THE LATEST TRENDS AND TECHNOLOGIES IN THE IT WORLD

BRIGHT MINDS SHARP SOLUTIONS | LUXOFT

LOGEEK MAGAZINE

6/2017

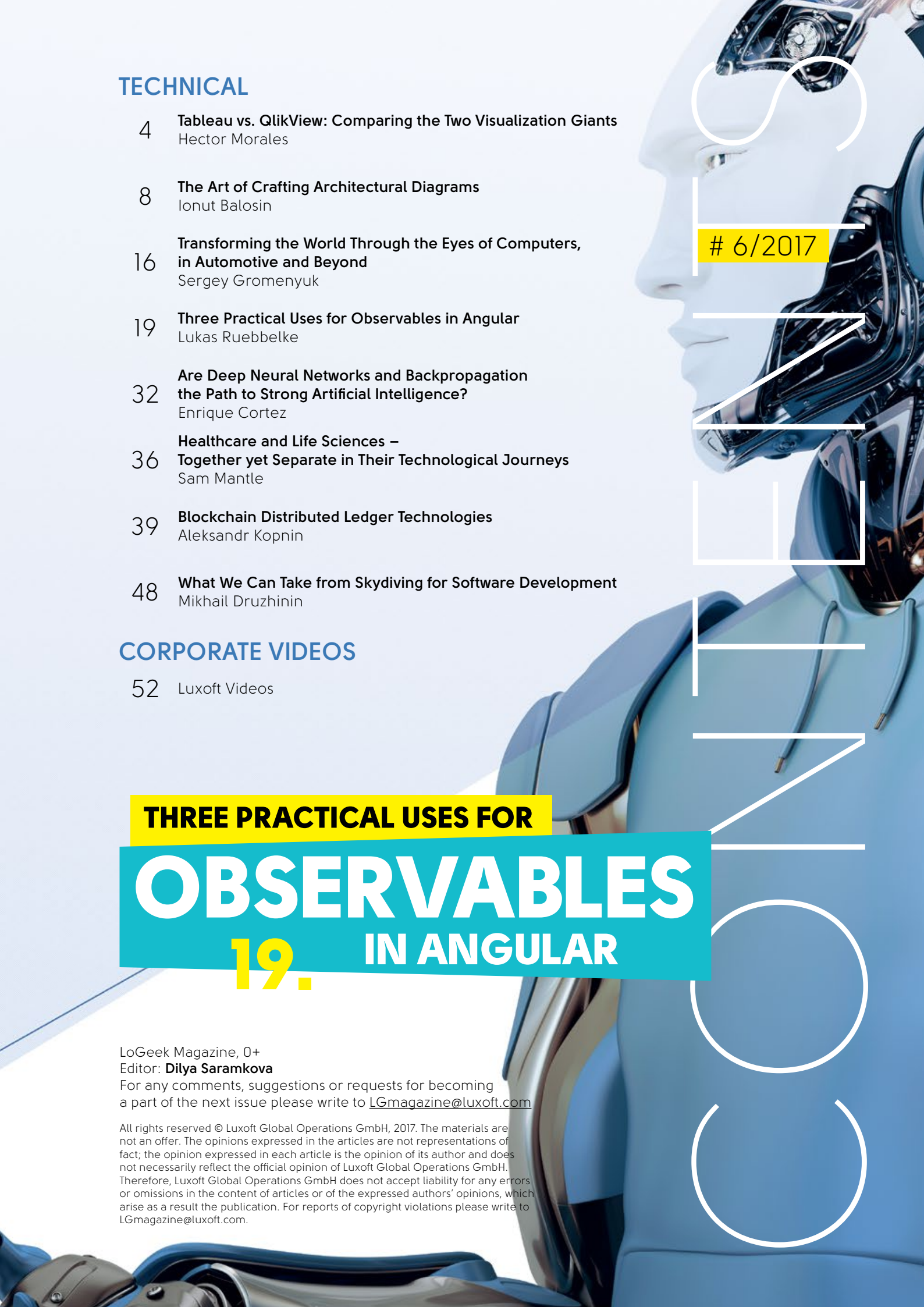
LTS# LOGEEK NIGHT

0+

Explore the content and find out more
at **www.luxoft.com/minds/**

BRIGHT MINDS
SHARP SOLUTIONS

LUXOFT



6/2017

TECHNICAL

CORPORATE VIDEOS

THREE PRACTICAL USES FOR OBSERVABLES IN ANGULAR

19.

LoGeek Magazine, 0+
Editor: **Dilya Saramkova**
For any comments, suggestions or requests for becoming a part of the next issue please write to LGmagazine@luxoft.com

TABLEAU VS QLIKVIEW

COMPARING THE TWO VISUALIZATION GIANTS

HECTOR MORALES

Today's businesses do not rely upon an analytical tool just because it's the latest in the market. They want advanced, convenient and smart tools that help analyze complex data instantly. Gone are the days when one-dimensional and static graphs and charts were used to demonstrate statistics. Businesses need more visually appealing, interactive and agile tools that fit into real-time requirements.

Tableau's deep expertise and rich features are evident in its advanced, interactive capability for visual exploration, its analytics dashboards and support for mobile-based exploration and authoring. Analytics are supported throughout visual exploration features, and dashboards and support with mobile authoring all reside in the cloud, providing enterprises with flexibility in reporting and analysis.

Qlik's architecture has been designed to support self-contained ETL and data storage, making this series of features a strength of the system when compared with Tableau. Both Qlik and Tableau are evenly matched on self-service data preparation, an area which both companies have publicly stated they are investing a significant amount of their R&D budgets to this year.

Qlik and Tableau are also well-matched on all aspects of analytics and BI, with Tableau having the edge for its platform work integration, ease of use, and visual appeal. For enterprises that rely on a wide variety of data integration points, Qlik has the advantage, given the depth and breadth of its options in this area.

01 EASE OF USE

Tableau: Its interface is simple, not overcrowded with too many features on one page and has a drag and drop interface. It doesn't provide a feature that allows the user to search for content across all their data. Users can easily create their own views using various objects, and it's easy because of a well-designed GUI interface.

QlikView: It's easy to use and explore hidden trends. To search, just type any word in any order into the search box for instant and associative results and it will show connections and relationships across data. It is difficult for the user to design their own views due to menu-driven properties.

02 EASE OF LEARNING

Tableau: It also has an actively engaged community and resources. Tableau is a simple drag and drop application, making it very easy to learn.

QlikView: It has an actively engaged community with resources to help you learn the software in the best possible manner.

03 COST

Tableau: Tableau offers a free desktop version called "Public", which makes data available for download. Private versions come with a fixed fee of $999 or $1,999, depending on data access. For Tableau Server, anecdotal evidence says $1000 per server user with a minimum of ten users, plus maintenance.

QlikView: The personal edition is free, with limited document sharing. Each named user license is $1,350 and it's $15,000 to add a concurrent user. The server license is $35,000 per server. It costs an additional $21,000 per server for PDF distribution service and $22,500 for SAP NetWeaver connector. May require RAM upgrades if there are large numbers of concurrent users.

04 CONNECTIVITY WITH OTHER TOOLS, LANGUAGE, DATABASE

Tableau: Can integrate with a broader range of data sources including spreadsheets, CSV, SQL databases, Salesforce, Cloudera Hadoop, Firebird, Google Analytics, Google BigQuery, Hortonworks Hadoop, HP Vertica, MS SQL Server, MySQL, OData, Oracle, Pivotal Greenplum, PostgreSQL, Salesforce, Teradata, and Windows Azure Marketplace. It can connect with R, which fuels the tool's analytical capabilities of the tool. It can also connect with big data sources.

QlikView: Can integrate with a very broad range of data sources, such as Amazon Vectorwise, EC2, and Redshift, Cloudera Hadoop and Impala, CSV, DatStax, Epicor Scala, EMC Green Plum, Hortonworks Hadoop, HP Vertica, IBM DB2, IBM Netezza, Infor Lawson, Informatica Powercenter, MicroStrategy, MS SQL Server, My SQL, ODBC, Par Accel, Sage 500, Salesforce, SAP, SAP Hana, Teradata, and many more. It can connect with R using API integration, and can connect with big data.

05 DEPLOYMENT PROCESSES & SYSTEM REQUIREMENTS

Tableau: Does not have its own data warehouse. Cannot create layers while connecting with data sets. Easier to deploy because it requires more structured data.

QlikView: Has its own data warehouse and the addition of a scripting feature adds value. We can use multilevel layers in QlikView deployment. QlikView is easily deployable and configurable, and produces stunning reports within minutes of installation. This product does not use cubes; hence, it loads all the tables and charts in memory to enable interactive queries and the creation of reports, a technology not found in other products. It can be developed on both 32 and 64 bit. Its associative technology makes data modeling easier.

06 VISUALIZATION OBJECTS

Tableau: Offers good visualization for objects, with better formatting options. Good geo-spatial visualizations. Numerous options for vizualizing data. Visualizations are always top-quality.

QlikView: Offers good options for visualizing information, and is loaded with objects. We can play with the properties of these objects easily to customize them. We can customize objects easily by playing around with their properties, and it's also easy to create custom charts like waterfall, boxplot, or geospatial charts by customizing properties. While inserting objects, QlikView offers layout and formatting options fitting with the document theme. Here, we need to work on formatting options to make it more visually appealing.

Advantages of Tableau

EVERYONE CAN USE IT
Tableau allows the common user to generate interactive business intelligence reports, even with little technical proficiency. Tableau is perfectly suitable for top management employees who need a smart business intelligence tool that is easy to operate.

FAST
Tableau is able to create interactive visualizations and reports in a few minutes, making it a very fast tool.

MAPS
Tableau is efficient in creating multi-dimensional maps with the help of its built-in geocoding feature.

INTERACTIVE DATA VISUALIZATION
Tableau prepares interactive reports and gives recommendations according to user specifications. Tableau is rich in data visualization capabilites, thanks to which it automatically measures and segments data.

TIME AND EFFORT SAVING
One can easily connect to the database using Tableau, which reduces time and effort that can be better used in analytics.

COST-EFFECTIVE
Not only does it save time and effort, but money as well - this tool allows companies to avoid investing more money in acquiring additional IT resources.

INFORMATION SHARING
Tableau creates reports and allows sharing online and offline, making the process faster.

MAINTAINS SECURITY
Tableau controls information access through various security measures.

Advantages of QlikView

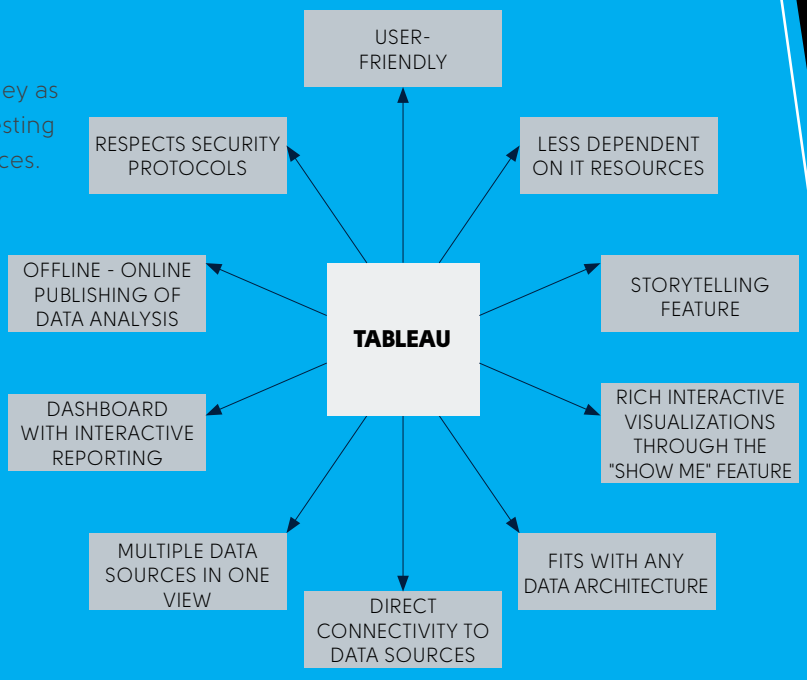
USER-CENTRIC INTERACTIVITY
QlikView is highly user-centric, as it allows the user to find meaningful insights instantly with the help of a rich set of visualization features.

FIND ASSOCIATIONS
Qlikview lets you find associations between datasets and conduct searches across all available data, and then filter results according to your needs.

