the lower levels of testing, except in audits or assessments. These test plans would look very different.

Dev groups doing system testing will need to think hard on the planning. This section is just a short start.

Planning the Agile test life cycle (which most efforts strive for)

I expect most IoT efforts will follow Agile concepts, maybe combined with Dev-Ops (see below) as the two are closely related.

Most teams want to be "better faster cheaper" within a balance of these. The old joke was pick two of the three, but in IoT teams will use balance and agility to constantly adjust these. The balance on each of these will change over time and product releases.

Readers should read and really understand the Agile Manifesto. It states:

"We are uncovering better ways of developing software by doing it and helping others do it.
Through this work we have come to value:
Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan
That is, while there is value in the items on the right, we value the items on the left more."

These are a simple and beautiful set of words that took a lot of time and thought to craft. You should take time to read and appreciate. I have found groups that claimed Agile. For example, on team said they were Agile but on quizzing, all they were doing is hacking the software and creating zero documentation. They missed the point. Reread the manifesto.

Next I recommend Gregory and Crispin's book on Agile testing. I have both books on us shelfs and talk/listen to these authors every chance I get.

I am picking several key ideals from Agile that I think IoT teams should know and within the right context, practice to build skill and successful system devices. These are:

1. Be flexible in responding to change on plans, products, contracts, etc.

2. Continuous integration to at many levels, e.g., software to software, software to hardware, system, and systems of systems.

3. Continuous testing including

a. Verification and validation by the team e.g., developer testing, modeling, analysis, inspection, etc.

b. Acceptance test driven development (reference Agile Software Testing – Crispin and Gregory)

c. Acceptance/qualification cycle with short and/or automated efforts

d. Ops testing driven by users, testers, and data analytics

Some critical IoT efforts might justify and need concepts like outsource Testing as a Service (TAAS) and IV&V, but only if the integrity level (reference IEEE 1012) justify the cost and time. Examples of where a high integrity level might be found are:

1. Safety related items

2. Hazard items where large monetary loss are possible

3. Items where large money can be lost (note this may not be easy to spot)

Continuous Test - An Agile test plan might look like

1. Test/V&V COTS as soon as they arrive (see above)

2. Have an IoT team test/V&V person support any COTS customization issues

3. Hardware tests (see above)

4. Assist and evaluate TDD using one on one and/or metrics

5. Develop ATDD from id to design to implementation and provide these to dev ASAP

6. Conduct Software test using

a. ATDD

b. Attacks

c. Exploratory

d. Tours

Continuous integration – Conduct integration as you go in a CI test plan. This might include

1. Obtain COTS product what has been V&V/tested

2. Do integration cycles (bottoms up, top down, iterative screw)

3. Assess if wrappers are in use and does the testing happen inside and/or outside of wrappers

4. Build iteratively based on criticality, schedule, and importance

5. Test interfaces

6. Build CI test features over time

7. Check CM components (and work with mutants as needed)

8. Build data, stubs and/or drivers if needed

9. Verify to ICDs

10. Validate ICDs

11. V&V to standards of products, protocols, communication, etc. as needed

I take from the manifest the following points. In Agile teams learn as I go, so I must respond to change in plans as the project unfolds. I communicate, which maybe should have been on the critical list, but I believe in all engineering and business communication is as critical as thinking (ref). I agree that testing in Agile succeeds based on skill. Reading this eBook give only some knowledge. Skill is different from knowledge, so an Agile team will have people who practice the craft of testing to build skill in software, testing, hardware, and likely even systems critical thinking.

If you don't have skills, continuous planning, communication with stakeholder, and critical thinking, then just spouting Agile ideals or traditional standards will not save a project. Failure is likely

Planning the dev-ops test approach

In part, Dev-Ops was an outgrowth of the Agile movement. Dev-ops seeks to integrate development and operations to the advantage of both parts of the organization and to bring about quicker product updates. To be successful, teams and management must buy into dev-ops concepts as well as work actively to improve their skills (not just call themselves "dev-ops"). This kind of change can be a culture shift for many organizations. Organizations must look for improvements, identify bottlenecks/road blocks and then attempt to overcome them. My focus here is on the test planning aspects a Dev-Ops team may want to consider, including:

1. Continuous test (CT)

2. Continuous Integration (CI and delivery)

3. User as tester

4. Full product lifecycle (womb-to-tomb) V&V/testing

5. Comprehensive V&V with the right mix of automation (CT/CI) and manual exploratory test

6. Developer test (see agile)

7. Shift testing left (on the schedule) into to dev phase with a short (automated) "test only cycle"

8. Tests like those used in production environment testing

9. Testing in production ops (e.g., Chaos engineering)

I will not go into great details on most of these due book size limitations. If the above concepts are unknown to readers, more reading in other reference sources and the internet is advised.

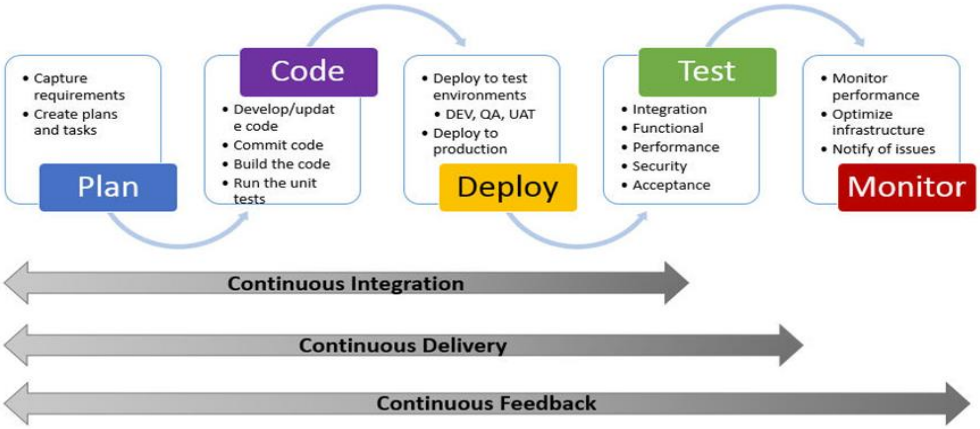

Figure Dev-Ops Planning

Ref:https://media.licdn.com/mpr/mpr/AAEAAQAAAAAAAAysAAAAJDRmMzNlOGNiLWQ4MWMtNDBkOC1iNjBiLWNjYWY4NGFmZWRmYw.jpg

Dev-Ops builds on Ag3ile many Agile concepts. The planning cycle has testing in play and addresses operations. Not as clear is the Continuous Integration (CI), Continuous Test (CT), Continuous Delivery (CD) and user as tester. I expect Dev-Ops test teams should be a major consumer of use data (see sections data analytics).

Why are operations important for IoT?

Well given that I expect Agile or iterative traditional cycles to be in use on IoT Dev, that many IoT devices will be pushed into the market place based on time, and that most IoT devices will generate a lot of data to operation centers, it is reasonable to expect that many products will conform to a Dev-Ops model. Further, I expect that many IoT devices will, in part, be “tested” by users.

What is needed for Dev-ops testing to be successful?

1. Team attitude and commitment

2. Communication from the IoT device(s) into the cloud

3. Data streams and processing

4. Service architectures

5. Avoid teams that are essentially in “silos” i.e., communicating only upwards and no within or across the organization.

The service architecture will likely include the following development, test, platform hardware, infrastructure, data analysis, and ops. Teams will likely mix and match each service area. Some service areas will be kept within the organization while other services are outsourced. Most teams will not establish their own cloud environment. Further, many hardware-based companies will outsource development, testing, cloud services, data analysis, and infrastructure to other service providers.

