Internet of Everything – Test Strategy

Contents

Part I – The IoE Testing Challenge

Introduction

In the first two articles of this series, I set the scene of the IoE, considered its still-evolving architecture and speculated on the emerging risks of it [1, 2]. These articles describe the scale of the challenge and introduce a seven-layer architecture that might help you to understand the features that must exist to make the IoE happen. The second article also sets out, at a high level, some of the risks that we must address.

This is the third instalment – on IoE Test Strategy. At the outset, I have to tell you that figuring out how we test what I will now suggest will be every Internet-connected system component that exists now and forever has proved to be quite difficult! In this article, I won't solve all of the testing challenges involved, but what I can perhaps do is suggest some of the dimensions, some of the influences, and some of the opportunities of the IoE testing problem.

I can only paint a picture of what I believe we will have to test for the next ten to fifteen years and will suggest what testers need to think about and much more tentatively, how they might think – because the challenge we face is quite different from what we have tested before.

Some of the statements I make in this paper derive from a recent trip to Barcelona. I was a visitor to the Mobile World Congress [3] and learned a lot from the people I met there. I was one of 93,000 visitors, apparently.

Scope of the Internet of Everything

When I embarked on this article series, I thought, like many people still do, that the IoT was mostly devices at the 'edge of the internet', but over the course of the last twelve months or so, the scope of what is increasingly called the Internet of Everything has broadened. Some parts of the IT landscape have been brought under the IoE umbrella and quite a few more will be included in time.

The IoE includes the vast number of static devices, mostly sensors, but also mobile devices including cars, trucks, buses, trains, planes, ships and even satellites. It will come to include both every-day and obscure objects that we encounter at home, in cities, our work environment, hospitals and entertainment venues. Naturally, it includes wearable and even human-embedded devices, mobile phones and tablet computers too. And, after all that is taken into account, much of the data that is captured by these devices will be stored, processed and analysed by cloud-based servers that integrate with other cloud-based services, partners, payment systems and other popular services and the legacy systems of large companies.

I expect that the Internet of Everything really will become an apt description. In fact, *in time, the Internet will become shorthand for the IoE.*

With this seeming inevitability in mind, the test strategy for the IoE really does become the test strategy for everything on the internet.

The Biggest Thing in 30 Years?

The IoE is the most exciting change in our industry since client/server came on the scene in the mid/late 1980s.

Client/server was important because the Internet, Web services, mobile computing are all essentially client/server implementations. Some would argue that Object-Oriented analysis, design and development are significant, but I think these affect only programmers and designers.

Agile is significant – but I think it is an approach, not a technology and anyway, it is transitional. Continuous delivery and DevOps are bringing factory automation processes into software and perhaps are the most appropriate approach for many mobile and IoE implementations. After Waterfall and Agile, Continuous Delivery and DevOps are morphing into 'the third way'.

The IoE, if it develops in the way that some forecasters predict, will affect everyone on the planet. Estimates of the number of connected devices on the IoE range from 50 billion devices to 700 billion devices over the next 5 to 20 years. No one knows of course, but the expectation is that we are on a journey that will increase the scale of the internet by one hundred times. This will take some testing! But how on earth will we approach this challenge?

The Latest Step in the Testing Journey

There have been some quantum leaps in the complexity of our systems and the IoE is the next step on our journey. Let me recount the history of our software testing challenge. I know testing started before we had 'green screens', but it is a convenient starting point.

1. Screen based applications running on dumb terminals were the norm up to the mid-1980s or so (and of course are still present in many companies). Each screen typically had a single entry point and a single exit point, the content of the screen and data input was limited to text. Screens might be relatively simple – but systems had many screens.

2. With the advent of GUI applications, it became possible to have many windows active at the same time. There were many ways in and ways out. The content of screens could now include graphics, sounds and video. Windows applications had to deal with events – triggered by the user clicking anywhere onscreen or from other windows or applications resident in the GUI environment.

3. At the same time as GUIs arrived, client/server became the architecture of choice for most applications. The scope of an application was no longer limited to workstations but might involve the collaboration of many servers connected with (often flaky) middleware. Performance and reliability become more prominent risks.

4. The emergence of the internet, although hailed as a revolution is really an instance of client/server. What was different though, was the exposure of

private networks, functionality and data to the public internet. Security and privacy become much bigger concerns.

5. We are in the thick of the 'Mobile Revolution' right now. The democratisation of software means applications are available anytime and anywhere on varied devices and configurations in numbers far beyond our ability to test comprehensively. The proliferation of apps and devices and our enthusiasm for this expansion knows no bounds.
6. And now, on top of all of the technologies above, we expect that the network of connected 'things' will increase the scale of the internet by perhaps 100 times. These devices range in sophistication from 50 cent components to buildings, ships, cars, aeroplanes and cities. And that's the point - IoE brings all of these plus 'all of the above' with the full range of sophistication from devices costing a few pennies to the infrastructure of a city that might cost billions.