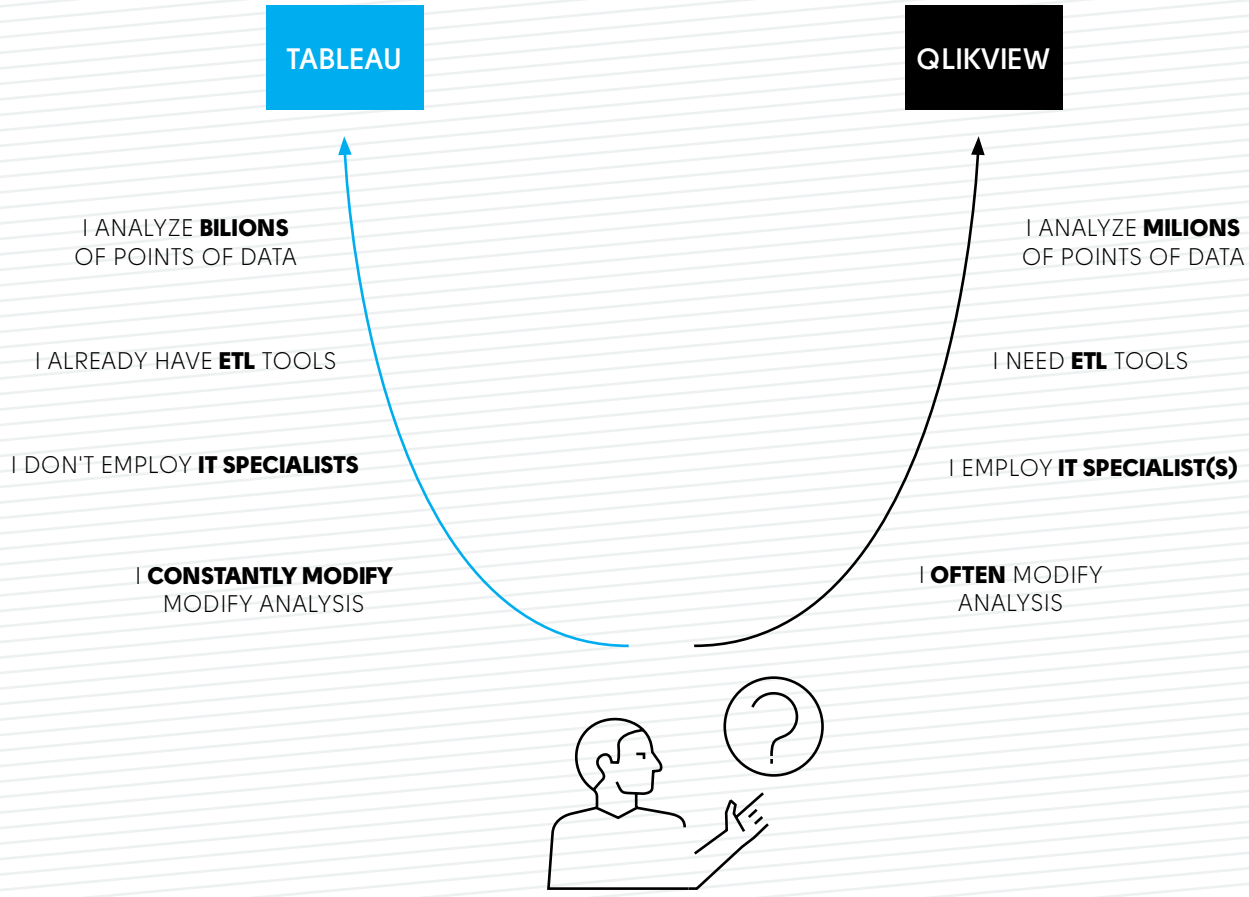
FLEXIBILITY IN ACCESS
You can access your data in QlikView while sitting on the opposite side of the world at any time, regardless of the device being used.

FAST ANALYSIS
QlikView is able to deliver results the moment you search. Its highly optimized nature and scalability helps it access large data sets on the fly.

QUICK TIME TO VALUE
Allows you to measure time-to-value for whatever time duration you choose and find accurate results in no time.

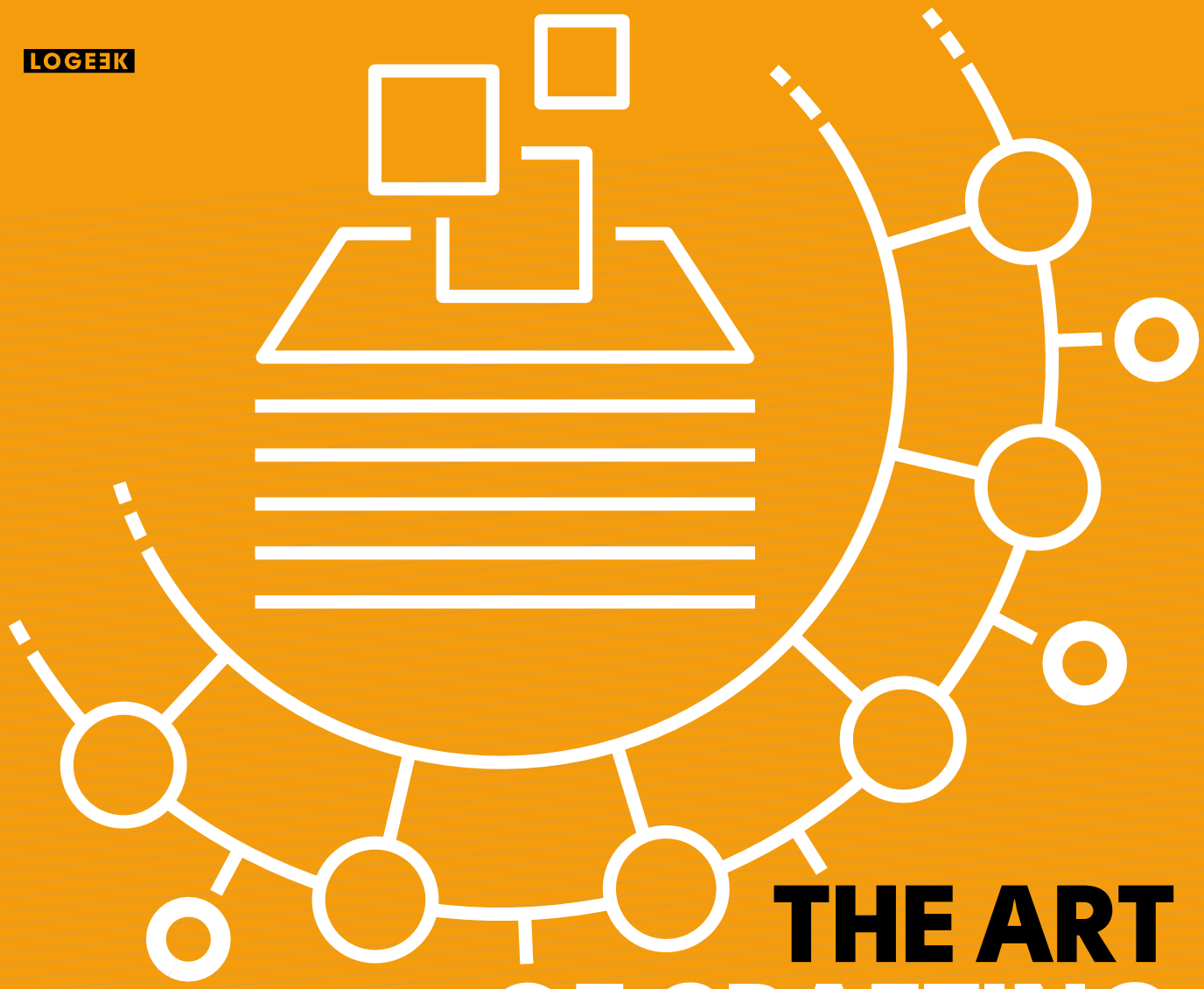


IN CONCLUSION: EITHER CAN BE BEST FOR YOU, DEPENDING ON WHAT YOU ARE LOOKING FOR.



HECTOR MORALES KOTT

Hector Morales Kott is an Analytics Professional with a passion for driving actionable insights through data visualizations in multiple industries such as retail, technology, travel, manufacturing, and the public sector. Hector is working as a Senior Business Intelligence Consultant for Luxoft, where he is developing high-quality BI Tableau solutions/dashboards for a client's supply team. Previously Hector worked as Lead Analyst BI & Reporting Latin America for The Hershey Company, where he started as a Sales Representative and acquired a unique feel about the real information necessities in the field. He graduated with a Bachelor of Science degree in Computer Engineering from Anahuac University. Contact him at www.hectormoraleskott.com.



THE ART OF CRAFTING ARCHITECTURAL DIAGRAMS

IONUT BALOSIN

The need to create architectural diagrams might arise at some point with any software project we are involved in. Whether we are following a formal architectural model (e.g., Kruchten 4+1, Rozanski & Woods, etc.) or not, there is still a need to document some parts of the application by creating diagrams. In software architecture, such diagrams are created incompliance with views which are related to a specific viewpoint that could be part of a model, but in the current article I prefer to stick to an informal use of the term "architectural diagram"; I don't intend to cover all other aspects.

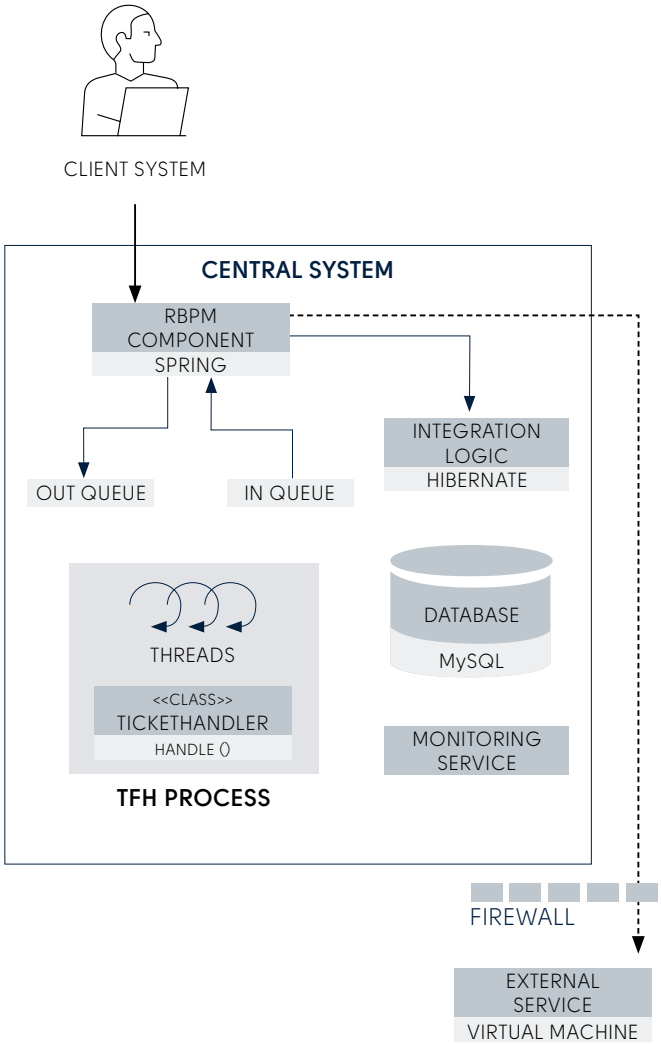
Based on my experience as a software architect and technical trainer, there are multiple discrepancies with how architectural diagrams are created. This varies from project to project, inside project teams, and even from developer to developer. I see many instances of inconsistency, fragmentation, and granularity of rendered information and the look of diagrams. Compared to an architectural model which must be formal and don't necessarily follow a specific standard. Nevertheless, diagrams must be self descriptive, consistent, relatively accurate and connected to the code. That's why it is important that every architect or software engineer rely on several guidelines when creating architectural diagrams, since they are the common ground for communicating the application's architecture over time (e.g., structure, elements, relationships, properties, and principles), and across different stakeholders with various technical backgrounds and concerns.

KEY TAKEAWAYS

- Designing architectural diagrams might not be an easy task; they can be tricky and error prone, even the simplest ones. Creating consistent and meaningful diagrams, however, brings clarity and consensus across different stakeholders.
- In most cases, the real issues are not strictly related to using a less efficient Architectural Description Language (e.g., UML), but the misunderstanding of the diagram's importance, relying on improper or inconsistent guidelines or even the lack of architectural education.
- In the process of creating diagrams, try to blend those that are automatically generated with manually created ones in order to minimize the work, illustrate different set of concerns and cover multiple abstraction levels of the system.
- As the system evolves, maintaining up-to-date diagrams requires extra effort. We need to know how to proceed efficiently in such cases while still maintaining consistency and robustness across architectural diagrams.
- Modern architectures bring extra complexities, which are reflected in the diagrams. Additional concerns might emerge and could easily lead to cluttered, incomplete or fragmented diagrams; hence, finding the right balance is key

CURRENT PITFALLS WHEN DESIGNING ARCHITECTURAL DIAGRAMS

Before going deeper into ppotential issues, I would like to quote an English idiom, "A picture is worth a thousand words." According to the Wiki explanation, "It refers to the notion that a complex idea can be conveyed with just a single still image or that an image of a subject conveys its meaning or essence more effectively than a description does." The same concept applies to an architectural diagram: if it raises more questions than answers, the diagram has not been constructed well. An architectural diagram should never require a thousand explanations!