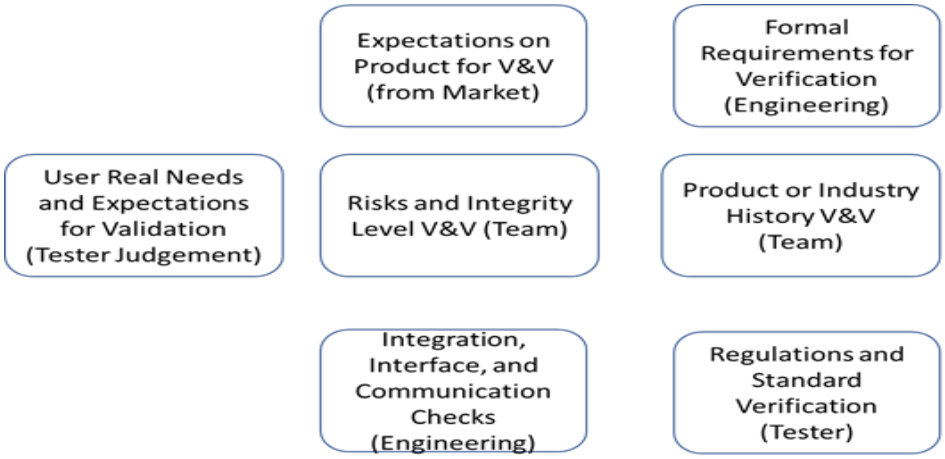

Figure –Factor to Consider in Test Planning
Ref: Jon Hagar Class Production

What are some possible resulting stories?

Well, consider the Tesla self-driving car (SAE level 3) story. Tesla had done some testing because the system (car) functioned in the field. Yet in 2016 (https://www.theguardian.com/technology/2016/jun/30/tesla-autopilot-death-self-driving-car-elon-musk), a person was killed while the car was in "auto drive" mode. According to reports, the user was not ready to take over control of the vehicle, instead was viewing videos on his mobile phone. (The data from several devices were used to construct this story.) Now, Tesla has stated "a user cannot do what this user was doing," however, users are human. Users are likely to do similar scenarios on other IoT devices and systems Some of these users will suffer the same consequences and as developers/testers, I should build my systems with this likelihood in mind.

What should development and test do?

The answer is still up in the air. The U.S. Government is currently working on self-driving car policies (ref). However, testers in Dev-Ops organizations cannot wait on the rules and standards. Engineering is happening now. As the Tesla story indicates, as long as users understand the limitations, which are need to be better advertised since many Dev-Ops organization do not, then Dev-Ops is the way to go.

For testing in Dev-Ops, I follow the Agile testing approach (See http://www.ibm.com/developerworks/library/a-devops9/). And get a product out using short cycles. Given short cycles, most Dev-Ops advocate a large dependency on test automation as well as manual experience-based approaches, such as exploratory testing, attack-based testing, and testing tours.

Automation in Dev-Ops testing can include:

1. Automation of repeated or labor intensive testing (e.g., regression, critical risk areas, etc.)
2. Use data analytics to know where errors cluster and, therefore, where to automate more testing
3. Automation of CT, CI, and CD critical risk areas
4. Test outside of the UI (e.g., use API, unit, integration testing, etc.)
5. Design the IoT software, hardware, and system to support automation
6. Plan and estimate testing to include automation (Remember, automation takes resources)
7. Measure automation to be able to report cost savings and identify improvement
8. Remember test automation is development and has repeated development cycles (be Agile)
9. Leverage tools (e.g., open source and commercial tools that fit your processes)
10. Test in the cloud so resources (tools and computers) can be expanded without buying more software or hardware, as needed

In Dev-Ops, the team and testers need good configuration management to function effectively with hardware, software, systems and documentation, including:

1. Tools
2. Development code
3. Development hardware
4. Development support documents
5. Test environment
6. Third party hardware, software, and systems which are in use
7. Numerous device configuration

Finally, on many Ops teams, members may not be called “testers” but they will need to think like testers. Ops staff will need to watch for issues, find bugs, and define unhandled (untested) use-cases that have not been seen before.

I suggest the Ops team have people skilled in test thinking. Why?

For example, some Ops teams will find data that is not a pure error, but where a tester might be able to quickly identify a trend which, if followed, will find an issue. Testers will use data from exploration, ad hoc, tours and attacks to find bugs. Testers develop mental models of software-system “smells” and

patterns, which, if followed, will yield a bug that should be fixed sooner rather than later. Thus, I recommend the Ops staff either have testers on board or have advanced tester skills. For more see https://en.wikipedia.org/wiki/DevOps.

Planning the hybrid test life cycle

This is where the IoT lifecycle may have a mix of traditional, Agile, Dev-Ops, and iterative approaches covering hardware, software, systems, and operations. Smaller IoT devices may not face the hybrid lifecycle case or there may be a smaller mix. However, it is likely that if the IoT device is part of a larger system or the team is dealing with a larger system, then the hybrid case is more likely.

Many IoT teams will want to follow a "keep it simple stupid" (KISS) approach and only deal with their small corner of the world in their device. This is viable, but when done I recommend the literature and information about the particulars of the IoT device clearly define and state what is in bounds and out of bounds for the device/system. This is because most consumers will expect the IoT device and "system" to work out of the box, when they just plug it in. They will not understand that the development and testing was done on a "smaller" view of the lifecycle and user operations space. If teams can restrict their device to what was really developed and tested, happy consumers will be the result.

Figure: All the Good Ideas Come Into Play
https://qualityhouse.com/appdata/pics/c_p/5fb772079ea6c82789f486c6377448d3.png

However, most of us that have worked with systems and systems-of-systems know that during integration (of hardware, software, networks, IoT devices, and systems) is where (some really big expensive) problems appear and maybe after long periods of use. This is because in software, coupling can "hide" errors until the certain real world inputs and device conditions are encountered.

Many IoT devices and systems will function in the hybrid development space of different lifecycles and approaches between hardware, software and systems. While the hybrid world and system impacts are larger topics fully address than in this eBook, I provide a starting point for teams.

In the hybrid case, learning from history to support test planning assures the following happens (a partial list):

1. Architecture
2. User needs (requirements, stories, models, etc.)

3. Design and implementation

4. Data analytics (in development and ops)

5. V&V/Test and Test Taxonomies

6. Ops and users as testers (generates data as part of Dev-ops that is fed into new development)

7. Integration, Integration, Integration

The IoT team working in a hybrid lifecycle will need to master and use the earlier engineering-planning information and concepts listed above.

Sources of errors for hybrid IoT systems include:

1. Environmental

a. Low signal quality, low power, lack of resources

b. Input space not fully tested

c. Physical environment factors

2. Internal to the device

a. Functional and computational problems

b. Quality non-functional failures

c. Data and storage issues

3. Output from the device

a. Interoperability to other devices

b. Big data analytics and stale data (especially in notification messages)

c. D2A and environmental mismatch

d. Timing and performance issues

Test budgeting, estimating, and scheduling

An introduction to test budgeting, estimating, and scheduling are presented in my eBook part 2. Please refer to that source or the following sources:

1. https://en.wikipedia.org/wiki/Test_effort

2. http://www.guru99.com/an-expert-view-on-test-estimation.html

3. Proposal Preparation – Ann and Rodney Stewart

4. The Tester's Pocketbook – Paul Gerrard (see page 8 on the "equation of testing")

There are very few books on test budgeting, estimating, and scheduling. I do not necessarily recommend nor follow exactly any of above pointers but they, along with eBook 2, will give readers a reference point to continue learning.

Most of us learned it by doing. The above references are only starting points. Perhaps another eBook is needed on these topics.

Planning wrap-up: regression test cases

Most software V&V/teams are familiar with the idea of regression testing (see https://en.wikipedia.org/wiki/Regression_testing). Teams understand that during development as the software changes in areas that are already tested, regression testing is needed to answer the question "did I break something that was already working". Regression happens in software partially because of coupling (See https://en.wikipedia.org/wiki/Coupling_) or lack of team understanding of design.