The IoE brings new levels of complexity and scale. The non-functional risks are reasonably well-known and we know how to address them. *What is new is the need to do functional testing and simulation at scale.*

Some Other Considerations

CMOs May Hold the Budget for IoE

There seems to be a shift in power, at least in the mobile space (which accounts for a significant proportion of the IoE market). It appears that for mobile businesses, the budget for building mobile and IoE systems may be held by marketers. If CMOs hold the purse strings, the justification for IT will be driven by the need to meet both short and long term business goals. Development must align with the need to experiment and learn.

To do this, IT will have to be agile and most likely adopt a continuous or DevOps approach. Delivery will be based on experimentation to drive the rapid evolution of products and services to meet rapidly changing requirements. DevOps, continuous delivery, high levels of automation, simulation, experimentation and test analytics matched with business and production analytics in some blend will be required.

Connecting the Unconnected

The challenge for the network operators around the world and the service providers wishing to gain customers is to connect 'the second half of the world'. In developing economies, it is likely that mobile internet rather than cabled links will be used to connect widely separated communities. Mobile phones took off when the cost of handsets dropped below $40. Perhaps when smartphones are available at that price, the mobile internet will take off too.

The goal of 'Internet.org by Facebook' [4] is to provide cheaper access to data by (to perhaps 1/100th of the current price) using network extension technologies and unused or white space spectrum [5] and to reduce the volumes of data used through local caching and compression technologies. Connecting the unconnected is happening on a global scale, and a country by country basis and gathering pace. IoE implementations in emerging internet economies may have quite different approaches to local networking than the developed economies.

Recently, the Colombian president announced that every citizen of Colombia now has their own email address. Colombia is an examples of a country where there are 'many connected, many unconnected'. The internet.org service was recently launched in Colombia [6].

The acceptable cost for mobile internet based IoE is very low. For example, the cost of smart meters monitoring energy usage in domestic homes needs to cost no more than a penny per device per month. In the developed economies the costs of mobile internet is still rather too high, but it is coming down.

The constraints of local mobile connectivity in the developing economies will drive the development of cheaper technologies. The developing economies will probably lead the way in implementing the IoE with mobile internet.

What networking technology/protocols will be used to implement IoT local connectivity? One speaker at the MWC suggested that, "No technology dominates yet. We are waiting for technologies to improve in range and performance. The standard technology for connecting the IoT hasn't been invented yet."

We are still at the start of the IoE journey.

Analytics and the People Who Use Them

Most of the data being captured by sensors is time series data, that is, the history of some measurement is captured and stored for a device over a period of time. The device might be fixed or mobile, but ultimately, the data it collects reflects a measurement captured over time. Analytics tools have not been great at integrating and correlating measurements that share different, possibly random capture frequencies. Depending how you cluster and average data, different patterns might emerge.

The data captured ultimately represents some physical attribute or measurement. Capturing data and obtaining statistics is one challenge, but to get value out of the statistics you need also what might be called physics-based modelling in order to get the 'deep insight' and control.

Big Data has been around for a few years now, but the use of analytics or data-science to derive insights from data is still evolving. The discipline of data-science is not yet well-defined and the availability of skilled data-scientists is generally low, but improving. At the same time, however, the data skills of the people who would use the analytics and visualisations are still lacking. Data-driven management as a discipline is still embryonic. One MWC speaker suggested, "You can have the best insight, but if people haven't been trained they'll do what they know, rather than use the data".

The Scope of IoE Testing

The range of concerns that the IoE brings is wider than ever before. Of course, not all IoE systems will be huge, complex and expensive, but most will bring a new technology and risk profile. The approaches we will need to tackle the risks of failure will have to change. Here are my suggested 'dimensions' of the technology and testing problem. This isn't very likely to be the last word.

Scale: The obvious first challenge is scale of course, but that is primarily a logistical problem for implementers. Of course, there will be the scalability challenges at various levels of the architecture and we'll have to do some large-scale load, performance and stress testing.

Hardware-Level functionality: the lowest level devices are sophisticated, but essentially perform simple functions like sensing the value of something or changing the setting, position or speed of a machine. These devices are packaged into objects that will need testing in isolation but most of this will be performed by manufacturers.

Object and Server level functionality: The vast majority of functionality that needs testing will reside on local hubs and aggregators and data-centre-based server infrastructure. Internet-based and native mobile apps will deliver the data, visualisations and control over other aspects of the architecture. Architectures will range from simple web-apps to systems with ten, twenty or more complex sub-systems.

Mobile objects: testing static objects is one thing, but testing objects that move is another. Mobile objects move in and out of the range of networks; they roam across networks. The environmental conditions at different locations vary and may affect the functionality of the object itself. Our sources of data and the data itself will be affected by the location and movement of devices. Mobile devices will drift into and out of our network range, but also drift into and out of other networks, not necessarily friendly ones. Power, interference, network strength, roaming and jamming issues will all have an effect.