Graph 1: An example of an improper architectural diagram. It suffers of most of the issues described below.

Let's now go through a list of pitfalls that could hinder the process of creating good architectural diagrams.

01 WHAT DOES A BOX OR SHAPE DENOTE?

Using a box or shape that is improperly documented might result in multiple interpretations. It might be associated with a piece of data, a bunch of code, or a process. A simple box in a diagram might raise multiple questions. It is important to avoid them by explicitly listing details about the box or shape's meaning in the diagram legend.

02 WHAT DO A SHAPE'S DIFFERENT EDGES REPRESENT?

Each edge of a shape (e.g., dashed, dotted, etc.) can be misunderstood in a poor diagram. Does a specific border refer to a specific component type (e.g., a dashed line refers to a container, a microservice, a layer, etc.), or it is just the designer's preference to have a rich look and feel? Avoid such confusion by providing accurate details in the legend diagram when choosing multiple or non-standard edges.

03 WHAT DOES A LINE OR ARROW DENOTE?

A line or an arrow can be interpreted either as data flow (data flows from system A to system B), or as a relationship across elements (component A depends on component B). In most cases the relationships or data flows represented by arrows do not converge in the same directions and it is important to explicitly document this in the diagram legend.

04 WHAT COMMUNICATION/ ASSOCIATION TYPE DOES A LINE OR ARROW REPRESENT?

Even if the line refers to data flow or a relationship across components, the communication type (the case of data flow) or the association type (relationship) denoted by that line or arrow must be detailed. For example, if the line represents data flow, the communication might be synchronous or asynchronous, but if the line refers to a relationship, it might be represented by a dependency, inheritance, implementation, etc. All of these details must be present in the diagram legend.

05 WHAT DOES THAT COLOR MEAN?

Having a diagram with multiple colors for boxes and lines without a corresponding key will raise multiple questions (Why are some boxes green while others are red? Why are some lines black and others blue?). Color scheme is less important in a diagram, and using a rich number of colors doesn't add valuable content or information. A diagram can be self-explanatory and well-designed with just the use of black and white. Unless there is a reason to emphasize some parts of the diagram by using distinguishable colors, it adds unnecessary complexity. It is always better to stick to simplicity in terms of colors, but if you choose to use them, don't forget to clearly label what each color represents.

06 MISSING RELATIONSHIPS BETWEEN DIAGRAM ELEMENTS OR ISOLATED ENTITIES

Missing relationships between elements or isolated entities in a diagram might indicate incompleteness. From both a structural and behavioral perspective, each element should have a relationship, represented by a line or an arrow, with another part of the system, which is represented through another element.

07 MISLEADING/ UNDOCUMENTED ACRONYMS OR VAGUE AND GENERIC TERMS

When labeling in a diagram, avoid using misleading or undocumented acronyms that might cause confusion. A sequence of letters (e.g., TFH, RBPM) do not mean anything without a corresponding explanation on the diagram, or even better - in the diagram legend (e.g. TFH - ticket feed handler, RBPM - rates business process manager). Another problem encountered in naming diagram elements relates to the use of extremely vague or generic terms (e.g., business logic, integration logic, etc.). The issue might reside at the code level as well. Always use self-explanatory names and follow clean code principles.

08 EMPHASIZING TECHNOLOGIES, FRAMEWORKS, PROGRAMMING (OR SCRIPTING) LANGUAGES, IDE OR DEVELOPMENT METHODOLOGY ON DIAGRAMS

Architectural design is not fundamentally based on any one technology, framework, programming or scripting language, IDE or development methodology. All of these come later on in the process to help build the architecture, but they are not the central focus. They should not be included in diagrams, but stated in the architectural description instead, including the rationale around choosing them.

09 MIXING RUNTIME AND STATIC ELEMENTS IN THE SAME DIAGRAM

Runtime elements (threads, processes, virtual machines, containers, services, firewalls, data repositories, etc.) are not present at compilation time and mixing them with static elements (components, packages, classes and so on) in the same diagram should be avoided. There are dedicated diagram types (such as concurrency diagrams or deployment diagrams) which are primarily focused on runtime elements. It is important to distinguish between these two categories and to avoid mixing them as much as possible.

10 MAKING ASSUMPTIONS LIKE "I WILL VERBALLY DESCRIBE THIS," AND "I WILL EXPLAIN IT LATER."

Everything not described by the diagram itself is missing, and there isn't enough room to provide written details to complement a diagram. All explanations mentioned verbally but not captured in the diagram are lost later on when other stakeholders, such as the develop or the architect, read the diagram and can't understand it. Include all necessary details in the diagram to avoid unnecessary confusion and the need for further clarification.

11 CONFLICTING LEVELS OF DETAIL OR MIXED ABSTRACTIONS

Adding elements related to different levels of abstraction in the same diagram can conflict since they are seen from different perspectives. For example, adding components to an architectural context diagram or classes to a deployment diagram can distract from the purpose of the diagram itself. When creating a diagram, try to stick with the same level of abstraction.

12 CLUTTERED OR VAGUE DIAGRAMS SHOWING TOO MUCH OR TOO LITTLE DETAIL

"Everything should be made as simple as possible, but no simpler, " is a well-known quote attributed to Albert Einstein. This is valid for architectural diagrams as well; the level and the granularity of captured information should be meaningfully chosen. This is not an easy thing; it depends on the architectural model used, the experience of the architect and the complexity of the system.

GUIDELINES TO FOLLOW WHEN CREATING ARCHITECTURAL DIAGRAMS

Apart from the pitfalls above, which are a prerequisite checklist of what to avoid, there are also general guidelines about what you should do to properly create diagrams.

CHOOSE THE OPTIMAL NUMBER OF DIAGRAMS

As Philippe Kruchten said, "Architecture is a complex beast. Using a single blueprint to represent architecture results in an unintelligible semantic mess." To document modern systems we cannot end up with only one sort of diagram, but when creating architectural diagrams it isn't always straightforward which diagrams to choose and how many to create. There are multiple factors to consider before making a decision, such as, for example, the nature and the complexity of the architecture, the skills and experience of the software architect, the time available, the amount of work needed to maintain them, and what makes sense or is useful for meeting stakeholder concerns. For example, a network engineer will probably want to see an explicit network model including hosts and communication ports and protocols, while a database administrator is concerned about how the system manipulates, manages and distributes data, etc. Based on all of these concerns, you should pick the optimal number of diagrams, whatever that number is. If there are insufficient diagrams, parts of the architecture might be hidden or undocumented. On the other hand, if there are too many, the effort needed to keep them consistent and updated might considerably increase.

KEEP STRUCTURAL AND SEMANTIC CONSISTENCY ACROSS DIAGRAMS

Diagrams should be consistent with one another in terms of boxes, shapes, borders, lines, colors, and so on. The structural look and feel should be the same and each stakeholder should have no difficulty understanding diagrams created by different developers on the same team. Ideally, stick to a common diagramming tool and reuse it across all projects.

From a semantic point of view, diagrams should be periodically synchronized with the latest code changes and with one another, since a change in one diagram can impact others. This process can be manually or automatically triggered by using a modeling tool. The latter is the preferred mechanism but this varies from project to project. In all cases, the idea is to maintain consistency between diagrams and code, independent of the method or tool. Simon Brown said, "Diagrams are not useful for architectural improvement if they are not connected to the code," which emphasizes the idea of semantical consistency.

PREVENT DIAGRAM FRAGMENTATION

Having multiple diagrams might make the architectural description difficult to understand but also to maintain. As a side effect, fragmentation might occur (for example, when two or more diagrams illustrate the same quality attribute such as performance, scalability, etc., but each of them is individually incomplete). In such cases the diagrams that do not reflect relevant quality attributes (linked to architecturally significant requirements) should be either removed or even better the diagrams should be merged together.

KEEP TRACEABILITY ACROSS DIAGRAMS

Making comparisons between different versions of diagrams plus easily reverting to a previous version is important. Use a modeling tool that allows you to across diagrams. do this. The latest trends in the industry rely on using simple and intuitive plain-text language to generate diagrams, which seems to solve the traceability concern. Another advantage of such an approach is that it implicitly ensures homogeneous structural consistency across diagrams.

ADD LEGENDS NEXT TO ARCHITECTURAL DIAGRAMS

If you do not follow a standard architectural description language (UML, ArchiMate, etc.), detail every piece of the diagram in the legend, including boxes, shapes, borders, lines, colors, acronyms and so forth. If you do, just add the architectural description as a key in the legend and there will be no need for additional explanations, since readers will follow those language specifics to understand the diagram.

DOES THE ARCHITECTURAL DESCRIPTION LANGUAGE (E.G. UML, ARCHIMATE, ETC.) MAKE A DIFFERENCE?

There are a lot of opinions regarding which description language should be adopted for a project. Some argue that UML is rigid and not flexible enough to model architectural design, a point of view which I agree with. Nevertheless, in some cases it might be more than sufficient to document the fundamentals of an architecture without relying on any UML extensibility features like profiles and stereotypes. By taking a look at other description languages, we can see that ArchiMate is more powerful and suitable for modeling enterprise systems compared to UML. There is also BPMN, which is particularly targeted at business processes, etc. The comparisons might continue, but I do not intend to give an in-depth review of them here.

Having an architectural description language that is both comprehensive and flexible enough should be a solid criteria when making your choice. From my perspective, however, the real cause resides somewhere else and is related to the fact that architectural documentation is not created at all. People often find the process of creating it boring, useless or pointless. The number of software projects with improper documentation, or none at all, is huge. I don't think people are purposefully creating or involved in the creation of architectural diagrams using an improper description language (and if they were to replace them with a better one the results would be architectural diagrams); rather, most of them have no idea how to properly create it at all.These are the things we need to address first - to understand why documentation matters and then how to properly create it by training software engineers. After that, the selection of proper tools comes naturally.

HOW CAN DIAGRAMS BE KEPT UP-TO-DATE AS THE SYSTEM IS BEING DEVELOPED, AND HOW DO CHANGES TO THE ARCHITECTURE MATERIALIZE?

There are a few approaches to keeping diagrams updated. I will touch on three of them.

The first approach, and the easiest one, is to automatically generate diagrams out of the source code, which is the ground truth. This guarantees they are all consistent to the code. Unfortunately, with existing tools this isn't fully possible yet (at least to my knowledge), since tools cannot create any type of accurate and meaningful diagram based only on the source code without significant manual intervention. Len Bass said, "The ideal development environment is one for which the documentation is available for essentially free with the push of a button," implying the autogeneration of diagrams, but we're not at that point yet.

The second approach would be to design diagrams first using a dedicated tool which then generates the skeletons of the source code (components/packages with boundaries, APIs, etc.) used later on by developers to fill in the code. This way, every change in the architecture is triggered by the diagram itself, which automatically regenerates or updates the code skeleton.

The last approach involves manually updating the diagrams every time a new feature is implemented that impacts the architectural design. To be sure all code changes are reflected in the diagrams, updating diagrams should be part of the "definition of done" in the development process. This option can easily result in outdated or inconsistent diagrams (developers often forget or are not in the mood to update diagrams), and unfortunately this still happens frequently. This option comes least recommended.

Taking existing tools into account, my recommendation is to have a mix. For example, try to autogenerate diagrams, which can be reasonably rendered by tools based on source code without too much noise (cluttered or meaningless information). In this category, we can include both static diagrams and diagrams more prone to frequent development changes and with a high degree of volatility. Such diagrams might include context diagrams, reference architecture diagrams, package diagrams, class diagrams, entity diagrams, etc. Nonetheless, in some cases, it isn't obvious based only on the source code how the system meets quality attributes (such as availability, scalability, performance), so it isn't sufficient to automatically generate diagrams. In such cases, automatic generation needs to be complemented by manually-modeled diagrams. Some examples of such diagrams include sequence diagrams, state diagrams, concurrency diagrams, deployment diagrams, operational diagrams, and so on.

WHAT COMPLICATIONS (OR SIMPLIFICATIONS) EMERGE FOR ARCHITECTURAL DIAGRAMS WHEN DEALING WITH MODERN ARCHITECTURES - FOR EXAMPLE, MICROSERVICES?

Microservices or any other modern architectural style (serverless, event-driven, etc.) only drive the structure of the system, how the components communicates each other and what principles govern them. Personally, I don't think the architectural style should change the rationale or concept around creating the diagrams (and implicitly, the architectural description), or what they should capture. Nevertheless, when we talk about modern architecture systems that are more complex than older, classical systems, it definitely has an impact on the descriptions and diagrams, in the sense that there are multiple aspects to consider. Such considerations might include understanding the number of distributed components like distributed microservices, each component type, how the components communicate with one another, APIs and messages, their lifecycles and who owns them.