Teams develop different strategies for regression testing, which include:

1. rerun all testing

2. rerun tests in the related by function or code areas

3. rerun a standard regression test suite

4. rerun what the team things needs to be run

Each of these strategies has plus and minus considerations. It is out of the scope of this paper to define a complete regression strategy and approach. Indeed there is debate in the test community about the validity of regression testing (see http://kaner.com/pdfs/gui_regression_automation.pdf).

The validity question comes from the pesticide paradox

see https://en.wikipedia.org/wiki/Paradox_of_the_pesticide

In the pesticide paradox the ability of a test case to provide information diminishes the more times it is run without changes. Rerunning regression tests over and over is thus of questionable benefit, but most organization employ some levels of regression testing/analysis. The debate about regression test will go on.

My personal experience on one project was that I did a mix of items 2, 3, and 4. I did RBT and test planning. I changed "old" test to some extent to avoid aspects of the pesticide paradox, while still addressing regression. I made sure a test existed that could assess the error or change in the software. I did a lot more tests during O&M because I employed V&V/test automation (see automation section below).

On one O&M cycle I had the following percentage allocations:

Standard regression tests 20%

Expert team selected regression tests 30%

New tests (includes developer tests) - 30%

Test traced to coupling - 10%

Historic system test not run in a while - 10%

Now the percentages are nothing magical. In fact on other O&M cycles the percentages looked different. The key was I applied thinking test planning with RBT early in the O&M cycle. I got stakeholder acceptance of V&V/test plans, and I modified test plans as needed during the cycle.

Also, the reader may notice the last category percentage, which may seem "new". This was added by the team and was different than "standard" regression concepts. I learned over the O&M years that it was a good idea to pull out historic tests that had not run for a while and rerun them with changes for the new O&M cycle. I did this during most O&M cycles, on slowly over the years (it was a long term project), the team would cycle through the major of historic system tests. What the project was looking for was a regression that had escaped previous test cycles. Now if I had the time and high levels of automation, I could have avoided this step, but I did not have the time and budget, so "cycling" over the test suite was a compromise.

Strategy: The big picture starting points

My definition of strategy for this eBook comes from a dictionary.

Strategy - the art of devising or employing plans or stratagems toward an overall goal [1]

Test Strategy - part of the Test Plan that describes the approach to testing for a specific test project or test sub-process or sub-processes [2]

Test Basis – bodies of knowledge used to drive test planning and design including techniques and processes which lead to detailed test activities.

Testing is about providing information about the qualities of the product to stakeholders. Many people have observed that quality is value for which someone is willing to pay. Products have many qualities that people should investigate during testing. The first quality that most people think of for any product is function (verify that it works to defined requirements). However, this view is narrow. There are other qualities just as important including: performance, safety, security, usability, operability, reliability, and others.

There is one quality that most IoT stakeholders want to avoid, that being a (bad) quality of having errors or bugs. Many authors on testing advocate the only good test is one that finds an error/bug. This is a simplistic view. Many tests that do not find a bug still provide useful information about a device. It is true that testers would like to find errors when they exist, and so I should plan, design, and modify tests to optimize error detection. I will address this "bug hunting" aspect of V&V/testing throughout these eBook parts. Many of the test techniques, patterns, attacks, tours, etc. are based on the ability to find errors, but keep in mind there may be other kinds of information V&V/testing should provide. Test strategy as part of planning should support many kinds of information gathering.

In the past, many of us separated system, hardware and software V&V/testing, but with IoT these divisions must blur. Given these viewpoints, testers in IoT should understand that classic testing of just the software will not be enough. I advocate that testers will also need to take on system testing including assessing the unique aspects of hardware.

There is great wisdom to be found in standards whether tailored or used completely. And, while there is some controversy about the use of standards in the industry, many useful IoT concepts can be gleaned from IEEE 1012 and ISO 29119. Any users of standards should be tailoring, especially in the case of ISO29119 and analysis of integrity levels in IEEE 1012. I also expect that most testers working on planning and strategy for IoT devices will have some familiarity with books in the reference appendix. Certainly my view of test strategy is different from pure ISO 29119, IEEE 1012 or other parts of the industry, but my definition is my beginning at tailoring for my needs in IoT. We all need to learn tailoring and how to reuse modification lessons from such references, as well as to practice the skills of testing.

With these expectations for readers using other references, the sub-sections of this chapter provide an overview of test strategy concepts. Experienced testers can skip these sections, but readers who are not familiar with strategic concepts can begin learning from the following sections.

Strategy and basics

Do plans include strategies or do strategies drive plans? Well, yes, both.

Project test strategies are the starting point to define the test plans. Test strategies and plans can be governed by organizational test strategies and/or policies (i.e., regulations and standards). Test strategies are the high level (referred to as the 10,000-foot level) and lead to the test plans that are more detailed (the 1,000-foot level). ISO29119 part 1 defines that strategies include the following: "the test practices used; the test sub-processes to be implemented; the re-testing and regression testing to be employed; the test design techniques and corresponding test completion criteria to be used; test data; test environment and testing tool requirements; and expectations for test deliverables." Test strategies are typically documented as part of the overall top level (or master) test plan.

IoT Test strategy

I find one or two common, but often unspoken, test strategies in use in test groups. These may be used in combination with each other or alone. Testers use them and the supporting test basis over and over,

as if they were the only way to do testing--which results in limitations of test thinking including: missed errors, expensive testing, and testing that takes longer than stakeholders want it to take. Do you remember Einstein's definition of insanity, "doing the same thing over and over again, expecting a different result each time" (https://www.brainyquote.com/quotes/quotes/a/alberteins133991.html)?

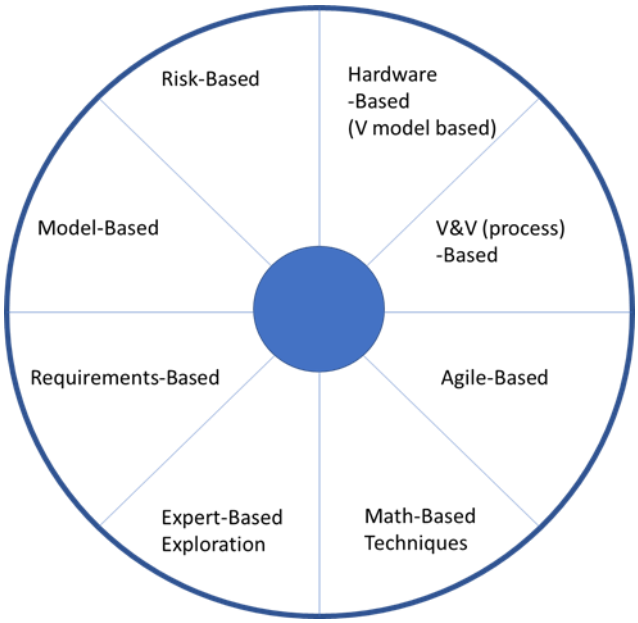

Figure: Test strategies and basis of support for test plans 2.0
Ref: Jon Hagar Productions

As depicted in the above figure, test strategy and basis forms the supporting spokes for improving test plans, which in turn leads to improved test designs. There are and can be other strategies; some readers may have different names for these or feel something else should be addressed strategically, but for this eBook, this is how I am defining strategies to help in IoT planning.

Here are the common IoT test strategy and basis (with the names that I use and most are shown in the strategy wheel) that you may wish to consider for your test planning and strategies:

1. Risk-based testing – testing in which the management, selection, prioritization, and use of testing activities and resources are consciously based on corresponding types and levels of analyzed risk (for example see ISO 29119) and/or integrity levels (for example see IEEE 1012).