Moving networks: Some objects move and carry with them their own local network. A network that moves will encounter other networks that interfere or may introduce a rogue or insecure network into range and pose security problems. Cars, buses, trains, aeroplanes, ships, shopping trolleys, trash trucks, hot-dog stalls, tractors – almost anything that moves – might carry with them their own networks and bridge to and collaborate with 'friendly' networks as they encounter them. But they must also block foreign and/or unfriendly networks too.

Network security risks at multiple levels: Rogue devices that enter your network coverage area might eavesdrop or inject fake data. Rogue access points might hijack your users' connections and data. Vulnerable points at all levels in your architecture are prone to attack. Networks will need to be hardened and tested.

Device registration, provisioning, failure and security: Devices may be fixed in location or mobile but the initial registration and provisioning (configuration) are likely to be automatic. More complicated scenarios arise where devices move in and out of range of a network or transition between networks. Needless to say, low-power devices fixed in perhaps remote locations are prone to power failures, snow, heat, cold, vandals, animals, thieves and so on. Power-down, power-up and automated authentication, configuration and registration processes will need to be tested.

Collaboration confusion: Mobile, moving devices will collaborate with fixed devices and each other in more and more complex ways and in large numbers. For example, in a so-called Smart City, cars may collaborate as a crowd to decide optimum routes so every car gets to its destination efficiently. But, however these resources are controlled, accidents happen, drivers change their mind, car park spaces will become available and unavailable randomly and so the optimisation algorithm must cope with rapidly changing situations. At the same time, these services must not confuse public services, commercial vehicle drivers, private car drivers and passengers. Managing the expectations of users for all systems that collaborate dynamically will be a particular challenge.

Integration at all levels: Integration of physical devices or software components will exist at every level. Integration of data will encompass the flows of data that is correctly filtered, validated, queued, transmitted and accepted appropriately. Many IoE devices will be sensors chirping a few bytes of data periodically, but many will also be software components, servers, actuators, switches, monitors, trip-ups, heaters, lifts, cars, planes and factory machinery. The consistency, timeliness, reliability and of course safety of control functions will be a major consideration. Industry standards in application domains that are safety-related are likely to have a role. However, for now, much of the legislation and standardisation that will be required does not yet exist.

Big Data – logistics: Needless to say, much of the data collected by devices will end up in a database somewhere. Some data will be transactional, but most of it will be collected by sensors in remote locations or connected to moving objects. A medium sized factory might collect as much as a Terabyte (1,000,000,000,000 bytes) per day. Performance and reliability requirements might mean this data must be duplicated several times over. Wherever it is stored (and for however long) a very substantial data storage service will be part of the system to be tested.

Big-Data – Analysis and visualisation: Analyses of data will not be limited simply to tabulated reports. The disciplines of data science and visualisation are advancing rapidly, but these rely on timely, accurate and consistent data; they rely on data acquisition, filtering, merging, integration and reconciliations of data from many sources, many of which will never be under your control. Often, data will be sparse or collected infrequently or at random times. Data will need to be statistically significant, smoothed, extrapolated and analysed with confidence.

Personal and corporate privacy: The privacy of data – what is captured, what is transmitted, what is shared with trusted or unknown 3rd parties or stolen by crooks and misused is probably peoples' most pressing concern (and this is also a barrier to exploitation of the IoE). The current legal framework (e.g. the Data Protection and privacy laws in the UK) may not be sufficient to protect our personal or corporate privacy. Hackers and crooks are one threat, and central government tracks and listens to them. But they tend to listen to all citizens, not just suspects. Your own government may be seen to be a villain in this unfolding story.

Wearables and Embedded: Right now, there is a heavy focus on wearable devices such as training and activity trackers and heart monitors that connect with apps on mobile phones. Your health data might be integrated with Google Maps to show training routes, for example. Other devices such as smart watches, clothing and virtual reality headsets are available now. But there are increasing numbers of applications where the device is not worn, but are actually embedded in your body.

Healing chips, cyber pills, implanted birth control, smart dust and the 'verified self' (used to ID every human on the planet) are all being field tested. It will be hard not to call human beings 'things' on the internet before too long. Will we need to hire thousands of testers with devices embedded in their body? Surely not. But testing won't be as simple firing off a few messages to servers using the tools we have today. Something much more sophisticated will be required.

Everything connected: The time will come when all of the devices used in a hospital, hotels and factories for example will be tagged or connected. Hospital staff spend a lot of time finding and moving equipment from place to place to hook up to patients and locating equipment faster will save time and could save lives. A less critical application might be in a hotel where every piece of cutlery, cup, plate, dish, glass is tagged and located. How many fewer of these need be bought by a hotel if the location of each is known? The same argument applies to tools, equipment and robots in factories and of course the people who use and maintain them.