Taking all of this into account, views that capture decomposition, development, deployment and operability should be considered by default. Imagine a system with an impressive number of microservices. In such a case the number of diagrams might significantly increase because each microservice might end up having its own set of corresponding diagrams. Issues regarding consistency (for example, changing the API of one service impacts other services, therefore all impacted diagrams need to be updated), fragmentation (e.g. high availability or performance between distributed services is not consolidated into one diagram), or cross-cutting concerns (like who is responsible for illustrating aspects like monitoring or security across entire system elements in a concise manner), may be difficult to handle. On top of this, challenges may arise related to a team's ability to communicate and collaborate effectively both during project development and after, when it comes to maintenance.

To summarize, additional concerns can arise in modern systems with complex architecture that can lead to complications, even at the basic level of the diagram.

IONUT BALOSIN

Ionut Balosin is a software architect with 10+ years of experience in a wide variety of business applications. He is keen on performance & tuning and software architecture topics. At Luxoft, he conducts trainings on Java performance and tuning, and software architecture. Ionut is a regular speaker at both Luxoft internal events (LoGeek, SoftLabs) and external conferences (Voxxed, BJUG, DevTalks, XP Days, Agile Tour, etc).


LTS#
LOGEEK NIGHT
Develop your skills & keep up
with the latest trends
LoGeek Night events offer a great chance to meet professionals with similar interests and passions. Learn and discuss the latest technologies in a laidback gathering replete. with refreshments.
All you have to do is keep an open mind, and come!
https://career.luxoft.com/events-and-webinars/
BRIGHT MINDS
SHARP SOLUTIONS
LUXOFT

TRANSFORMING THE WORLD THROUGH THE EYES OF COMPUTERS

IN AUTOMOTIVE AND BEYOND

SERGEY GROMENYUK

While still an emerging technology, computer vision is shaking up industries in new, innovative ways. It aids adopters in everything from making movies to monitoring traffic. At Luxoft, we are working on the development of modern and future technologies that utilize computer vision, notably in Automotive. Computer vision is the key technology that lets the car "see" and analyze the environment. It makes possibilities like advanced driver assistance systems (ADAS) – which demand flawless safety and reliability – possible. The truth is, it's important to understand there is a need for computer vision. Why is it so disruptive, and why is everybody seemingly implementing it?

WHAT IS COMPUTER VISION?

Computer vision is the process of taking data from images or video footage and transforming it into new representations. The transformation might be converting a grayscale image from a color image, recognizing a pedestrian crossing the street by analyzing color and brightness, or producing high-quality video footage by removing camera shake. To assist in the creation of these new representations, the input data may contain useful contextual information such as "the camera is mounted in a car" or "the laser scanner detects an object within a 10-meter distance."

Because humans are visual creatures, this process may appear to be a simple task. After all, how hard can it be to identify a pedestrian when they are standing right there in front of you?

The truth is, seeing an object is more complicated than it sounds. For humans, the brain divides the light hitting the retina from an object into many channels, each streaming different information back to the brain. In response, the brain's visual attention system identifies important streams to analyze – depending on the reason for viewing the object – while suppressing the analysis of others.

This massive amount of feedback involves widespread associative inputs from muscle control sensors and the senses – sight, smell, touch, taste and hearing – allowing the brain to draw cross-associations from past experiences. These feedback loops return to previous processing stages, including streaming to the brain and the "hardware sensors," aka the eyes, which control lighting via the iris, essentially "tuning" the reception of light on the surface of the retina.

However, in a computer vision system, the computer receives a grid of numbers from a camera or a disk with a microcontroller unit – and that's it.

Computer vision systems, unlike people, do not actually see real-world objects. They only "see" a grid with numbers corresponding to the brightness or saturation value of each analyzed pixel in a photo. A camera in the computer vision system also lacks associative thinking and pattern recognition – both human skills that help us identify objects. This is what makes the process of developing computer vision systems so complex. Converting a table with numbers into a deterministic representation or perception, such as a pedestrian crossing the road, is no simple feat. But thanks to various algorithms, it can be done.

WHY SHOULD I CONSIDER COMPUTER VISION?

Computer vision helps automate routine processes, cutting costs and maximizing efficiency. All that engineers have to do is build trainable systems that process incoming information from real-time visual data streams. Soon enough, we'll be able to equip machines with the same self-learning cognitive abilities that humans possess, eliminating the need to train the computer to learn a number-based table with associated data. In combination with the ability of computers to do monotonous work quickly and cheaply, these systems will positively impact the development of businesses across industries. Here are just some of the many uses of computer vision:

- **Face and fingerprint recognition**
- **Surveillance monitoring for intruders**
- **Analyzing highway traffic**
- **For moviegoers, computer vision is used to merge computer-generated imagery (CGI) with live action footage by tracking feature points in the source video to estimate the 3D camera motion and shape of the environment.** Widely used in Hollywood, this technique requires the use of precise mapping to insert new elements between foreground and background elements.
- **For machinery, computer vision speeds up quality assurance inspections by using stereo vision with specialized illumination to examine machine parts.** This includes measuring the tolerances of aircraft wings, auto body parts or finding defects in steel castings by using X-ray vision.
- **In healthcare, using computer vision allows practitioners to implement medical image analysis** (such as noninvasive diagnosis, image-guided radiotherapy and treatment planning), or perform long-term studies on brain morphology, following subjects as they age.

COMPUTER VISION IN AUTOMOTIVE

Specifically, computer vision plays a key role in reshaping the automotive industry. It's associated with the goal of making the car safer, more convenient and more intelligent.

Initially, computer vision was introduced as a technology that improves advanced driver assistance systems (ADAS), which help the driver and keep them safe. It was prevalent in the lane assist system (which helps keep the vehicle in the desired road lane), road sign recognition, obstacle detection (such as spotting pedestrians), blind area monitoring, road boundary detection, active cruise control and parking assistance systems. Further development of ADAS led to the creation of autopilot, which takes control of the car in relatively simple situations (such as highway traffic). Consequently, the next step for computer vision will be using it to develop control systems for autonomous vehicles.

Thus, the future of the Automotive industry is dependent on computer vision. For users, this technology is important because it allows automakers to create simpler and safer cars, improving the user experience. Drivers don't have to worry about accidentally hitting pedestrians or driving outside of their lane, since the car "sees" where you're going and the environment around you.

For example, one of the key computer vision applications for Automotive is technology that recognizes road signs. When driving, it's easy to get distracted and accidentally miss a sign. This is potentially dangerous – it could have been a sign that warns about a dangerously sharp turn or a speed limit decrease in a school zone. To mitigate this, computer vision allows the car to "see" the sign, recognize it and show it on the dashboard display for the driver to see.

As you can see, computer vision is being widely applied across industries as a cutting edge technology. It also integrates well with other sectors, such as artificial intelligence, robotics and autonomous vehicles, allowing for seamless experiences.

SERGEY GROMENYUK

Sergey Gromenyuk is a C++ and computer vision developer who who has been working on software for self-driving cars for more than four years. His professional strengths include real-time systems, Sensor Fusion, IMU and embedded systems. Sergey is currently applying his expertise for the Luxoft automotive team as a lead developer for computer vision algorithms.

THREE PRACTICAL USES FOR

OBSERVABLES IN ANGULAR

LUKAS RUEBBELKE

Observables have fundamentally changed the foundation of how we approach building applications. Whereas complexity in the form of managing state, controlling flow and code volume has been a challenge since the beginning of programming, observables have provided us with an elegant way of dealing with these difficulties.

Because JavaScript is asynchronous by nature, one of the most common conundrums of working with the language is how to determine when something is done. There has been a steady progression of solutions, starting with callbacks and leading up to promises. Promises were a huge step forward in inverting how events were communicated. They allowed the consumer to handle the result, and more importantly, chain units of logic together in a somewhat declarative fashion. The shortcoming of promises is that what they provided was relatively limited and the concept of "done" was in fact "one".

How do you handle an event that will happen more than once? What do you do if the event is ongoing? It is in these scenarios that observables elevate the entire playing field with how we handle truly complex and asynchronous scenarios. In this article, we are going to examine three basic use cases within the context of Angular and how they can be applied to your work.

USE CASE #01 TRANSFORMING DATA

If you asked an Angular developer if they have ever used an observable, I would wager that 99.9% of them would say yes. I would also wager that the majority of those developers have not gone beyond consuming a simple HTTP call and making the results available to a component. This level of understanding is superficial, and by itself does not provide any real advantage over using a promise.

A typical pattern that I have observed is when a component receives the result of an asynchronous call within a subscribe block and then proceeds to make significant changes to the payload so that it can work within the component. This approach is the equivalent of walking away from free money, as RxJS provides us with well over a hundred operators to help us transform our data into whatever shape we need. This is the perfect place to start our first example - how to use RxJS operators to process an observable stream so that the results arrive in **precisely** the same way we want at the `subscribe` block.

SETUP

Instead of working with an HTTP response, we are going to work with DOM events. They better represent observable streams because something is always happening in the browser. In the code below, we have a local variable in our template called #btn that we reference in our component class with the help of a @ViewChild decorator.

```
js
import { Component, OnInit, ViewChild } from '@angular/core';
@Component({
  selector: 'app-basic-sequence',
  template: `
  <button #btn md-raised-button color="accent">Click me!</button>
  <div class="container">
    <h1>{{message}}</h1>
  </div>
  `
})
export class BasicSequenceComponent implements OnInit {
  @ViewChild('btn') btn;
  message: string;

  ngOnInit() {
  }
}
```

Now that we have a reference to an element in our template, it is time for us to introduce an observable stream.

ENTER RXJS

We are going to use the `Observable.fromEvent` operator to start an observable stream from a DOM event. Next, we will capture a click event and then immediately fire the resulting stream in our subscribe method.

```
js
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/observable/fromEvent';
@Component({
  //...
})
export class BasicSequenceComponent implements OnInit {
  //...
  ngOnInit() {
    Observable.fromEvent(this.getNativeElement(this.btn), 'click')
      .subscribe(result => console.log('Button clicked');
  }

  getNativeElement(element) {
    return element._elementRef.nativeElement;
  }
}
```

Notice that we are only using the observable stream to tell us when the button was clicked, and nothing more. For every button click, we will see a new log statement in our console window. If we were to trace out the `result` parameter, we would see information about the DOM event itself.

MAP

The beauty of observables is that we are free to define our data however we see fit. We often do not care about the underlying DOM event, but rather want to associate a button click with a value. For instance, when we click a button on a calculator, we only care about the value that the pressed button represents. To accomplish this, we just need to map the incoming parameter to something we want. In this case, we are taking the DOM event and mapping it to a string of our choosing.

```
js
import 'rxjs/add/operator/map';
//...
export class BasicSequenceComponent implements OnInit {
  //...
  ngOnInit() {
    Observable.fromEvent(this.getNativeElement(this.btn), 'click')
      .map(event => 'Beast Mode Activated!')
      .subscribe(result => this.message = result);
  }
  //...
}
```

Because we have mapped the event to a new string, the result in our subscribe block is no longer the event object, but our custom string.

FILTER

RxJS operators are designed to operate as pure functions. As a result, we can stack them together to accomplish a sequence of logical transformations. For instance, if we only wanted to capture click events while the shift key was being held down, we could easily filter out any event that did not match that criteria.

```
js
import 'rxjs/add/operator/filter';
//...
export class BasicSequenceComponent implements OnInit {
  //...
  ngOnInit() {
    Observable.fromEvent(this.getNativeElement(this.btn), 'click')
      .filter(event => event.shiftKey)
      .map(event => 'Beast Mode Activated!')
      .subscribe(result => this.message = result);
  }
}
//...
```

When an event comes through the stream, the filter operator will check to see if the event.shiftKey property is true and if so, let it pass through to the next operator.

COMBINELATEST

Up until this point, we have been transforming data from a single stream that is already quite powerful. What happens if we have more than one stream and we need to combine them intelligently? We will finish off our discussion of transforming data by showing an example using `Observable.combineLatest`.