2. Model-based – testing in which models (not mental models), such as Unified Modeling Language (UML), UML testing profile (UTP), and others are used to drive one or more testing activities, such as to: manage, design, implement, execute-automate and/or report testing.

3. Agile-based – software development methods, including testing, in which requirements and implementations evolve, being done by cross-disciplined, self-organized, and collaborative teams including testers. In Agile, testing is typically practiced by the whole team throughout the lifecycle.

4. Hardware-based (V model based) – a sequential development and test process in which activities progress steadily through phases, such as: proposal, initiation, requirements analysis, design, implementation, testing, production, operations, and maintenance. In traditional efforts, testing is often done by a specialized independent group towards the end of the life cycle. V-model based testing is common and likely to continue for hardware developers and teams. It is often called the "V" model. V model based testing has been used by many software teams, though many "struggled" with it

5. Requirements verification checking and documentation-based testing – testing in which the requirements or other software artifacts are used to prove (as in verify but not mathematically prove) that the code satisfies requirements and artifacts. There is often a legal reason to show that the "shall" statements are met, as well as demonstrate compliance with standards or regulatory specifications. The "checking" tests are performed and documented in written (scripted) test procedures and reports.

Note: some would consider this the same or a close cousin to V-based, but I separate the strategy since verification checking or documentation can be done in any testing.

6. Expert-based exploratory – testing in which the tester's knowledge, skill, and historic practice are used to plan, design, implement, learn, and report about the testing. Many (possibly most) testers employ some aspect of experience-based testing, although some use it more extensively. Supporting concepts include: error guessing, ad hoc testing, and exploratory testing.

a. Break it/Attack-based with patterns and tours – testing in which the tester attempts to find errors by using patterns (attacks) or meta-patterns (tours) to find errors in the software (to break it). This strategy is closely related to experience-based testing and often includes aspects of risk-based testing.

10. Math-based – testing in which tests are planned, designed, implemented, and/or analyzed based on mathematical concepts and techniques. The mathematical concepts/techniques include: statistical, design of experiments (DOE), formal proofs, combinatorial, random, fuzzing, and domain (set theory, such as equivalence classes and boundary value analysis).

11. Verification and Validation (V&V) based – testing in which testers try to show that the development efforts have created the "product right" (i.e., meets requirements, design, standards, etc.) and "right product" (i.e., meets what the users and/or customers want). Products being V&V'd can include operational concepts, requirements, design, models, and implementations as any of these can have errors. I feel because of my "systems" view, that t e V&V should be a common strategy for IoT and V&V-based usually uses several of the earlier strategies of this list.

Most comprehensive project test plans will use several of these strategic methods in combination. In combinations, one strategy may be chosen as "primary" and then others used as needed. There is no "best" strategy or single one to use. Also, you can define other strategies, which you may wish to add to this list. Refine your strategy based on your local context.

What is the best strategy and a checklist for strategy selection?

Since there is no secret single or "best" strategy except to consider a project context at all points in time. A tester or test team must use critical thinking to select a good mix of strategies to go into any plan. Further, once a strategy and plan has been defined, testers should expect changes to strategies and plans, although detailed plans will likely change more than test strategies for many reasons. Here is an outline of a checklist to consider for test strategy selection:

1. Focus on context. Context will include things such as, cost, schedule, product nature, organizational policy, regulations and standards, customer or stakeholder expectations, and test team skills.

2. Include risk and integrity level of the IoT device and maybe even the system or systems it will function with or in. The more risk or higher integrity level, the more planning and strategies are needed.

3. Consider how strategies can be mixed and matched for an optimal mix given context.

4. Which strategies will offer the most effectiveness in testing (i.e., how can you optimize your testing meaning to "kill several birds with one stone")? Some strategies may be hard to implement, but offer very effective testing, in which case you may want to consider the extra work to set them up.

5. Make sure you include V&V of hardware, software, and system aspects of the IoT device.

6. Each strategy requires skilled and even expert testers, so make sure you have a complete test team.

7. Finally, how does the lifecycle of the product impact the strategy? For example, a new start-up product will have a different strategy mix than one that is historic and only running maintenance testing.

Approach an IoT test effort by (first) critically thinking about what your strategy and test basis will be. You do not want to be trapped by thinking there is a single "best" strategy or that the strategies you are most familiar with should be what you select just because you are familiar with them. This is a common trap many testers fall into and yet they complain that something needs to change to make testing better.

The strategy/basis will flow into the test plans and then down into the test design-implementation. When teams incorrectly limit strategy/basis, then they limit test results and then IoT product success may suffer. Good testers can balance and optimize the test strategy and basis going into the IoT test planning.

An example: A beginning test strategy

An IoT project I know had a strategy of verification and validation based in the risks of the system with specific plans to use models, test attacks, math, requirements checking, inspection, and analysis during a traditional lifecycle development model. It was costly, but then the system involved millions of dollars and even loss of life. The IoT team cared deeply about testing, standards, policies, strategies, plans, and doing the actual work.

The team had strategies of V&V and experience-based. They had had problems with devices that were left on for long periods of time and having hardware "melt-downs." So, their test plans would always include a long duration test of the hardware performed at a max level of usage. But, when it came to the IoT software, the long duration hardware test did not stress test software, so they were not really doing a long during software stress test. After going live, they got reports of the software failing after being left on for very long periods of time (thousands of hours) with the software receiving stress usage cases. They ended up having to fix the software.

They updated their test strategies to include some more of the earlier strategy list with plans to include more software stress testing on long duration usage. So, the test group's expanded test basis of having a risk focus, multi-prong approaches, verification, and validation test strategies bore success in the long run as it continued to evolve over many years of the product's use and maintenance.

What is a strategy for the individual IoT tester?

I hope that testers in even the lowest level of IoT will practice more strategic thinking about testing before just jumping into detailed test plans, design, and running tests. Keep in mind the "bigger picture" at the 10,000-foot level. When appropriate, add strategies over basic requirements verification scripts such as math- or experience-based software testing. Testers should work on mastering application of other strategic approaches and bases. This can be started even in the daily detailed test design and planning. This gets testers ready for more advanced planning in their IoT career.

System V&V planning: start with a combination of strategies

A robust start in V&V/test planning covering the hardware, software, and system could include:

1) Risk-based test planning to refine the below selections on an ongoing basis

2) Requirements verification checking allocated during sprints and automated at the "end"

3) Analysis of components during development

4) Review and inspection of component products as they are created and evolve

5) Developer or supporting testers doing attacks during development using lower level test techniques

6) Experienced-based exploratory testing early in the life to "learn" the software and find bugs

7) Math-based testing as the components integrate and mature

8) Experienced, attack-based testing as other strategies and components are completed

9) IoT Device Testing at End-Sensor, edge, Comm-Gateway to the system end

10) Cloud based IoT Platform Testing

11) End-to-End Testing of key quality characteristics including Functional, Performance and Security

12) Realistic Field Trials by third parties, IV&V, or government

Depending on a personal skill and the organization maturity, I might add other strategies and reduce efforts spent on the above list, but I have run projects with a plan outline supported by the above strategies.

Now, let's refine this starting point based on a project's focus areas (e.g., hardware, software, etc.).