The range of issues we need to consider in testing the IoE has increased and the scale of the testing required has increased too.

Part II - Test Strategy for the IoE

The way we have tested in the past, will work for some of the IoE applications of the future. But for many systems, the number of dependent variables that influence their behaviour increases dramatically. The nature of Big Data (think volume, velocity, variety and veracity), merging and integration of multiple data sets means that the range of analyses and possible insights derived are only limited by our imagination.

The change in thinking required to test large IoE implementations is significant. Let's explore some of the gaps in our abilities and how we might approach the challenge.

Modelling, Testing and Test Data

We have some good techniques already to model technical architectures, and tools do exist (used mostly in the systems engineering space) to capture models and to generate code and a covering set of tests automatically. But they tend to be based either on the technical architecture or domain-specific use-cases.

There are very sophisticated real-time modelling and simulation tools used in the automotive and aerospace industries, for example. These model the physical components of cars and planes to enable a large amount of testing to be done without costly full-scale prototypes (which might be tested to destruction). There are proprietary (and even some open source tools) to simulate traffic flows in cities, the movement of people in airports and of course aeroplanes in the skies.

Sophisticated, expensive simulation tools might be used by pilots, astronauts and Formula 1 drivers. Much better known are the simulators that are used by gamers of course. It is these that might provide the basis of usable tools to simulate large-scale collaboration and scenario testing. The software used to provide a car-chase experience could be used to model traffic flows. Shoot 'em up games could be adapted to simulate the exchange of data or payments rather than bullets, and the social engineering games adapted to simulate the exchange of labour, goods and services. And so on.

But these adaptations are probably not high on many people's agenda. Only large companies like Google, the car and aeroplane manufacturers, suppliers of power generation and distribution, signalling, traffic control and of those who supply to the military can afford to buy or build such tools. These products are not likely to be available to start-up companies – at least for the next few years.

The best you can do is to run manual simulations supported by tools that can repeatedly generate scenarios to be tested, record the outcomes and replay the simulations for later study perhaps. Needless to say, you need to incorporate test-hooks or other features in your systems and build utilities that will help you to easily inject data, capture and reproduce or replay scenarios.

Cem Kaner has written quite a lot about what he calls 'High Volume Test Automation' [7]. This is a good starting point.

Because the amount of activity and data these systems require and generate, you will be using Big Data test techniques. You'll need to find data that matches your test requirements; you will need to generate, tag, edit and seed data so you can trace its usage; you will need tools to monitor the use of tagged data and the ability to reconcile data from collection, storage, use and disposal. It will be helpful to use the visualisations created in your system for end-users and to enhance them for debugging and diagnostics.

One aspect of test generation worth thinking about is the concept of 'pattern-based test design'. The goal of this technique is to generate tests from known data to simulate particular scenarios that can be characterised as a pattern. Suppose we are testing a self-driving car (Google, are you listening?)

On British roads, where we drive on the left (or should do), cars that turn right and cross the path of oncoming traffic cause a large number of accidents. So an obvious pattern to test would be 'cars turning right that cross oncoming traffic'. If we have a selection of routes with right turns, we could generate a large set of very short journeys that include a right turn. Then we could overlay a second set of journeys that intersect at the junction (to intersect at the same moment) and for each intersecting pair or journeys, run our simulation.

What is the expected result? Well, we would be interested in seeing outcomes where one car or the other is delayed by more than a few seconds perhaps (indecision?) We would look for outcomes where the two cars appear to occupy the same geolocation at the same time (crash?) and so on. We could run these tests at varying velocities for both the turning and the oncoming cars at many locations across a busy city.

We could, for either car, change the persona of the driver from the automaton to a teenager or retired schoolteacher or change the car itself to a van, lorry or bus and so on. Obviously, we could run simulations with thousands of cars on intersecting journeys simultaneously. Of course, this is a test that we would run eventually – *but we would probably not start with that test*, as it would be an almost impossible task to debug the failures we would expect to see.

(Given the personas of some drivers I've encountered over the years, I am looking forward to seeing how a Google or other self-driving car copes with Central London traffic or that in Bangalore! Or maybe not.)

At any rate, the goal of pattern-based test design is to feed your automated tools hundreds or thousands of scenarios to simulate. Part of the brief for your developers must be to include features or hooks to facilitate High Volume Automated Testing.

Test Environments; Testing in the Field

Requirements for the scope of test environments for IoE implementations will vary widely. In a simple case, a test lab for a home environment management product could be set up in any office as the scope of the local network is confined to a single household.

In the case of an urban environmental management system that monitors air pollution levels across a city for example, the sensor data capture could be simulated in a lab, but you would expect to have to pilot the service in a real city environment to calibrate the sensors, data aggregation and integration processes and have meaningful data visualisations.