We will create two separate observables that wrap two different arrays. The users$ observable just contains a list of users, and the `items$` array a list of items; however, each item includes a reference to a user ID similar to how a relational database would. We will use this to our advantage.

```
js
import {Observable} from 'rxjs/Observable';
import {User} from './user.model'
import {Item} from './item.model';
import {UserItem} from './user-item.model';
import 'rxjs/add/observable/of';
//...
export class App implements OnInit {
  users$: Observable<User[]> = Observable.of([
    {id: '1', name: 'Victor Wooten'},
    {id: '2', name: 'Marcus Miller'},
    {id: '3', name: 'Jaco Pastorious'}
  ]);
  items$: Observable<Item[]> = Observable.of([
    {id: '1', name: 'Item 1', userId: '3'},
    {id: '2', name: 'Item 2', userId: '2'},
    {id: '3', name: 'Item 3', userId: '1'}
  ]);
  usersItems: UserItem[];

  ngOnInit() {
  }
}
```

COMBINE DATA

We can use `Observable.combineLatest` to combine two (or more) observables. The function takes observable streams as arguments, and its last argument takes a callback function that exposes the underlying data of those streams as arguments.

```
js
import 'rxjs/add/observable/combineLatest';
//...
export class App implements OnInit {
  //...
  ngOnInit() {
    Observable.combineLatest(this.users$, this.items$, (users, items) => {
      return users.map(user => Object.assign({}, user, {
        items: items.filter(item => item.userId === user.id)
      }));
    })
    .subscribe(result => this.usersItems = result);
  }
}
```

The result is that we can transform the data from any number of streams into a custom format of our choosing. In this case, we will map over our users, retrieve the items for each user, and append those items to the user object. The result is our original list of users, with an additional `items` property populated with a user's items.

USE CASE #02 COMMUNICATING EVENTS

Something happens in your application, and you need to notify one or more interested parties of the event. This is an all-too-common scenario. Traditionally, a developer would use standard events to accomplish this while keeping track of the direction of the event itself. Was it flowing up or down? In AngularJS, `$rootScope` was considered off limits for storing global state, but as an event bus it was second to none. The reason being is that we could control the direction of the events because they were always unidirectional, from parent to child. Let us see an example of this within an example service that is designed to broadcast a message to interested parties. Observables receive a new value each time a `next` event is called on it. We typically do not call `next` on the observable itself, as most observables fully encapsulate the inner workings of what makes it tick. We can, however, gain the ability to update an observable by using a subject. Fully elaborating on how subjects work is out of the scope of this article, but I highly recommend checking out the documentation to learn more. In our `MessageService`, we define the `messages` subject that will be responsible for updating the stream. We will also expose a `messages$` stream, which is the `messages` subject operating as an observable.

```
js
import {Injectable} from '@angular/core';
import {Subject, Observable} from 'rxjs/Rx';
import {Message} from '../models/message.model';

@Injectable()
export class MessageService {
  messages: Subject<Message> = new Subject();
  messages$: Observable<Message> = this.messages.asObservable();
}
```

When we need to update the `messages$` stream, we just need to call `message.next` and pass in the new value. Any consumer that is subscribed to the `messages$` stream will be immediately updated with the new value. We have just created a super powerful event bus in just a few lines of code.¬

```
js
import {Injectable} from '@angular/core';
import {Subject, Observable} from 'rxjs/Rx';
import {Message} from '../models/message.model';

@Injectable()
export class MessageService {
  messages: Subject<Message> = new Subject();
  messages$: Observable<Message> = this.messages.asObservable();

  notify(message: Message) {
    this.messages.next(message);
  }
}
```

We can then consume the service elsewhere in the application and call `this.messageService.notify` to send a new value down the event bus. A typical use case for this is when we want to notify a top-level component that something happened at a sub-component level.

```
javascript
export class ClientDetailsComponent{
  message: Message;

  constructor(private messageService: MessageService) { }

  notify(client: Client) {
    const message: Message = {
      title: 'Saving Client',
      type: 'info',
      body: `Saving ${client.name}`
    };
    this.messageService.notify(message);
  }
}
```

Within our root component, we can also subscribe to `this.messageService.messages$` and assign the value to a local `messages` property that we can bind to in our template.

```
javascript
export class AppComponent implements OnInit{
  message: Message;

 constructor(private messageService: MessageService) { }

  ngOnInit() {
    this.messageService.messages$
      .subscribe(message => this.message = message);
  }
}
```

We can take this one step further and instead of subscribing to the `messages$` stream and unpacking the incoming value, we can just create a reference directly to the stream itself. The issue is that we now have a reference to the encapsulating observable and not the internal message that matters to us.

```
javascript
export class AppComponent implements OnInit{
  message: Observable<Message>;

  constructor(private messageService: MessageService) { }

  ngOnInit() {
    this.message = this.messageService.messages$;
  }
}
```

Angular has thankfully provided a convenient way to unwrap the observable in our template using the `async` pipe. The `async` pipe will automatically subscribe, and unwrap the observable it is bound to, for us.

```
html
<div class="app-content">
  <app-message [message]="message | async"></app-message>
  <router-outlet></router-outlet>
</div>
```

The above example shows how easy it is to use observables as an event bus to communicate events within your application. I recommend that this approach be used for communication scenarios and not as a mechanism to manage complex

application states. Situations where you need to implement a toaster or growl-type interface to communicate the state of certain actions or even to control a loading indicator are good use cases for this technique.

If we were to extend this example further to include a more advanced subject called the `BehaviourSubject`, we would be very close to the `@ngrx/store` implementation of redux.

USE CASE #03 HANDLING DOM EVENTS

With a standard HTTP call, we often receive a single response, and that is that. If we stopped there, we would have a superficial understanding of observable streams and never truly understand the power behind them. The browser, on the other hand, is an incredibly rich source of never-ending events that we can capture into observable streams and harness to create engaging user experiences. To prove this point, we are going to dig into a more complex scenario that involves drag-and-drop. I have personally seen some very complex implementations of drag-and-drop, but we are going to do it in less than fifty lines of code.

SETUP

Once again, we will start with a simple Angular component that has a single div representing a ball. We create a reference to the element by adding a `#ball` attribute to the div, and then using `@ViewChild('ball') ball;` to tell Angular that we want a reference to the element. We are also going to add a `position` member to the class. The position will contain `x` and `y` coordinates for the position of the ball. Lastly, we will bind the x and y coordinates with the `style.left` and `style.top` properties, respectively, so we can easily control the position of the ball.

```
js
import { Component, OnInit, ViewChild } from '@angular/core';
@Component({
  selector: 'app-triggers',
  template: `
  <div #ball class="ball"
    [style.left]="position.x + 'px'"
    [style.top]="position.y + 'px'">
  </div>
  `
})
export class TriggersComponent implements OnInit {
  @ViewChild('ball') ball;
  position: any;

  ngOnInit() {
  }
}
```

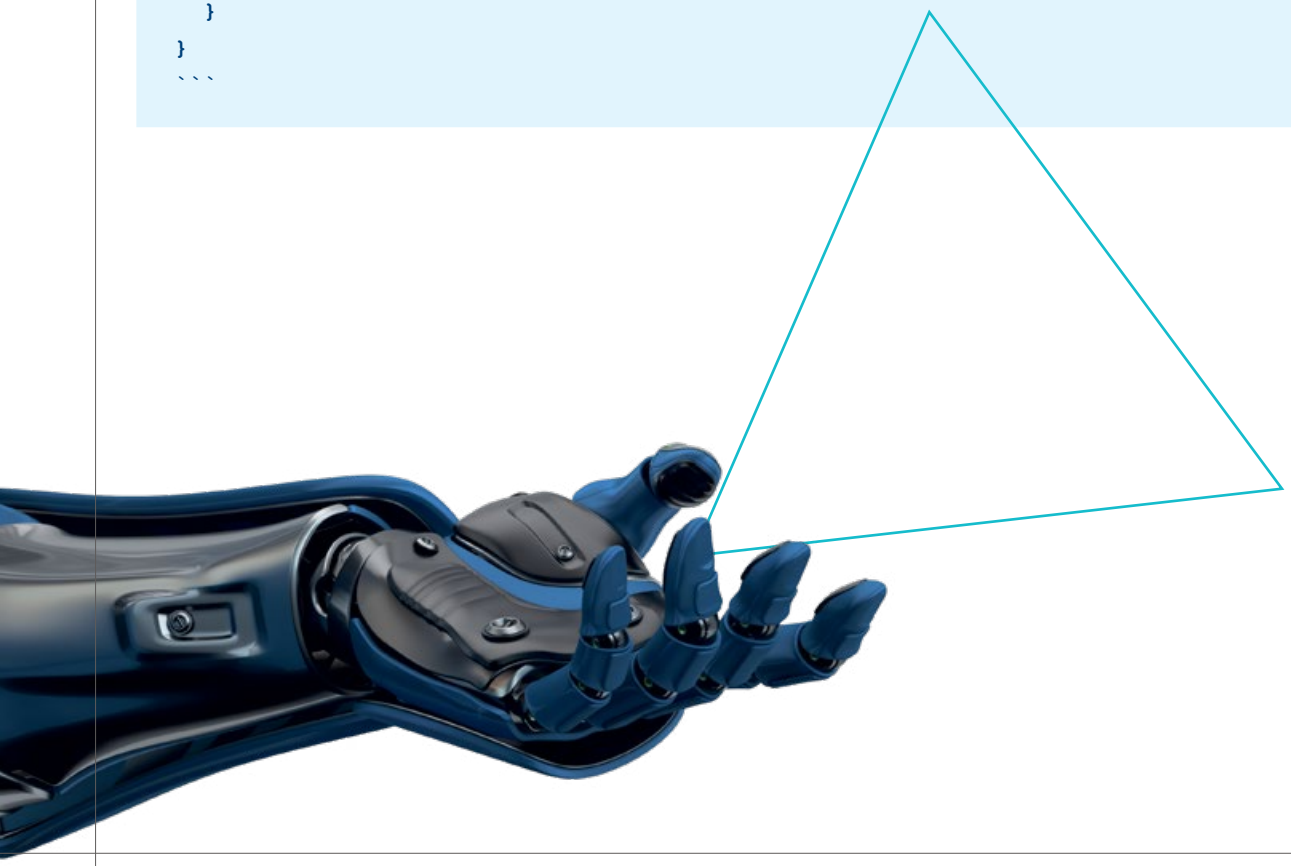
MOVE STREAM

Next, we will use `Observable.fromEvent` to listen for the `mousemove` event anywhere in the document. We then use the map operator to take each event, perform a bit of math, and turn it into an object with x and y properties.

We assign this whole stream to a constant called move$.

```js
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/observable/fromEvent';
import 'rxjs/add/operator/map';
//...
export class TriggersComponent implements OnInit {
  //...
  ngOnInit() {
    const BALL_OFFSET = 50;

    const move$ = Observable.fromEvent(document, 'mousemove')
      .map((event: MouseEvent) => {
        const offset = $(event.target).offset();
        return {
          x: event.clientX - offset.left - BALL_OFFSET,
          y: event.pageY - BALL_OFFSET
        };
      });
  }
}
```



MOUSE STREAMS

Our next course of action is to create two more streams from the `mousedown` and `mouseup` events. Since we are mainly using these streams as triggers, the only action we need to take is to set a CSS property on the ball. Since we are not modifying the observable stream itself, we can use the `do` operator to accomplish this.

We then assign the event streams to constants called `down$` and `up$`.

```
js
//...
export class TriggersComponent implements OnInit {
  //...
  ngOnInit() {
    const down$ = Observable.fromEvent(this.ball.nativeElement, 'mousedown')
      .do(event => this.ball.nativeElement.style.pointerEvents = 'none');

    const up$ = Observable.fromEvent(document, 'mouseup')
      .do(event => this.ball.nativeElement.style.pointerEvents = 'all');
  }
}
```

BRINGING IT ALL TOGETHER

Now for the real power of observables. First, let's explain what we want to happen in plain English:

1. Wait for the mouse to be pressed and held.
2. When, and only when, the mouse is pressed, track all mouse movements (and set the coordinates of the ball).
3. Track mouse movements until the mouse is not being pressed anymore.

Like many programming problems, this seems simple in English, but it can't be that easy to code, right? Wrong! Let's now translate our to-do list into observables:

1. `down$`
   (Emit mouse down event)
2. `.switchMap(...)`
   (Take the `mousedown`, and do whatever is inside the callback we supply...)
3. `move$.takeUntil(up$)`
   (Track mouse moves until the mouse is no longer pressed.)

```
js
import 'rxjs/add/operator/startWith';
import 'rxjs/add/operator/switchMap';
import 'rxjs/add/operator/takeUntil';
//...
export class TriggersComponent implements OnInit {
  //...
  ngOnInit() {
    down$
      .switchMap(event => move$.takeUntil(up$))
      .startWith({ x: 100, y: 100})
      .subscribe(position => this.position = position);
  }
}
```

For good measure, we throw in `.startWith({ x: 100, y: 100})` so that the ball starts off at the right spot on the screen. Finally, we subscribe to the entire stream and provide a callback that takes the coordinates and assigns the `position` class member appropriately.