Hardware V&V strategy

In this planning, a team has more of a focus on the hardware. This is not to say that software can be ignored, but teams sometimes focus on a component, while leaving the software to someone else's strategy. I will start with the list from the "system combinations" section above, but refine with the following as examples:

1. More focus on experience-based testing using inspection

2. Have experts to do analysis of the hardware (e.g., electrical, mechanical, form, fit, function) before the "final" hardware is created.

a. Prototyping and hardware "models" using things like 3-D printing fits in analysis

3. Off-the-shelf selection analysis decisions

4. Hardware production facility selection analysis decisions

5. First production line run test and inspection

a. Risk-based

b. Math-based (test to failure, test to stress)

c. Requirements verification checks

6. Quality control checks of randomly selected samples once full production starts

Software V&V strategy

Again, I would start with the "System Combinations" strategy from above, but refine for a software focused team. The team cannot ignore the hardware during testing, because at some point the software and hardware must integrate before proceeding with the system testing. However, I am interested in making the software "good enough" to move on the hardware and the into system efforts. I have seen cases where a software bug "burned up" expensive hardware after integration on the real hardware happened.

Story: Burning up production hardware because of software problems
I was testing system that had limited budget, hence limited time for testing, and so management

decided to use the actual hardware of the system as part of the test environment for software. This strategy reduced the budget because a complete specialized software test environments did not need to be created. So, during one test cycle, the test team installed the software, powered up the system, and started the test. Right way a piece of production hardware had a motor actuator turn on full power, run up against the hardware stop, and stayed there at full power (think if your car was against a wall with the accelerator fully on). Before the testers could stop the test and power the system down, the motor burned up. The costs were: 1) $100,000 replacement motor system, 2) the test managers job (not me), and 3) a learning experience for me.

Here are my refinements for system strategies. These have a more software focus, which can include the following from "Software Test Attacks to Break Mobile and Embedded Devices":

1. Static code analysis attacks as the code is developed

2. Software structural test, attack-based testing including logic, data, and missing cases

3. Integration attacks using emulation, simulation, or prototype hardware before going to real hardware

4. Test attacks after integration on the hardware in either the lab or real world.

5. System software tours and math-based testing including combinatorial, mobile app testing, and cloud end

6. Regression and retest issues planned

I would certainly consider test automation, model based testing, and other concepts if the organization and skill base is mature enough. However, automation needs care (see http://www.satisfice.com/articles/cdt-automation.pdf).

The above list is only an example to use as a starting point. I would expect most real IoT efforts to be different.

Ops V&V strategy

In part 2 of this eBook, I talked about Dev-Ops and I hope you have read that so that this short section makes more sense.

In Ops V&V, I expect V&V/test staff to be in play (actively using ops information for more cycles and product changes). I also expect short and frequent development cycles. The test strategy, while building on early test planning and strategies will be different from "new development." I believe that strategies will be selected and will need to address the following considerations, including

1. Data analytics

2. Rapid response-correction teams

3. Security and privacy

4. Safety or other “qualities” deemed by the stakeholders to be “critical” to the business

5. Product lifecycle stage (new, mature, near retirement)

6. Risks and integrity of the product

7. Regression and re-test

8. Users as testers (feedback from the actual usage in the field via data and data analytics)

Since IoT spans devices from industrial to consumer use, the selection of strategies during Ops will vary. You and your stakeholders should create your own factors list. The strategy list above may get expanded as teams understand more about IoT Ops.

Integration and Test Planning Strategy

Most traditional IT and software development teams do integration and perhaps found issues during integration. Teams should be aware of integration approaches (aka strategies) such as bottom up, top down, continuous (agile), mixed, etc. (see https://en.wikipedia.org/wiki/Integration_testing.

Many software teams are less familiar with hardware component integration and set-based design (http://www.doerry.org/norbert/papers/SBDFinal.pdf) for hardware because they have traditionally been software focused.

The integration of hardware elements and software should have their own schedules and plans as well as should be on master schedules and plans. The more complex the system, the more planning is needed and the more likely plans will change. Places I have worked did weekly--and at times, daily integration schedules and plans. Things changed rapidly and I was always looking for workarounds, which often required creative thinking and problem solving.

Teams will find as some general rules of thumb during integration:

1. The hardware will be “late”.

2. The hardware will have problems, which are often fixed by software changes.

3. The test team will get “crunched” (meaning schedules will move to the right but the end ship date does not move).

4. Integration-testing will find problems in both hardware and system.

5. System testing will find more problems.

6. Regression and re-test will have more cycles than the schedule planned (I always planned 3 and expected 6).

7. Customers, managers, and/or stakeholders will always want better, faster, and cheaper.

Teams will need to select V&V/test integration strategies and plans based on the local context including these rules of thumb. I do not give an example strategy list here because I think the rules of thumb are a better point for consideration in integration strategy selection, though I would certainly start with the "system integration strategy combination" list as my starting point.

System (and system of system) V&V test planning – a conceptual introduction

Our bet is IoT system and system-of-system testing will be ignored or lacking for some period of time. Development organizations will focus on the individual devices. There will be "standards" for product interfaces and protocols, but currently those are in a state of flux. IoT products will be promoted as "compatible" but minimal real testing and assessment will have been done. In part, this is because the question of ownership may be undefined.

Who owns the system or system-of-systems? Which owner has the responsibly for the IoT devices all playing nicely together with the possible interfaces and protocols? Does the city government own the "smarts" of the city? Many will say "yes," the owner owns the problem, but currently "owners" do not have the skill and knowledge to test systems or systems-of-systems. This can be seen in homeowners who use default router passwords and configurations for their current modems and computers. Or, in governments who are barely able to handle existing IT systems and will take time to expand their IT departments to be IoT/IT departments. Owners lack much skill and knowledge in the new IoT frontier. For these reasons, I have advocated better devices and ubiquitous UIs.

I will like to see expanded systems and system-of-systems IoT testing at some point, maybe into a separate eBook. In this short section, I introduce my current thinking, but I am actively researching and supporting this area of systems IoT.

In systems IoT, I recommend that integration test planning be a significant effort.

On the hardware side, I believe IoT systems will be built from the bottom up, with many integration cycles. On the software side, CI and CT will be good practice. The more complex a system or system-of-systems is, the more cycles, steps, and planning must happen.

In complex systems, dedicated integration and system test teams will likely appear. Such teams should consider:

1. "Good enough" definitions of many qualities with degrees of V&V/Testing

2. Active and evolved integration-test planning

note: in more complex system integration and test planning should be separate activities often done by different teams.

3. Integration-test team involvement with hardware, system, and software architectural efforts

4. Integration-system risk analysis formally and informally on an on-going basis

5. Having users be “testers” (see dev-ops section) with monitors, logs, and data analytics

6. Agile Testing

7. Classic old school V&V/IV&V/testing as needed for hardware, software, and systems

8. Advanced V&V/testing such as math-based, model-based, and experienced-based approaches

9. Support to the “external” organizations, which are likely to include vendors, third party providers, standards groups, IV&V, and customers

10. Communication to external organizations

11. See System Test planning and strategy sections

Large systems and systems-of-systems I have been involved in, often have problems that appear once integration happens, then later early use of the systems starts. We have all heard the stories of airplanes that fail during early test flights around the world; cars whose infotainment systems that are less than expected, and smart systems that get discarded:

see http://www.computerworld.com/article/2515483/enterprise-applications/epic-failures--11-infamous-software-bugs.html

see https://www.scientificamerican.com/article/pogue-5-most-embarrassing-software-bugs-in-history/

When the stories are told, I often hear: vendors point blame at each other, system providers sue vendors, and owners of smart systems scrap them and go to “new” providers. These are not stories of success. Vendors, prime contractors, and customers, who are willing to pay the (any) price to achieve the right “good enough” level of system quality will move forward into the future.

V&V/IV&V/testing will be part of the system and system-of-systems story. Vendors will need “good enough” V&V/testing. Prime system contractors will need yet more V&V/testing and in some cases IV&V. Finally, many customers/owners will need their own V&V/IV&V/testing.

Am I saying every mom and pop homeowner will need to contract with testers? No, but consumers will need to become much more educated about IoT devices. Most of us are familiar with the concept of the test drive of a car, inspecting your house before you close on it, or trying on cloths in the store before you take them home. The public will need to become more IoT “system” conscious when buy smart devices. Many of will develop “personal testing” concepts for smart devices just like people test drive a car.