The scale of the field testing of a service that is to be marketed to home owners or the authorities of a large city will vary widely. Field testing to demonstrate a system can accommodate the vagaries of the weather, traffic or the general population to obtain the confidence of stakeholders will become a common (and possibly mandatory) testing stage.

Barcelona has for a long time prided itself in being an innovative city. Recently, the city authorities have established an urban-scale testing facility. The Barcelona Urban Lab [8] has been set up as a test environment for innovative projects that meet the needs (and some other conditions) of the city. Approved projects may use the existing city infrastructure to pilot innovative systems that provide new and useful services to the city. The Urban Lab is specifically positioned as a field test environment to prove the viability of innovative systems.

It is expected that other cities and local authorities around the world will offer similar services and the development of Smart Cities will depend on partnerships between system developers and civic authorities.

Tool Support

There can be no doubt that automated support for testing non-trivial IoE implementations will be essential. A lot of manual and/or ad-hoc testing by developers and testers working closely will be performed, but automated testing will probably dominate. Many of the software components of an IoE infrastructure will provide access to their functionality through APIs or services delivered by web or other emerging messaging protocols so testing with tools should be eased.

Now, these tools may be COTS products or open source but testers may have to create their own test drivers. The technologies used at the local network level (connecting devices/objects to local integrator services) vary and although there are IP- and HTTP-based technologies in use, for which there are commercial and open source tools, there are quite a few other services that are not well supported by tools yet and custom drivers will be necessary.

The question then arises as to the nature of the tests to be run by these tools. In some cases, the number of tests to be run might be quite small. For example, many mobile apps are quite limited in function and so a combination of manual testing and a number of automated tests through web services or through the GUI of the app itself will suffice.

But there will be circumstances where the number of tests required to demonstrate the functionality and resilience of services will be extremely high. Where there is a high volume of data captured from distributed locations and the patterns of that data are used to make real-time decisions, it would be difficult to test thoroughly without tools and a known set of data (whether real or synthetic) that can be re-used to run series of tests.

It seems inevitable to me that although there will always be a need and opportunity to do manual testing, a much larger proportion of testing will have to be performed by tools than we are currently used to. The tools will need to execute very large numbers of tests. The challenge is not that we need tools to execute tests. *The challenge will be, "how do we design the hundreds, thousands or millions of tests that we need to feed the tools?"*

So for example, suppose we need to test a smart city system that tracks the movement of vehicles and the usage of car parking in a town. We'll need a way of simulating the legal movements of cars around the town along roads that contain other cars that are following their own journey. The cars must be advised of their nearest car space and the test system must direct cars to use the spaces that they have been advised of. We could speculate on some heuristics that guide our test system to place cars where their owners want them. But.... life happens and our simulation might not reflect the vagaries of the weather, pedestrians, drivers and the chaotic nature of traffic in cities.

Not every system will be as complex as this. But the devices now being field tested in homes, hospitals and public places all have their nuances, complications and will experience unexpected events. Even a 'simple' healthcare application that warns a patient of forthcoming doctor's appointments, schedules prescribed drugs and monitors your symptoms or vital signs could trigger hundreds or thousands of scenarios.

Increasingly, small simple systems will interact with other systems and become more complex entities that need testing. It's only going one way.

Test Analytics, Visualisation and Decision-Making

I have written about Test Analytics before [9].

Increasingly, practitioners are looking for automated ways to do things. Behaviour–Driven Development connects requirements, test code and code itself by generating and executing feature-level tests. Automated build and test processes create event and test outcome data that potentially can be used to trace progress, coverage and trends in your test results.

Think of your DevOps processes as 'things' that support the evolution, maintenance and smooth running of your production systems. These processes (or at least the data they produce) are part of the system deliverable itself. These automated processes exercise your systems and monitor their vital signs just like sensors. In your production systems, the analytics provide valuable usage and market information. This same instrumentation can inform your development processes in a similar way and statistics from your DevOps processes can be designed to tell you how your development process is working.

If marketers are in the driving seat, and experimentation in production is an explicit goal of a system, then developers will quickly become expert in instrumenting components to capture the required analytics. Of course analytics capture data to track user behaviour, but it will be easy to extend that to capture other data of interest to testers and developers.

If instrumentation can be enhanced to capture data that self-checks or reconciles selected outcomes, then you could envisage having a sensor-network on your production code. The analytics so derived can be set up as 'trip-wires' that signal problems in production. Of course, this same code can operate in your DevOps environment and give testers an indication of the health of your system in the test lab.

If your production systems are platforms for experimentation, then the code that serves marketers can also serve developers and testers. Think of your analytics code as your sensor network. DevOps processes are things, too.