CONCLUSION

I remember watching multiple talks on observables and RxJS and leaving with the sense that they were really cool, but with no idea how to do anything substantial with them. The goal of this article is to show you a few ways that you can start to use observables outside of managing HTTP responses. If you ever find yourself in a subscribe block and you are manipulating your response, you have probably missed an opportunity to do that same work with an RxJS operator. Likewise, if you ever wonder how to communicate that something has happened in another part of your application, using observables as an event bus is a quick and powerful way to facilitate this communication. Finally, my all-time favorite use for observables is to capture DOM events to create rich user experiences.

RESOURCES

1. RxJS Documentation
   http://reactivex.io/rxjs/

2. Angular Documentation
   https://angular.io/docs

3. Great Free Mini Course on Observables [RxJS FTW]
   http://rxjsftw.in/

4. Awesome Tools for Working with Observables in Angular [ngrx]
   https://github.com/ngrx

LUKAS RUEBBELKE

Based in Phoenix, Arizona, Lukas is a published author, award-winning developer, Google Developer expert, and Angular addict. He's also an industry educator, conference speaker, workshop trainer, and avid course producer.

CIO Finance

Gain a holistic view of how your systems work:

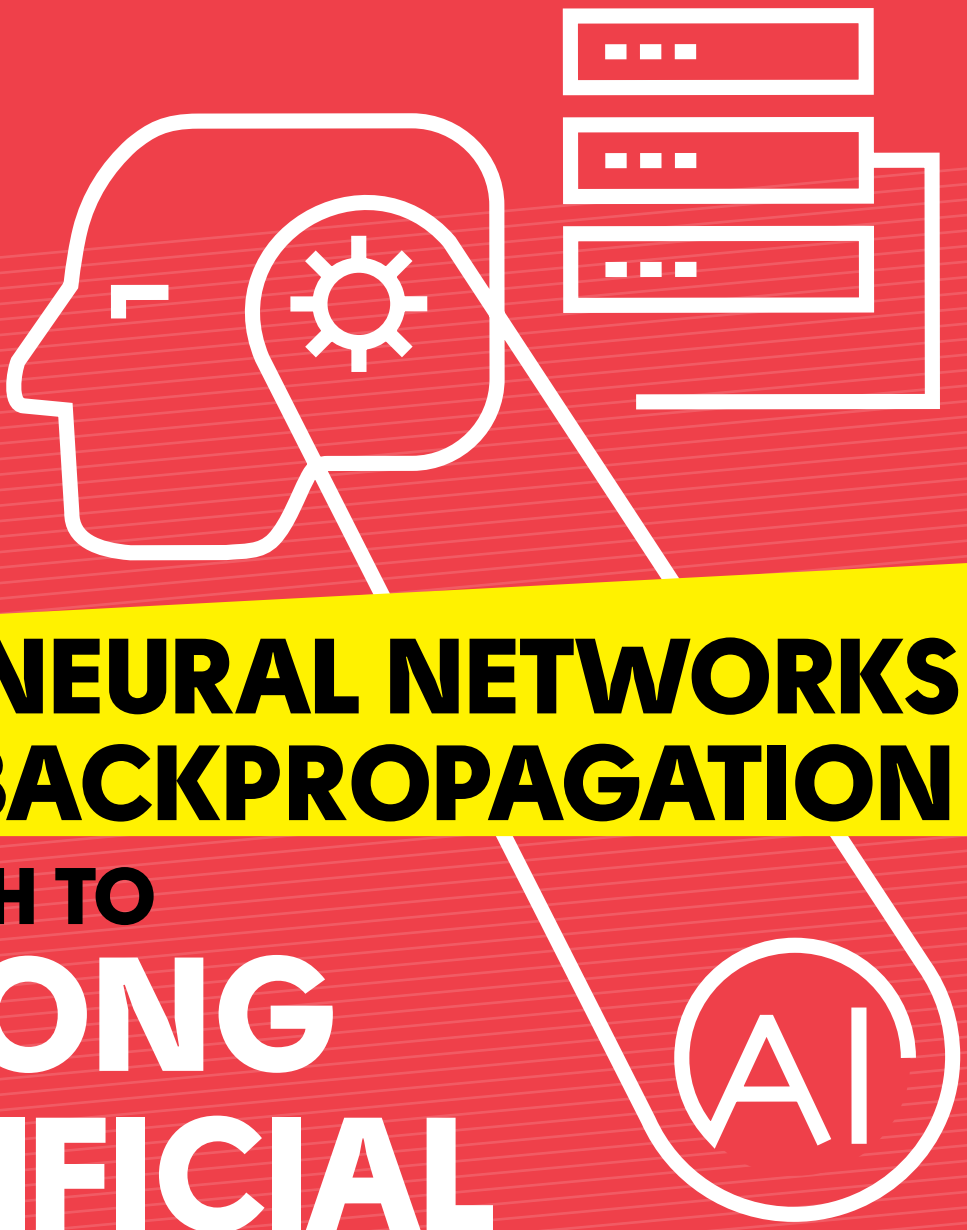
Only then will you be able to make necessary changes

Tired of short-term issues stifling long-term vision?
Read our e-book, **"Confessions of a CIO,"**
and contact us to find out how we
can help you stay ahead.

confessions.luxoft.com

BRIGHT MINDS
SHARP SOLUTIONS

LUXOFT



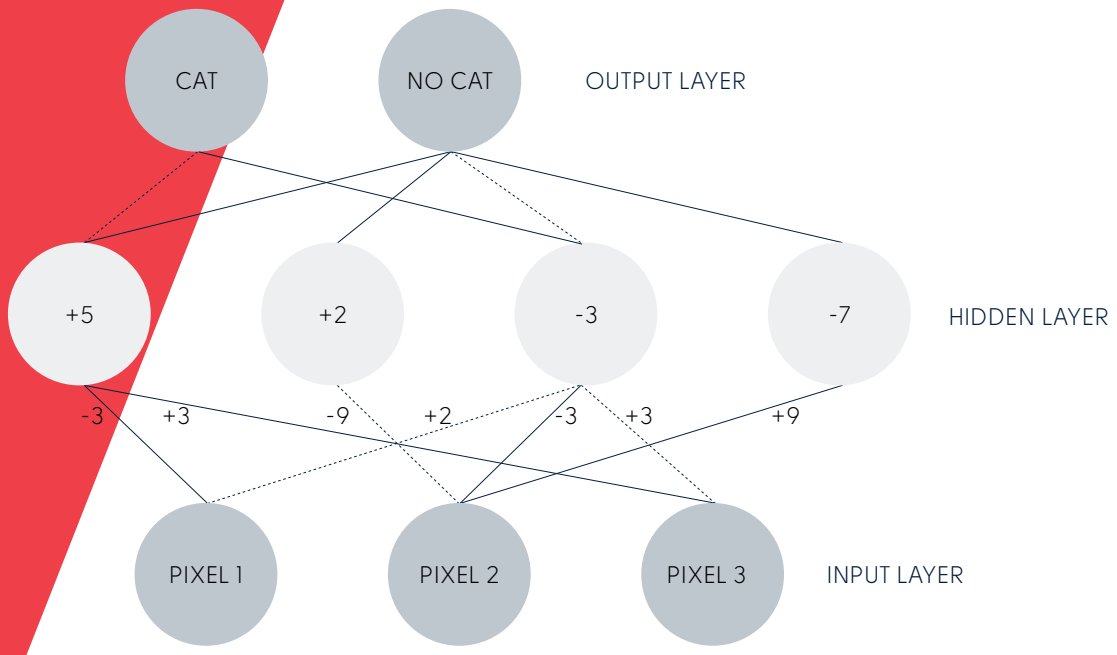
ARE DEEP NEURAL NETWORKS AND BACKPROPAGATION THE PATH TO STRONG ARTIFICIAL INTELLIGENCE?

ENRIQUE CORTEZ RELLO

Lately we have seen a great number of very useful practical applications that use Artificial Intelligence techniques. For example, machines are now very good at face recognition, machine translation, image classification, or learning how to play games. Most of these applications use Deep Neural Networks (DNN) that have been trained using backpropagation algorithms. Will DNNs take us to Strong AI, an artificial intelligence whose intellectual capabilities truly match those of a human in all domains? The answer seems to be that DNNs do not have the characteristics necessary to evolve to Strong AI.

WHAT ARE NEURAL NETWORKS? WHAT IS BACKPROPAGATION?

A neural network (NN) is an abstraction that simulates some functions of neurons. It is not a detailed biological simulation, since it only simulates strength of signals in neurons and synapses. Usually a NN is drawn as layers stacked on top of one another. The "neurons" are very simple computational units that are "stimulated" and pass this "stimulation," represented by a number, to other neurons connected to it. There is another number that represents the strength of the connection between one neuron and another neuron. This number models the strength of the synapses between neurons in the brain. A neural network has a number of layers. There is an input layer, a number of middle (hidden) layers, and an output layer.



A deep neural network (DNN) has many hidden layers. The stimuli go from the input to the output layers through the hidden ones, thus the correct name for DNN is Deep Feedforward Neural Networks.

One successful application of DNN is image recognition. To illustrate how image recognition works, suppose that we want to build a DNN to see if there is a cat in a picture. Suppose the inputs are B&W pictures with a resolution of 500x500 pixels. You feed the image by setting the stimulus level of the input to 250,000 neurons (500x500), depending on the brightness of each pixel. Then you pass the stimuli to the next layer of neurons and then to the next until you get to the final layer (output layer), which has two neurons: one represents "cat" and the other "no cat."

The idea is to train the DNN to stimulate the "cat" output neuron only if there is a cat in the picture. That is when backpropagation comes into the picture. Backpropagation is analgorithm used to train a DNN. A paper by G. Hinton and two of his graduate students[1] showed that DNN trained using backpropagation could beat all other image recognition systems at that time.

Although training an NN is still a laborious process, until backpropagation was discovered, it was very difficult.

Backpropagation is relatively simple, but only works well with large quantities of training data. In this case, that would mean millions of 500x500 pictures, some with cats and some without cats. Of course a human would need to label these pictures as "cat" or "no cat."

How does back-propagation (backprop) work? Suppose the training picture is a dog. You convert the intensities of each pixel into a number and you feed the input 250,000 neurons, and then propagate the stimuli through the multiple layers, until you get to the two output neurons. Since this was the picture of a dog, ideally the neuron that represents "no cat" should have a higher stimulus number. But suppose that is not the case, and the training image has been wrongly classified as a "cat." Backprop is an algorithm used to change the strength of every connection in the network (synapses) to fix the error for a given training example.

Backprop starts with the two output neurons (the output layer), and figures out the difference between what the stimuli numbers should have been, and what they actually were – i.e. the contribution of each neuron to the error. Then the previous layer must be looked at to examine the contribution of each neuron to the error, and so on. Backprop propagates the errors backwards through the layers of the network. The remarkable thing is, once millions of examples have been fed to it, the DNN becomes pretty good at determining which images have cats. Although Backprop was described more than thirty years ago, the ability to use it in a practical way is recent, due to new computational techniques, availability of data, and computing power.

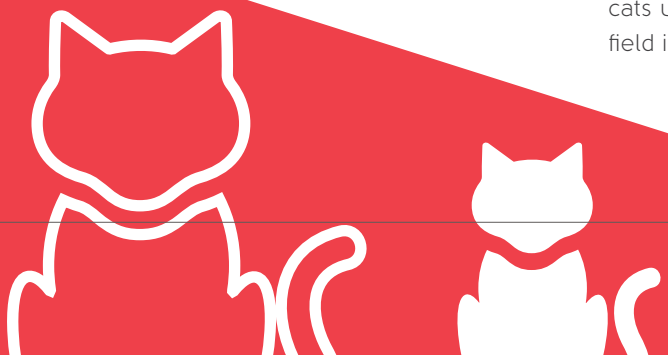
If you think about it, DNNs are not very intelligent. They are pattern recognizers that cannot explain how they classify items. Also, although DNNs can find pictures of cats, they cannot explain what a cat is, or even explain why the image is a cat. DNNs represent a very narrow kind of intelligence: the ability to classify things. What is interesting is that this can be anything: text, images, video, sounds or anything that is digital. From a mathematical point of view, what is happening is that DNNs are taking these things (images, sounds, text) into a high-dimensional vector space.

WEAK AI AND STRONG AI

The ultimate goal of AI is to create the so-called **Strong AI**, a general AI that is functionally equal to human intelligence. So are DNNs the path towards strong AI? Probably not, because they represent a very narrow type of intelligence. Some authors call what we have today **Weak AI**.