Some groups (e.g., governments, companies, and other larger organizations) will need to develop or hire V&V/IV&V/test organization. These may be separate teams or be part of existing IT groups. I expect V&V/IV&V/test providers to evolve.

Finally, system test teams may find some specific V&V/test ideas/techniques in the list below (use the internet or eBook 4 to find specifics on these ideas):

1. Field Testing
2. Simulation analysis
3. Modeling
4. Prototype testing
5. Destructive testing
6. Electrical analysis including sneak circuit analysis
7. Burn in testing (long and hard usage)
8. Shake and bake V&V (vibe and heat)
9. Stress (physical) testing
10. Dog food test
11. Quality control in production

Planning the test environment

Think of creating a lab test environment as a project inside of a project?

My first book (See Software Test Attacks to Break Mobile and Embedded Devices) had extensive information on setting up a test environment for embedded and mobile devices. The concepts of that book are a good starting point for some and so will not be repeated here. However, here is an outline of the key points, including:

1. Developer level testing environment and tools

a. Static code analysis tooling

b. Structural unit test tooling

c. Unit code data generation tooling

d. Error or backlog tracking facility tooling

e. Connection to development environment tools (optional)

f. Interface with model based test tooling

g. Data visualization and analytic support tooling

2. Integration environment lab and tool

a. Hardware, internal and external connections

b. Software, internal and external connections

c. System/systems of systems connections

d. Timing and performance analysis tooling

e. Communication systems (Wi-Fi, Bluetooth, cellular, etc.)

f. Simulation and emulation to tooling

3. Functional and quality test labs

a. Hardware support

b. Software support

c. Internal visibility support

d. System support

e. Simulation and modeling support

f. Security sand box support

g. Quality area support (reliability, safety, hazards, performance, etc.)

h. Cloud support (virtual labs, devices, app testing, simulation, emulation, etc.)

4. Real world test labs and beds

a. Smart cities

b. Smart homes

c. Smart building

d. Smart companies

e. Smart offices

f. Smart people (medical)

g. Test in the real world with controls in place to keep testing information available

The bigger and/or more complex an IoT device or system of devices is, the more complex the lab environments will need to be. Here is a list of example labs I have already seen being used:

1. “Iron bird,” where the avionics systems of an airplane can be “flown” without ever leaving the ground through the use of extensive simulations where every environmental factor can be “induced” as inputs.

2. “Flight test bird 1,” which was an airplane, version 1, that could fly but whose only mission was to test the integrated system, and not to carry passengers or fight wars.

3. “Test car 1,” which drove around a “test lot” within a company’s facilities where there was real traffic patterns and roads.

4. “Manikin 1,” which was a “test dummy” that simulated heart and lung conditions so that things like heart attacks could be simulated.

5. “SIL” (system integration lab), where the components of devices were all present but not in a form for consumers so that new hardware and/or software could be “plugged in” and tested quickly.

6. “ Mount Evans Test Bed,” (link http://www.mountevans.com/MountEvansCom/Mount-Evans-Things-HighAltitudeAutoTestLab.HTML) which was a test lab in a test building where cars from a manufacturer could be brought and tested in the real world of the Colorado Rocky mountain areas since it offers many extreme environments to “stress” cars (e.g., snow, cold altitude, heat dirt roads, etc.)

7. “The fun lab,” which was a place where IoT toys were “played” with by workers and their kids.

8. “The Barcelona test city,” (see http://www.iotsworldcongress.com/the-iots-world-congress-will-showcase-the-potential-of-industrial-iot-through-10-testbeds/) which is a whole “smart city” test bench (I expect that I will see more of these in each country).

9. Chaos testing, in which the tests were done on production systems in the real world (Google chaos engineering or http://principlesofchaos.org/). I expect a lot of this kind of testing in IoT systems.

The nature of a lab depends on the IoT device. The examples and lists described above are intended to be starting points for thinking about specific IoT test facilities.

A driving factor in test labs, tools, and environments is cost and schedule. I have spent thousands and millions of dollars setting environments up. Such large ranges are driven by the nature, context and regulations of IoT devices. As I said at the beginning of this section, teams creating labs should think of them as a project inside of a project—and always remember to test your lab. Teams will need budget, schedule, development, support functions, and testing of test labs.

Start-ups and smaller projects may have labs for the IoT device, an interface computer, and a table sitting in some testers work area. This may be enough.

Bigger IoT efforts will need bigger environments including going into crowd testing. Many companies and testers will want or need to be involved.

Many test environments in IoT will be in the "cloud." This trend follows what I have seen in the mobile smart phone world. I expect the cloud to be used for both hardware and software support. Vendors and providers can help and will be ready to market these services to IoT teams. Some cloud services are already being advertised as this is written.

There is to say on test environments, so continue doing reading and research outside of the eBook.

Planning use of IoT tooling and automation

One of the first things managers want to see when "improving" testing is the idea of automation. This is because companies and industries for decades now have introduced automation in the form of computers, software, and even robots. This has resulted in productivity increase and money savings.

Truly, automation is a good idea, particularly for repeated tasks. There are repeated tasks in testing, e.g. manual executing the same test procedure over and over. If you are this kind of tester you need to be working on your skills to keep jobs.

Another good area of automation is for tasks which involve a concept where logic or an algorithm can be introduced. Here again testing has these also. There are activities in testing place that are complex, happen very quickly, or involve other tasks where humans can fail but computers can shine.

The list of generic tools below introduces classes if tools. Additionally reads can refer to my first book as generic test support tools, monitor systems, and oracles are covered in some detail.

Software Tools:

Comm Traffic Monitors and Network Sniffers - applications used to monitor the traffic in the interface, source/destination host addresses etc. These can be over the air (traffic monitor) or one the wire (network sniffers).

Security Probe Tools – software such as disassemblers which allow binary code to be "read" looking for security holes and bugs

Fuzz tools – Security tools which aid in fuzz testing

Combinatorial test tools – tools which implement combinatorial test attack data selection

Test execution automation – tools which aid test execution (see notes below)

Spoofing tools which security spoofing attacks

Note: there are many software life cycle support tools which address areas like management, planning, reporting, configuration management, error reporting and generic aspects of testing, but these are deemed by me to be out of scope for the eBook.

Hardware Tools:

Hardware in circuit controller- This is similar to a software debugger. This device allows control and variable step by step inputs to the hardware-software. These systems can be used in monitor mode (records CPU) or input mode (inject information into hardware under test). Many of these systems also support performance/timing analysis.

Oscilloscopes and electronic probes- These are used to check and record electronic/hardware events with time stamps including power supply and signal

Software drive radio - This is used to emulates receiver and transmitter function of radio frequency systems for a large range of wireless gateways.

Control and routing panels – These are usually custom build electronic system that allow the patching via cables of different hardware and even software configuration.

System Tools:

Field support tools – recorder and tracker systems which instrument the system to collect data, but do not interfere with its operation

Data analytics with tools

These tools analyze and reduce the data steams into information humans can understand.

There are other tool concepts under these categories and new tools are coming online constantly. My web site (http://breakingembeddedsoftware.com/ link), list specific test tools, but the list is often out of date as I update the site only about once a year, and the names and availability of tools seem to change weekly. A better site for information on test tools is www.stickyminds.com. This site has list of tools which is largely vendor maintained, so it tends to be more up to date.

Detailed Test Planning (what is needed below the master plans)

Below the master test plan are often levels of test planning details. These levels may be associated with a test cycle, a product release date, a large build release, or even a sprint (if Agile). These plans outline specific tests and are often called a test specification (See ISO 29119 part 3 for industry based document titles/outline examples). The plans are very detailed and subject to large changes as the Dev-test efforts unfold.