Performance Testing and Test Data

Network protocols that are IP based, for example XMPP [10] and MQTT [11] might be used as a local aggregators or as central hubs working at high volumes. There are few dedicated tools to support the simulation of loads but custom drivers/utilities are quite straightforward to create. Also, cloud-based services are becoming available allowing developers to outsource these aspects of their architectures. We've built an MQTT broker and MongoDB service ourselves [12].

Other Low-Power and Lossy Network (LLN) protocols such as ZigBee [13], 6LowPAN [14], DASH7/RFID [15], Bluetooth [16] and NFC [17] only work at the local level. There are few tools that support load testing of these protocols, but if you do need to simulate hundreds or thousands of remove sensors and objects, then crafting an effective driver to plug into existing performance toolkits should not be so hard. After all, the drivers might only need to reproduce the regular chirps of sensors.

What makes life harder is when the data that is captured by sensors must be coherent. For example, if you need to chirp location data for a car, a random set of location co-ordinates will simply not do. Cars do not normally zip around randomly (and at very high speed). So your driver may have to use a traffic route using the Google Maps API and the periodic chirps of data calculated to simulate a real car travelling at a reasonable, defined speed along the route.

If you need to track the movement of devices carried by people in a shopping mall, then perhaps you will have to capture the GPS data using a tool. I'm currently experimenting with a tool called GPS Logger for Android [18] to do just this. To create a variety of unique paths, you could edit together segments of intersecting paths, for example. Of course, if your app collects data such as this, you could use an early version to collect some test data for you. Once in production, you will have a plentiful supply of data to use in testing (when appropriately anonymized).

New Model Testing

Why do we need a 'New Model for Testing'?

The current perspectives, styles or schools of testing will not accommodate emerging approaches to software development such as continuous delivery and, the new technologies such as Big Data, the Internet of Things and pervasive computing. These approaches require new test strategies, approaches and thinking. Our existing models of testing (staged, scripted, exploratory, agile, interventionist) are mostly implementations of testing in specific contexts.

Our existing models of testing are not fit for purpose – they are inconsistent, controversial, partial, often proprietary and stuck in the past. They are not going to support us in the rapidly emerging technologies and approaches of the IoE.

I have proposed an underlying model of testing that is context-neutral and I have tried to shed some light on what this might be by postulating the Test Axioms, for example [19, 20]. The Axioms are an attempt to identify a set of rules or principles that govern all testing. Some testers who have used them think they work well. They don't change the world, they just represent a set of things to think about – that's all. But, if you choose them to be true, then you can avoid the quagmire of debates about scripted versus unscripted testing, the merits and demerits of (current) certifications or the value of testing and so on.

The New Model for Testing [21] is an extension to this thinking. The model represents the thought-processes that I believe are going on in my own head when I explore and test. You might recognise them and by doing so, gain a better insight into how you test too. I hope so. As George Box said, 'essentially, all models are wrong, but some are useful'. This model might be wrong, but you might find it useful.

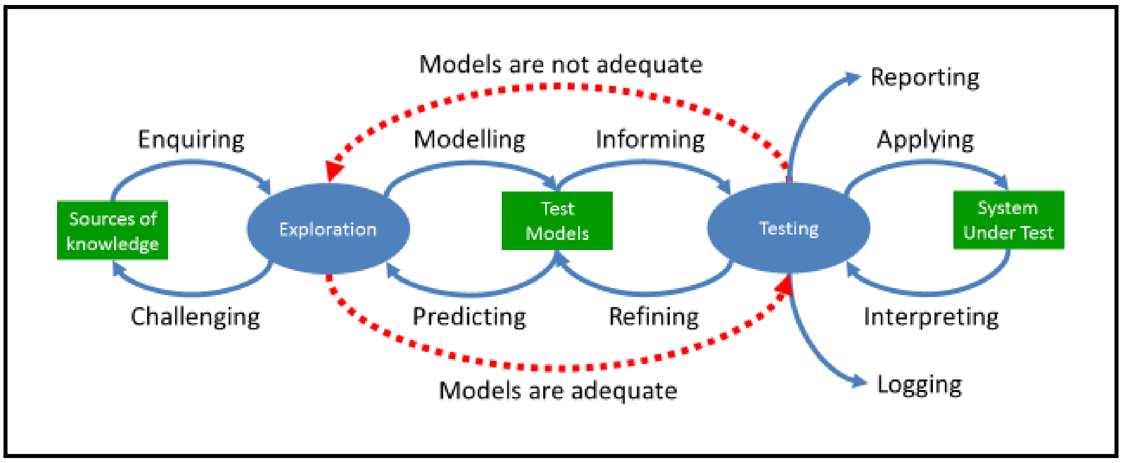
The New Model of Testing attempts to model how testers think and you can see a full description of the model, the thinking behind it and some consequences here: http://dev.sp.qa/download/newModel

Ignore Test Logistics

When tests are performed on-the-fly, based on mental models, your thought processes are not visible to others; the thinking might take seconds or minutes. At the other extreme, complex systems might have thousands of things to test in precise sequence, in complicated, expensive, distributed technical environments with the collaboration of many testers, technicians and tool-support, taking weeks or months to plan and apply.