Human babies do not need millions of examples to recognize a visual concept. Babies see a few cats, they learn the word "cat" and from then on, they recognize live cats, cats in movies, cats in pictures, the sounds of cats, what cats eat, and many other things. Human babies are efficient and fast, and make sense of new concepts in terms of concepts that they already understand. Babies create a model of how the world works, and they refine it over time. Babies can see a cat, touch a cat, smell a cat, hear a cat, see a cartoon of a cat, and understand that cats and tigers have something in common. Our brains build knowledge on top of previous knowledge, and our intelligence does not "break" if we change the objective or the domain of our present task. Our intelligence is not brittle; it is adaptable, curious, and resilient. We also possess intuition, creativity and something very special: common sense.

There is a different approach to do AI: Symbolic AI, based on mathematical logic. A big advantage of symbolic AI is that, since its basis is mathematical logic, all conclusions and decisions can be explained. However, Symbolic AI does not seem to be a path towards Strong AI either. There are purely symbolic approaches to a more general intelligence, such as, for example, the CYC project. The CYC project, led by Doug Lenat[2], has been building a "common sense" database and taxonomy that could be used in intelligent applications, but this task needs significant human intervention and needs to be updated constantly. For example, you would need to record into some type of database information about cats, such as that cats, lions and jaguars are related; that cats are domestic animals; that cats have great vision; that cats have whiskers, claws, and pointed ears; that domestic cats use litter boxes; "Cats" is a musical, and that Garfield is a cat. Once you know that and more about cats,



you could answer questions about cats, make inferences, and in general understand what a cat is. In practical terms, most recent breakthroughs of AI have not used a symbolic approach.

THE WAY FORWARD

Neither DNN nor Symbolic AI approaches seem to be the path to Strong AI. There are researchers working on novel approaches using what we've learned from the study of the brain and cognitive science, such as the Center for Brains, Minds and Machines at MIT, which brings together the computer, cognitive and neurological sciences. Some people believe that the shortest path is Biological Computation, defined as the use of biological systems to build computers. Perhaps the path is a better digital simulation of the brain functions. The brain has 86 billion neurons, but the real problem is that we still do not have deep understanding of how the brain works. There are significant research efforts to try to understand the brain, like the Human Brain Project, co-funded by the European Union[3]

We do not seem to be even close to Strong AI, contrary to many articles in the general press[4]. But the field of AI clearly has advanced recently especially in the refinement and application of DNNs to problems that seemed very hard just a few years ago. This is an exciting time to be working in AI.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, G.E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks" https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[2] W. Knight, "An AI with 30 Years' Worth of Knowledge Finally Goes to Work" , MIT Technology Review, March 14 2016 http://www.cyc.com/wp-content/uploads/2015/06/CycToIntel.pdf

[3] The Human Brain Project https://www.humanbrainproject.eu/en/

[4] L. Whitney, "Are Computers already Smarter than Humans?" Time Magazine, September 29 2017 http://time.com/4960778/computers-smarter-than-humans/

ENRIQUE CORTEZ RELLO

Enrique Cortes Rello is the Managing Director of Luxoft Mexico. Enrique is a seasoned executive with more than 20 years' experience developing and managing IT companies in Europe, Asia and Latin America. Enrique moved to Luxoft from Wipro Technologies, where he was the Geo Head of the North Latin America region. As a general manager, Enrique was in charge of sales, delivery and government relationships in Mexico, Central America and Colombia. Prior to Wipro, Enrique worked for 14 years at Perot Systems, acquired by Dell in 2009. While at Dell/Perot, Enrique managed operations and developed business in the UK, Philippines, India and Mexico. Enrique has been VP of IT/BPO at CANIETI, and member of the board of directors at CADELEC. He also has been a member of the advisory board for the School of Engineering at Tec de Monterrey and ITESO. Enrique also is an advisor to the Secretary of Economic Development of the State of Jalisco (Mexico) on attracting foreign investment in the information technology field. Enrique has a bachelor degree in Computer Science from Tec de Monterrey in Mexico, and a master's degree in Computer Science from Arizona State University. He is also a Ph.D. candidate in Math and Computer Science at Arizona State University. He also attended the "Leading Professional Service Firms" program at Harvard Business School.

HEALTHCARE & LIFE SCIENCES

TOGETHER YET SEPARATE IN THEIR TECHNOLOGICAL JOURNEY

SAM MANTLE

You see them everywhere – Fitbits, wearable trackers, and other health and diagnostic applications – that make monitoring your health easy and manageable. And in your doctor's office, you find automatic thermometers and digital blood pressure devices that are more efficient, accurate and more connected than their manual counterparts. While heading in the right direction, these devices are only the first step toward a truly digital Healthcare and Life Sciences ecosystem. To that end, Luxoft works with large and small clients at the forefront of this digital transformation. We understand the challenges of adopting new technologies across these two industries, and are helping our clients embrace the opportunities available by engineering transformational business solutions.

TWO INDUSTRIES, TWO PRIORITIES

Healthcare and Life Sciences are two large industries with two distinct priorities, yet are converging now more than ever through patient empowerment and advances in technology.

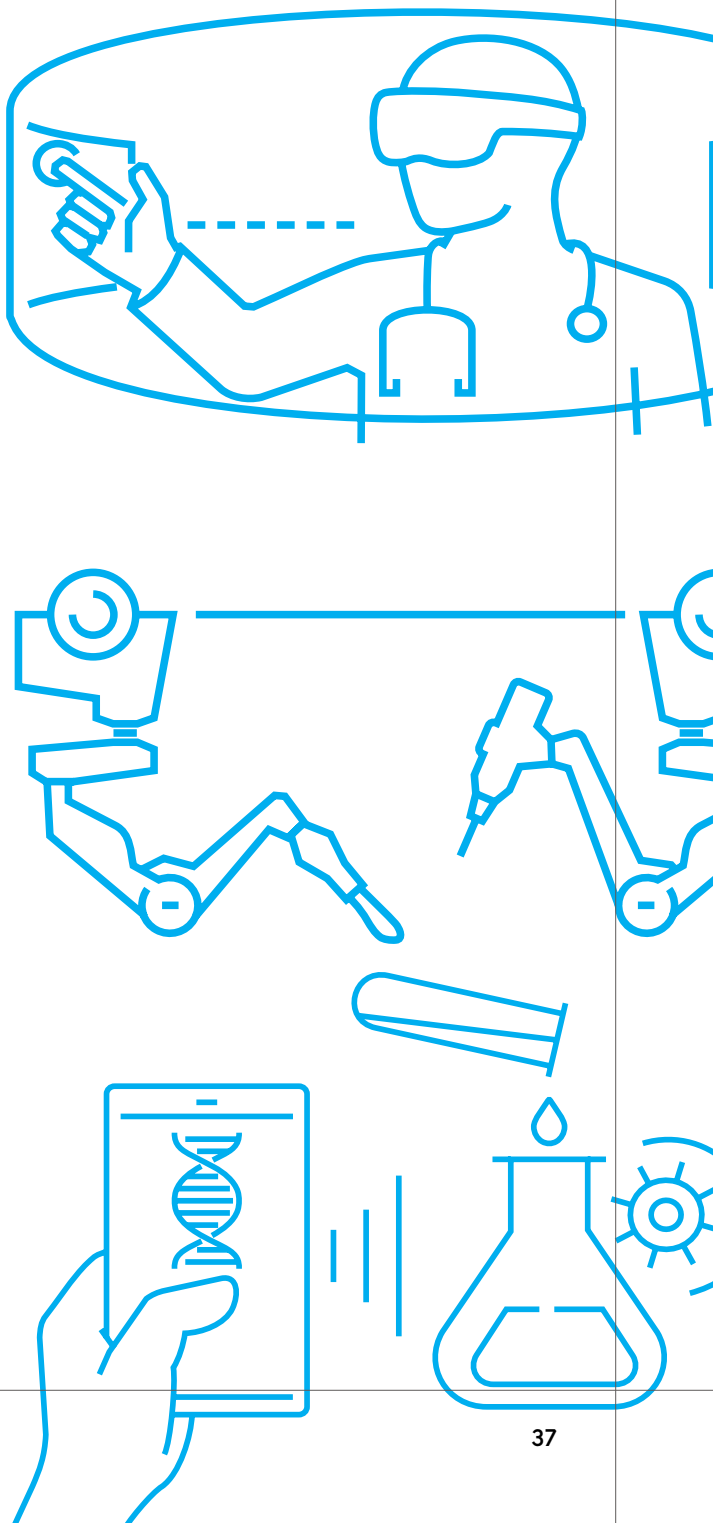
On one hand, the patient has traditionally been at the center of the Healthcare ecosystem. This ecosystem includes health insurers (also known as payers), hospitals and clinics (also known as care providers), as well as regulators and governments. And when it comes to technology, Healthcare is largely dependent on manual processes and siloed electronic data records. For instance, just think about when you go see the doctor – the front desk typically pulls out your physical file of your medical information and hands it off to whomever needs it. While connected electronic health records around the world would be great, the industry just isn't quite there – yet.

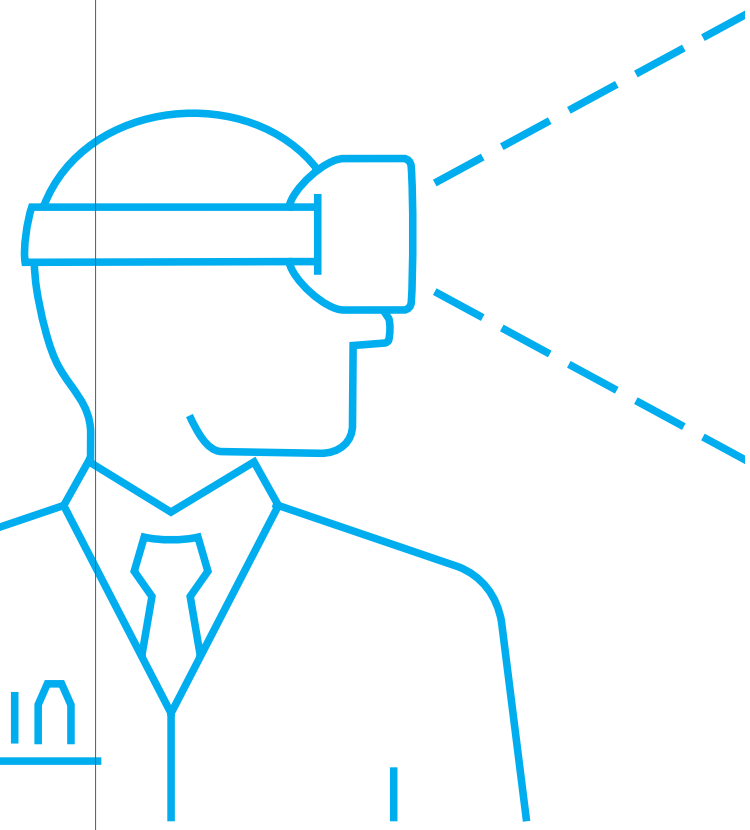
On the other hand, the drug has traditionally been at the center of the Life Sciences ecosystem. This includes large pharmaceutical and smaller biotech companies who discover, manufacture and commercialize new medicines, as well as medical device companies and CROs (clinical research organizations) that specialize in running clinical trials to get new compounds to market. Life Sciences companies, like their Healthcare counterparts, operate slowly with minimal automation, cumbersome processes and fragmented data sets. For instance, initial drug discovery, research and conducting lengthy clinical trials often takes a decade to get one drug to market. But one has to wonder: with more automation and more intelligent insights gleaned from integrated internal and external data sets, could that lengthy amount of time be shortened or even cut in half?

HOW ARE THESE TWO INDUSTRIES CONNECTED?

There are increasingly stronger links between the patient and drug ecosystems. Pharmaceutical companies are interested in getting deeper insights into the patient populations they serve, and Healthcare insurers are anxious to know that their members are being helped by the therapies they prescribe. Together, they are incentivized by ensuring that patients get faster access to therapies they need to maintain their health.

Perhaps more importantly, both industries are realizing is that, through increased access to health-related data and technologies, we experience a paradigm shift from "sick care" to "healthcare" – careful management of our health so that we do not become patients at all. The truth is taking preventative measures via modern data monitoring is beneficial for both the patient and the industry. Helping prevent or lessen the symptoms of diseases before they happen or become severe ultimately keeps patients out of the hospital, where the cost of care increases dramatically. For these reasons, the convergence between the Healthcare and Life Sciences industries is accelerating at an incredible pace, and we are seeing many solutions have an applicability across the entire ecosystem.





WHAT MAKES LUXOFT SPECIAL?

Luxoft addresses the digital challenge of Healthcare & Life Sciences from a holistic perspective – combining technologies, experience and knowledge that have been acquired across multiple domains. This unique approach allows industry and technology experts to deeply understand a desired business outcome and then design and implement necessary processes and digital assets, making the outcome a reality. All of this is done in a flexible, collaborative approach with a truly agile mindset and delivery. If you would like to learn how we can help your business flourish through applying digital transformation capabilities (such as IoT, AI, Big Data, Dev Ops and Cloud), please contact us by clicking here!