Projects test products at many levels and these tests are often included as part of development cycles, Quality Control (QC) in hardware manufacturing, and Quality Assurance (QA). Products are also tested as they are mass produced and this is often called QC. I do not address factory hardware QC production in this book. It is a big subject and will be important for mass production runs of IoT devices, but perhaps I will cover it in future eBooks.

In this section, I address some concepts and examples IoT teams many want to consider for detailed test types to plan.

Hardware detailed test patterns

Detailed hardware attack test/V&V patterns to consider at this low level include:

Circuit analysis
Sneak circuit analysis

Electric analysis
Battery demonstration
Mechanical system physical tests
Power tests
Radio signal tests
Color tests
Physical feel and packaging tests
Acoustic tests (drop, shock and vibe)
Temperature tests (bake, cold, hot, normal, stress, cycles)
Hardware usability
Long term wear tests
Quality checks (see ISO 9000 series)
Integrity V&V (see IEEE 1012)
Integration and interface
Review and inspections

Note: the details of these patterns and techniques are not defined in the eBook, but later eBooks, standards, and/or my first book. The readers should refer to these more information.

Software detailed test patterns

Detailed software attack test/V&V patterns to consider at this low level include:

Attacks in Software Test Attacks to Break Mobile and Embedded Devices
Software test techniques of ISO 29119-4
Software integrity checks of IEEE 1012
New attacks addressed in leanpub.com IoT eBook part 4
Reviews and inspections

System detailed test patterns

All of the list below is at the system (or system of system) level. Many of these features should have been tested during other testing, so reuse of tests may be possible, but here all the parts are "play" together, possibly for the first time. It is tempting to wait on some of these test activities until we have a system which is complete, but finding issues when a system is "complete" can cause waste by having to rework hardware, software, or ops. There is no one right mix of early and delayed testing. Skilled thinking and experience are needed to get an optimal mix, but even then only hindsight will be 20-20, not planning.

Critical system patterns/features to test include:

1) Usability:

Mandatory

ADA (Americans with Disabilities Act)/Disabilities

Failure and recovery modes

2) IoT Security:

Mandatory

3) Connectivity:

Full connect

Partial/limited connection

No connection (offline mode and restore)

Work with different vendors

Transfer rates

4) Performance testing:

Mandatory time based V&V

5) Compatibility and interoperability Testing:

Check to the architecture, hardware, software, third party vendors (OS, protocol, browser...), comm, etc.

Integration and interface testing/V&V

6) Pilot Field Testing:

Mandatory (Start in the lab, but the real word is full of surprises)

Who owns this testing (third party, IV&V, regulator, etc.)

Consider field sand box (limited alpha/beta in the field and chaos testing)

Field gets you ready for the release

7) Regulatory Testing

What regulations, standards, and/or legal issues exist which must be V&V'd

Is there independent assessment (IV&V, regulatory body, third party) which must passed (this is a risk)

Has the independent assessment be involved in the project (worked so there effort are successful)

8) Upgrade testing (hardware and/or software)

Push or Pull checks

What if an upgrade does not happen

Regression and new testing

Combination of multiple protocols, devices, operating systems, firmware, hardware, networking layers etc.

Planning individual tests (what testers should do daily)

I mention now the journey all testers should make into the world of test planning starting with planning the daily tasks and individual tests. Every tester, to some extent, should do this and build planning skills. True enough, the room for changes of plans for a particular test may seem limited, but many of the items listed in this eBook apply (e.g., how much time, what are the goals, how is testing to be done (scripted, exploratory, techniques), etc.).

When I started in testing, he was given a detailed plan, individual tests he had to engineer, and a top-level schedule (e.g., be done in a week). He started learning about planning using individual tests. This expanded, after some time, to detailed tests, and finally master test plans. Certainly, new skills and understanding of levels of detail were needed. He bought books, read standards, and reused information from other testers in his journey from standalone tester to test lead to test manager. His journey took years.

In the individual test planning, consider the following outline as a start:

1. Check email at beginning of day for any detailed or master plan changes
2. Estimate what design was needed for today's test
3. Conduct research of the test(s)
4. Produce test design (select methods, attacks, tours, techniques, etc.)
5. Create detailed test implementation (goals-requirements, input data, expected output, steps if any, scripts, etc.)
6. Conduct first pass exploration or ad hoc testing
7. Change design and/or test

8. Conduct test for credit

9. Gather outputs of test

10. Analyze test results for important information (pass, fails, bugs, lesson learned)

11. Report on testing

Now, some of you may release items into the real world of individual test planning and several of these steps may be done in parallel or in different order. Certainly, the parallel nature of how work gets done is a problem of the written world vs. how the human actually functions. Also, some may rebel at the mention of test "scripts." But some test efforts need documentation in some form and scripts may provide documentation.

The message here is that, as a tester, I had to start learning planning at some point. He did it with a modified form of daily personal software process (see http://www.sei.cmu.edu/reports/00tr022.pdf). Your example and outline should look different, but start improving your planning abilities initially small then build upwards to detailed, and then master plans. If you get good at the simplest levels of planning, then you can move on to harder levels of planning.

CH Test planning documentation, presentations, and proposals

The traditional side of systems and software engineering is often viewed as being "document centric." Earlier, I advocated that test planning documents do not need to be heavy weight (no large "thud factor involved). Agile has white boards, story boards, mind maps, and planning games. For many IoT devices, these can be sufficient documents. However, if you had a choice between IoT critical medical devices where all you knew about them was that for one, there had been planning on a white board while, another actually had test plan documents that had been reviewed/analyzed by experts, other testers, and regulators? Which would you "safer" in using?

There are IoT devices and systems in which stakeholders will want more documentation. When I enter a new project, I typically ask to see things like test plans. If they are years old, look like boiler plate generic words, and do not seem to be in use, then I feel like the project lacks levels of control and discipline. I dig deeper to understand what kind of testing this type of team is really doing. I would next look for presentations made on testing or quiz testers as to what they understood their jobs to be. If the answer were different than what plans said and not consistent with other testers, I tended to suspect the testing was out of control or not being done at all.

There is a fine line between too much useless documentation and useful documentation. Agile teams produce "just good enough" documentation to aid in communication within the teams and serve as a historical reference. I find that test documents help solidify the thinking of the planning process and then serves as a reference to stakeholders.

Another good guide to determining the right levels and types of test documentation is the stakeholder(s). If the project is small and only internal users existing white boards and some

smartphone pictures may be “good enough” documentation. If your stakeholder is the government, you will be subjected to audits, and there will be a long IoT device usage history, then more heavy weight (thud factored) documentation that is followed and updated may be advisable.

I have recommended the ISO 29119 part 3 software test documentation as a reference. However, this standard is very heavily weighted in terms of numbers of document types and the outline of material content. I recommend that everyone tailor to such a standard. In tailoring, a team should be able to get to the right “weight” level with stakeholder’s buy in.

Tailoring is not easy. It takes critical thinking and effort. It takes coordination and communication. This leads many teams and stakeholders to just say “we will do everything.” This is wasteful, creates documents that are not used, wastes project man hours, and worse than that--teams that do no documentation.

Getting the right weight balance of test documentation takes commitment. Like testing process and techniques, there is no “best” level and type of test documentation. Further, the levels and types of test documentation are likely to change over time, either increasing or decreasing. Again, being “agile” and thinking things through are needed.

CH Summary and what is next

Test planning and strategy may be a lead or management set of tasks and activities, but all testers should be able to do aspects of these “up front” testing actions. I have provided an introduction with pointers to more information on test planning.

As I said at the beginning, the test plan (a document) is nothing, but the planning is everything—this leads to success. Planning captures information that stakeholders need to grant approvals such that the testing provides the information of value to the team. Plans should address the goals, scope (inputs), strategies, basis, actions, and outcomes (possibly documentation) of testing.