Depending on the approach used, very little might be written down or large volumes of documentation might be created. I call the environmental challenges and documentary aspect 'test logistics'. The environmental situation and documentation approach is a logistical, not a testing challenge. The scale and complexity of test logistics can vary dramatically. But the essential thought processes of testing are the same in all environments.

So, for the purpose of the model, I ignore test logistics. Imagine, that the tester has a perfect memory and can perform all of the design and preparation in their head. Assume that all of the necessary environmental and data preparations for testing have been done, magically. Now, we can focus on the core thought processes and activities of testing. Here is the model:



The model assumes an idealised situation (like all models do), but it enables us to understand more clearly about what testers need to think about.

At the most fundamental level, all testing can be described in this way:

1. We identify and explore sources of knowledge to build test models
2. We use these models to challenge and validate the sources of knowledge
3. We use these models to inform (development and) testing.

I make a distinction between exploration and testing. The main difference from the common view is that I will use the term *Exploration* to mean the elicitation of knowledge about the system to be tested from *sources of knowledge*.

By excluding the logistical activities from the New Model, then the processes can be both simplified and possibly regarded as universal. By this means, perhaps the core testing skills of developers and testers might coalesce. Testing logistics skills would naturally vary across organisations, but the core testing skills should be the same.

From the descriptions of the activities in the exploration and testing processes, it is clear that the skills required to perform them are somewhat different from the traditional view of testing as a staged activity performed exclusively by independent test teams. Perhaps the New Model suggests a different skills framework. I have put together a *highly speculative list of skills that might be required*.

I hope the model stimulates new thinking and discussion in this field.

The Future for Testing and Testers

My suggestions below are partly informed by what I have seen in the technology and the testing markets and friends and colleagues who have shared their experiences with me.

Manual v Automated Testing?

It seems to me that high levels of test automation are bound to be required to make the IoE a reality. *Automation will not make testing easy; it will make testing possible.*

Web services, as an example, might be simple to test, but can also be devilishly complex but we have to use some form of driver to test them. The distinction of 'manual' and 'automated' testing in terms of ease, effectiveness and value is fatuous – we have no choice in the matter. What I have called the 'application of tests' in the New Model may be performed by tools or people, but ONLY people can interpret the outcomes of a test. That is all there is to it.

But I should add that more and more, test automation in a DevOps environment is seen as a source of data for analysis just like production systems so analysis techniques and tools are another growing area. I wrote a paper that introduces 'Test analytics' here [9]. This is a field that testers should pay attention to, I think. Given the shortage of data scientists out there, it is a field that might pay well.

Should testers learn how to write code?

I have a simple answer – yes.

Now it is possible that your job does not require it. But the trend in the US and Europe is for job ads to specify coding skills and other technical capabilities. More and more, you will be required to write your own utilities, download, configure and adapt open source tools or create automated tests or have more informed conversations with technical people – developers. New technical skills do not subtract from your knowledge, they only add to it. Adding technical skills to your repertoire is always a positive thing to do.

If you are asked to take a programming or data analytics course, take that opportunity. If no one is asking you to acquire technical skills, then suggest it to your boss and volunteer.

Shift-Left

It seems like every company is pursuing what is commonly called a 'shift-Left' approach. It could be that test teams and testers are removed from projects and developers pick up the testing role. Perhaps testers (at least the 'good' ones) are being embedded in the development teams. Perhaps Testers are morphing into business or systems analysts. At any rate, what is happening is that the activities, or rather, the *thinking activities* of testing are being moved earlier in the development process.

This is entirely in line with what testing leaders have advocated for more than thirty years. Testers need to learn how to 'let go' of testing. Testing is not a role or stage in projects; it is an activity. *Shift-left is a shift in thinking, not people*. I suggest that testers view this as an opportunity, rather than a threat.

'Test early, test often' used to be a mantra that no one followed. It now seems to be flavour of the month. My advice is don't resist it. Embrace it. Look for opportunities to contribute to your teams' productivity by doing your testing thinking earlier. The left hand side of the New Model identifies these activities for you (enquiring, modelling, challenging and so on). If your test team is being disbanded, look at it as an opportunity to move your skills to the left.

There is something of a bandwagon for technical people advocating continuous delivery, DevOps and testing/experimenting in production. It seems hard for testers to fit into this new way of working, but again, look for the opportunity to shift left and contribute earlier. Although this appears to be a technical initiative, I think it is driven more by our businesses. I learned recently that marketing budgets are often bigger than company IT budgets nowadays. Have a think about that.