To many test organizations, applying historic but simple strategies, such as verification checking of requirements, which they call “good,” often results in products that do not meet customer’s/user’s needs. Testing plans need to be comprehensive enough to address a master plan and then later, the detailed planning.

Further, planning must be agile and dynamic reacting to the information and data associated with products. Agile planning is needed even in more traditional approaches to development. Also, because testing is sampling, the incoming information and data should be fed into the updating of plans during phases from initial concepts (prototype) to final products (maintenance).

After this eBook, I will address the activities that come after planning such as test design, execution, reporting and environments. The foundation for these must be cohesive and work; organizations must do planning. While testing is never finished (100% complete with all things covered), it does stop when items (tasks and topics) defined in the planning are called “good enough.”

The definition of “good enough” for a project and testing is unique to each project, lifecycle stage of an IoT device. Test planning in part helps define the framework for “good enough” so that stakeholders can buy into the planning.

CH Appendix A: Skills for teams

Getting started – if you are new to test or management

This section provides help for those who are new to testing and just getting started in IoT.
Note: No list of skills would be complete in all areas and situations. What is provided in this appendix is just a starter.

People may get into IoT because of the number of opportunities and amounts of money in place (trillions of dollars). Many testers may come into software knowing little about testing and/or IoT devices. They may seek training and certifications, which are basic—but only starting points. They may read this eBook, a further start. However, past these beginning points, they should seek continuous experience and self–education in testing. Many companies feel you must have a degree, often in hard science, e.g., math or engineering, and while this certainly is one path into IoT testing, other education such as topics taught in liberal arts programs, can be just as good in basic education. However once a degree is obtained, the degree real means you can learn and maybe think a little. You must continue building skills and learning by practicing technical subjects. Practice makes perfect, as the old saying goes, but perfection is an illusion, so everyone is always practicing. Finally, some testers (and developers) are self–taught, and this can also be an effective start for the few who have the rigor and discipline to take this path.

Valuable knowledge leading to skill can come from:

1. Learning from the books in the reference section
2. Learning from online resources including Internet searches, on YouTube and others
3. Learning the basics of IoT devices and software such as on Wikipedia and About.com
4. Getting training from providers and conferences e.g., STAR, STPcon, CAST, and others
5. Engaging in testing discussions, attend forums e.g., LinkedIn.

Skill building will take a lifetime. Testers can also build experience by working with crowd source testing organizations such as, uTest, Mob4Hire, or others. Testing for crowd source organization is a great way to practice the skill of testing, and can lead to better employment.

CH Skills for testers (and all team members)

I have written about this before. Testing is a highly skilled profession. Just getting this eBook, any book, standard, or classes can give some basic knowledge. It does not make you a reasonable or even good tester. I have spent decades learning testing, and I am still learning. The skills of testing must be professionally practice over many years. For a list of skill software test areas that you should have or be working on, you can see the list and reference in Appendix A (AST skills list). I recommend that you get

an IoT device, maybe do some programming and start testing it. This is how people learn.
Note: the lists of these sections are not complete and subject to change over time.

1. Device to device production variation
2. Quality control in production and how it may impact testing data
3. Sampling of hardware in factor production (Quality control checks and 6-sigma)
4. Hardware configuration management
5. Inspection of hardware (visual)

a. Hardware checks
b. Electrical checks
c. Mechanical checks

6. Reliability testing of hardware (different life curve)
7. Availability testing
8. Burn in testing (long runs under stress)
9. Electronics testing

a. Circuits
b. Sneak circuit analysis
c. Test pin connections

Hardware engineer skills

One thing that makes IoT different is many of the devices have unique hardware. While you may consider yourself a software testing, those of us working with embedded and IoT devices quickly learn that many "hardware" skills are of benefit when doing software-system level testing in IoT. I find you cannot test the software in a void without aspect of the hardware. Beneficial hardware skills for a software tester can include:

Understanding basic electronics
Run and read an electronic scopes
Run and read tester devices for batteries, electronics
Run and read electronic signal generators (Software radio and hardwire)
Understand sensors
Understand actuators and controllers – Analog and Digital issues
Dealing with noise
Dealing with sampling rates and bites
Understand Mechanical system
Physical Environments – heat, cold, wet, altitude, vibe, salt, etc.
Packaging and transport of hardware

Software skills (but useful to testers and hardware staff too)

The following is a short and maybe critical list of skills for software developers covering from the architect, to designer, to coder, and support areas.

IoT Software/Hardware/System Architectural paradigms (KISS).
The physical environment and how it influences software behavior
College degree, boot camp or critical thinking.
Computer programming in core, high and low levels (e.g. machine code or assembly language)
Logical and structured critical thinking
Test/V&V abilities
Attention to detail
Imagination
Communication with teams and documentation
Representational state transfer (REST) or RESTful web knowledge
Math - Data analysist and statistics
Hardware engineering
System engineering
The role of standards, regulations, and legal issues in development

Operational testing skills

In Ops in addition to the previously address items, the skills need to support testing activities likely can include the following:

Data analytics including statistics, AI, and analytics
Understanding the historic and new user cases
Social sciences (culture, people, nationalities, etc.)
Psychology
Marketing and sales
Management
Dev, test, and support engineering

Appendix B: How to grow your test planning skills

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

Learn the Agile planning game
Record your hours spent on detailed test task (see TSP and PSP – Watts Humphry)
Estimate your plans (cost, schedule, strategies, design, etc.) on a detailed test task
Record your hours spent on detailed test task and compare to your estimate (refine)
Have a career plan, near term and lifelong
Work on skill areas (see Appendix A) and grow
Practice, Practice, Practice

Have some fun!

References

The following books and references were used in preparing this eBook and/or are of good general reading. Further, for IoT software systems, I 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 in the AST community) should not be done and is ill advised. However, I 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 may be limiting, when we do not need limits.

Software Test Attacks to Break Mobile and Embedded Devices - Hagar

Domain Testing Workbook – Kaner etal

Agile Testing: A Practical Guide for Testers and Agile Teams – Crispen and Gregory

Experiences of Test Automation – Graham

Works by James Bach – do a Google search

ISTQB syllabus (download for the web) - http://www.istqb.org

ISO/IEC/IEEE 29119 software test standard – must buy from ISO/IEEE web site

IEEE 1012 Verification and Validation standard – must buy from IEEE web site

Merriam-Webster Dictionary online version

Glossary

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

SEvoc — http://pascal.computer.org/sev_display/index.action

IEEE — ISO/IEC/IEEE 24765:2010 Systems and software engineering (Vocabulary)

Definitions in the How to Break book series by James Whittaker

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 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 or define terms.

Disclaimer: It should be noted that, in general, the software industry and software testing often uses terms slightly different or uses different words that mean the same thing as another term. I do not seek to solve this problem, but I do acknowledge it. I provide a set of definitions for my series of eBooks that help readers know how I am using a word in context within the eBook and not that my usage is universally right.

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

There are many sub forms of analysis that teams should be aware of:
Formal analysis (verification)
-Quality control (QC) of production lines in manufacturing
-Math modeling
-Simulation
- Others – See Modelling

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 under gone 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 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 higher level of risk and criticality.

Inspection – is the visual or human assessment of a product. Inspection is used often for hardware, for example looking for workmanship defect during receiving or before a final delivery (thinks 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 larger variation between humans doing assessments. Use of inspection should be limited to those areas where other concepts can NOT be employed.

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 (reference: George Box)

Test – exploration and experiments intended to provide information about the qualities (negative or positive) about a product.

Quality – 1) Value that someone is willing to pay for. 2) attributes of a product,

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 test environment. It is place to "play" with testing.