Marketers may be difficult stakeholders to deal with sometimes, but their power is increasing, they want everything and they want it now. The only way to feed their need for new functionality is to deliver in continuous, small, frequent increments. If you can figure out how you can add value to these new ways, speak up and volunteer. Of course large, staged projects will continue to exist, but the pressure to go 'continuous' is increasing and opportunities in the job market require continuous, DevOps and analytics skills more and more. Embrace the change, don't resist it.

I wish you the best of luck in your leftwards journey.

Summary

When we moved from testing dumb-terminals to GUIs, the number of possibilities exploded and it took time to get the hang of it. It was like learning to test in three dimensions rather than two. Testing the Internet of Everything requires a similar increase in scale, diversity and complexity. Testing in the fourth dimension, perhaps?

Although at the scale of a single component or sub-system testing is pretty much the same as before, for system testing we need to adopt simulation methods using high volume test automation. The tools we will need to do this probably don't yet exist so the pioneers in this space will have to customise performance and functional test tools and write their own drivers for a while.

High volume test automation requires test models, test data generators and automatic oracles. Modelling, simulation, analytics, visualisation and tool-supported decision-making will become important capabilities of test architects and testing teams. Testers used to (so-called) manual testing will have to learn how to create better test models and how to use them with more technical modelling and simulation tools.

Obtaining trustworthy, accurate test environments and meaningful test data will, as always, cause big headaches for testers.

Large-scale test environments in the lab and in the field will be required and the boundaries between experimentation in production and testing in the lab will become blurred. Test Analytics derived from DevOps processes will become a critical discipline in testing the Internet of Everything.

Some new test approaches will be required and the New Model is an attempt to trigger new thinking in this area.

References

1. The Internet of Everything – What is it and how will it affect you?, Paul Gerrard, http://gerrardconsulting.com/sites/default/files/IoEWhatIsIt2.pdf
2. Internet of Everything – Architecture and Risks, Paul Gerrard, http://gerrardconsulting.com/sites/default/files/IoEArchitectureRisks.pdf
3. Mobile World Congress, Barcelona, 2-5 March 2015, http://www.mobileworldcongress.com/.
4. Internet.org by Facebook, http://internet.org/
5. White Space Spectrum, http://en.wikipedia.org/wiki/White_spaces_(radio)
6. Internet.org App Launches in Colombia, http://newsroom.fb.com/news/2015/01/internet-org-app-launches-in-colombia/
7. An Overview of High Volume Test Automation, Cem Kaner, http://kaner.com/?p=278
8. Barcelona Urban Lab, http://www.22barcelona.com/content/view/698/897/lang,en/
9. Thinking Big: Introducing Test Analytics, Paul Gerrard, http://www.gerrardconsulting.com/sites/default/files/TestAnalytics2.pdf
10. XMPP, The Extensible Messaging and Presence Protocol , http://xmpp.org/about-xmpp/
11. MQTT, " a machine-to-machine (M2M)/"Internet of Things" connectivity protocol, http://mqtt.org/
12. Gerdle, Gerrard Consulting, http://gerdle.com
13. ZigBee, http://en.wikipedia.org/wiki/ZigBee
14. 6LowPAN, IPv6 over Low Power Networks, http://datatracker.ietf.org/wg/6lowpan/documents/
15. DASH7, RFID standard, http://en.wikipedia.org/wiki/DASH7
16. Bluetooth, , http://en.wikipedia.org/wiki/Bluetooth
17. NFC, Near Field Communication, http://en.wikipedia.org/wiki/Near_field_communication

18. GPS Logger for Android, http://code.mendhak.com/gpslogger/
19. Test Axioms, Paul Gerrard, http://testaxioms.com
20. The Tester's Pocketbook, Paul Gerrard, http://testers-pocketbook.com/
21. A New Model for Testing, Paul Gerrard, http://dev.sp.qa/download/newModel

Paul Gerrard

Paul Gerrard is a consultant, teacher, author, webmaster, developer, tester, conference speaker, rowing coach and a publisher. He has conducted consulting assignments in all aspects of software testing and quality assurance, specialising in test assurance. He has presented keynote talks and tutorials at testing conferences across Europe, the USA, Australia, South Africa and occasionally won awards for them.

Educated at the universities of Oxford and Imperial College London, in 2010, Paul won the Eurostar European Testing excellence Award and in 2013, won The European Software Testing Awards (TESTA) Lifetime Achievement Award.

In 2002, Paul wrote, with Neil Thompson, "Risk-Based E-Business Testing". In 2009, Paul wrote "The Tester's Pocketbook" and in 2012, with Susan Windsor, Paul co-authored "The Business Story Pocketbook".

He is Principal of Gerrard Consulting Limited and is the host of the UK Test Management Forum and the UK Business Analysis Forum.

Mail: paul@gerrardconsulting.com
Twitter: @paul_gerrard
Web: gerrardconsulting.com