Are you looking to bring your business into the digital age? How can the Healthcare and Life Sciences industries do it? In my next article, I will expand on the technology issues these two industries currently face and how to solve them.

Sam Mantle
Managing Director of Healthcare & Life Sciences
Luxoft

SAM MANTLE

Sam is a dynamic global business leader, technology executive & CIO who is passionate about leveraging technology to realize business value and high performance in Healthcare, Life Science and data-driven companies.
Sam has a strong track record of business partnership, digital transformation, successful IT delivery and talent development across geographies and functional areas. He is equipped with an agile, collaborative and entrepreneurial mind-set combined with the willingness to challenge his team, peers and business executives to achieve ambitious goals.

Main areas of expertise include:

-Large organizational transformation
-Business process transformation enabled by technology
-Digital selection, investments and transformation
-IT Program Management
-ERP (SAP), Cloud enablement & migration
-M&A, Divestitures, TSAs
-Big Pharma, OTC, Vaccines, Biotech, Biosimilars, Generics
-Drug Discovery & Development, Medical, RWE

BLOCKCHAIN DISTRIBUTED LEDGER TECHNOLOGIES

ALEKSANDR KOPNIN

The digital world continues to change every day. Market leaders work in innovation laboratories to develop next-generation technologies for shaping the possibilities of tomorrow. As one of these leaders, Luxoft pushes innovation forward in the fields of AI, Computer Vision, Big Data and Blockchain Distributed Ledger technologies (DLT). As the IT market rapidly grows, it becomes more complex and primed for innovation. In the past two years, the focus has changed from waiting for progress to the active digitalization of industries. Now, world-famous IT providers continue to look for new opportunities to integrate modern technologies to decrease network latency, increase security, minimize cost and improve data management.

The big challenge is a lack of practical experience. There are no real usage examples, or reliable benchmark forecasts. New platforms tend to be slow and unstable inspite of the fact that IT providers strive to create production-ready products. Our team has worked on Blockchain technology for more than two years, yet we still face challenges with choosing the appropriate platform for real projects. This article is based on Luxoft's practical experience using DLT in the IoT, Healthcare, and Travel industries. It highlights performance, scaling and security aspects of Hyperledger Fabric and R3 Corda.

INCOMING REQUESTS

We at Luxoft use modern technologies to solve long--term issues. We recently received an incoming request with technical requirements for a shared ledger between high-volume batch processing systems and realtime point-of-sales transactions.

1. Reconciliation of transactions during the batch process have to achieve 130 000tx/2hrs (== 20tx/sec). Meanwhile, additional online requests may appear in the system.

2. Data needed be isolated between competing parties but shared between those with business agreements.

3. The client was interested in using DLT to help solve their problem.

FABRIC DESIGN

TEST SCENARIOS:

Hyperledger Fabric v1.0.0: Private Blockchain, using channels for data isolation. Orders were configured to generate a new block every 10ms.

INVOKE operation

- **CLIENT** sends request to all **PEERs** and accepts their replies.
- **CLIENT** sends the endorsed packets to all orderers.
- Each orderer sends the transaction to a single random **KAFKA** node.
- **KAFKA** node sends the request to leading **ZOOKEEPER** node.
- Leading **ZOOKEEPER** node ensures the message order and delivers the messages back to **KAFKA** node in a predetermined order.
- **KAFKA** node delivers the TX back to the **ORDERER** node in a determined order.
- **ORDERER** node notifies **PEER** nodes of a new blockchain state.
- **PEER** node notifies the **CLIENT** node of successful transaction completion.

QUERY operation

Sends the request to a **PEER** and accepts the reply.

Testing

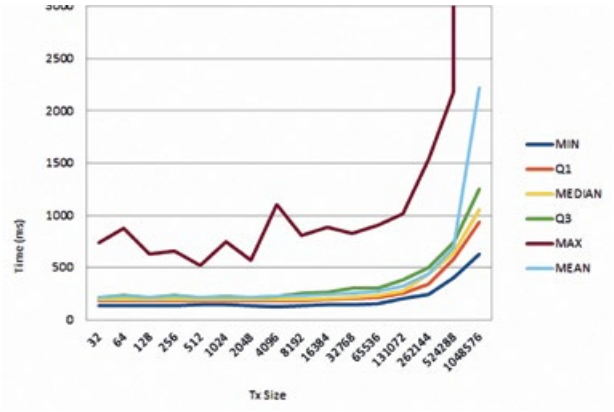
To measure performance, we prepare benchmarks and run tests with the following configuration:

NETWORK TOPOLOGY	HARDWARE CHARACTERISTICS	NUMBER OF VALUES	COMMENTS
1 Peer + 1 Orderer + 1 Solo	Intel(R) Core(TM) i3-3220 CPU@ 3.30GHz, 16GB Ram	1024	All in one local machine
3 Peers + 3 Orderers + Kafka	1 VCPU, 2GB Ram, Disk: 20GB	Min: 209 Max: 216	1 VCPU, 2GB Ram, Disk: 20GB
3 Peers + 3 Orderers + Kafka	1 VCPU, 2GB Ram, SSD: 20GB	Min: 848 Max: 940	AWS instances with multiple regions

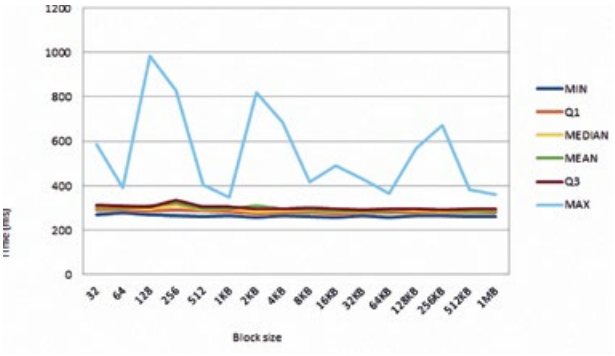
List of tests:

1. ISSUE TX

The application sends the transaction to the smart contract, consisting of pairs (key[keySize], value[valueSize]). Each key and value has a specified size and is filled with random bytes. The smart contract doesn't change the storage and is just an empty function, so it doesn't have either a reading or a writing set.



Picture 1. Local machine

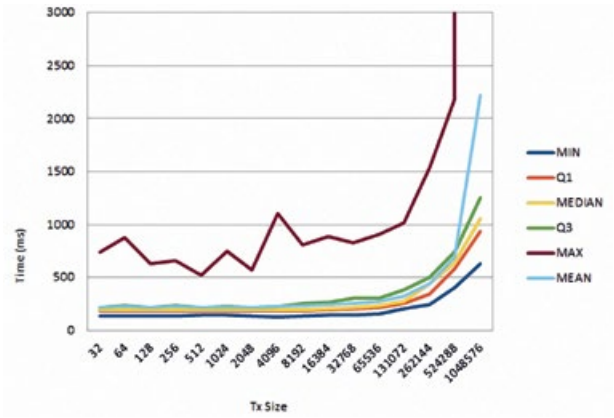


Picture 2. Cloud instances

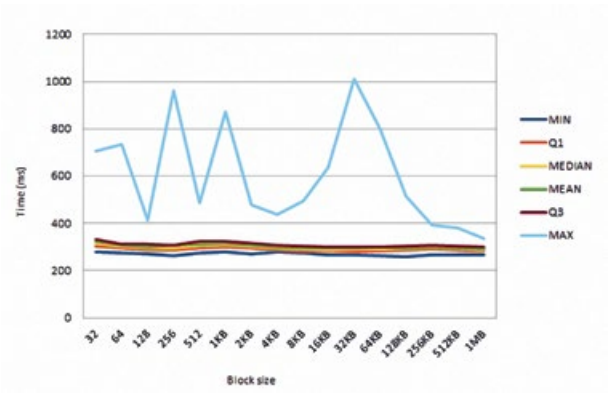
Picture 3. AWS Regional instances

2. WRITE STORAGE

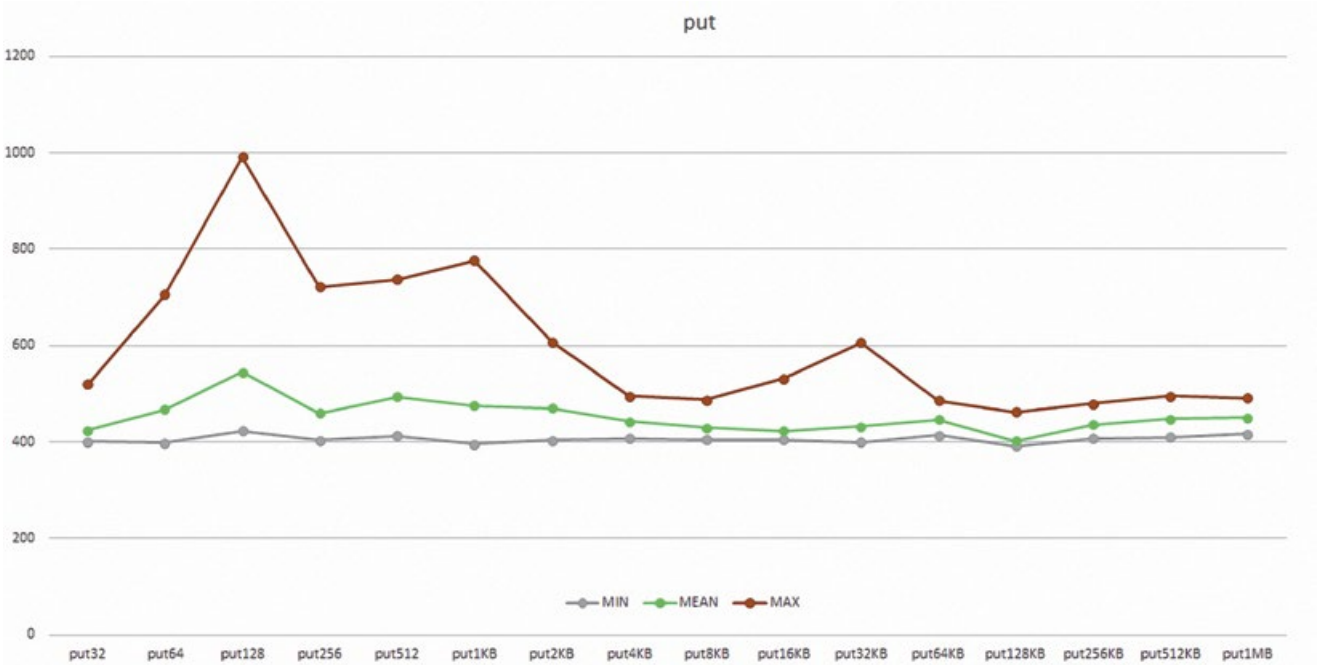
The application sends the transaction to the smart contract, consisting of pairs (key[keySize], value[valueSize]). Each key and value has a fixed size and is filled with random bites. This way, the keys are ensured to be unique. The smart contract saves these keys and values to the blockchain storage.



Picture 4. Local machine



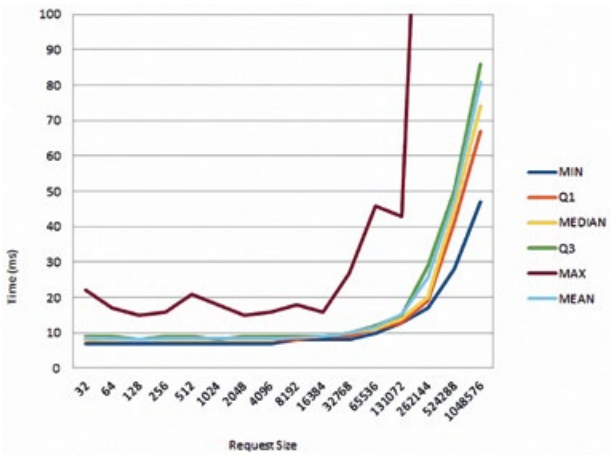
Picture 5. Cloud instances



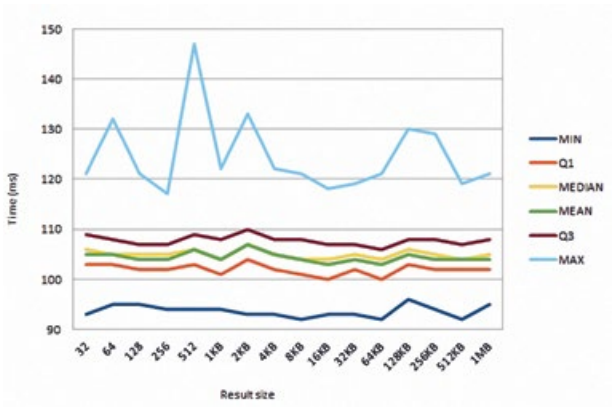
Picture 6. AWS Regional instances

3. READ STORAGE

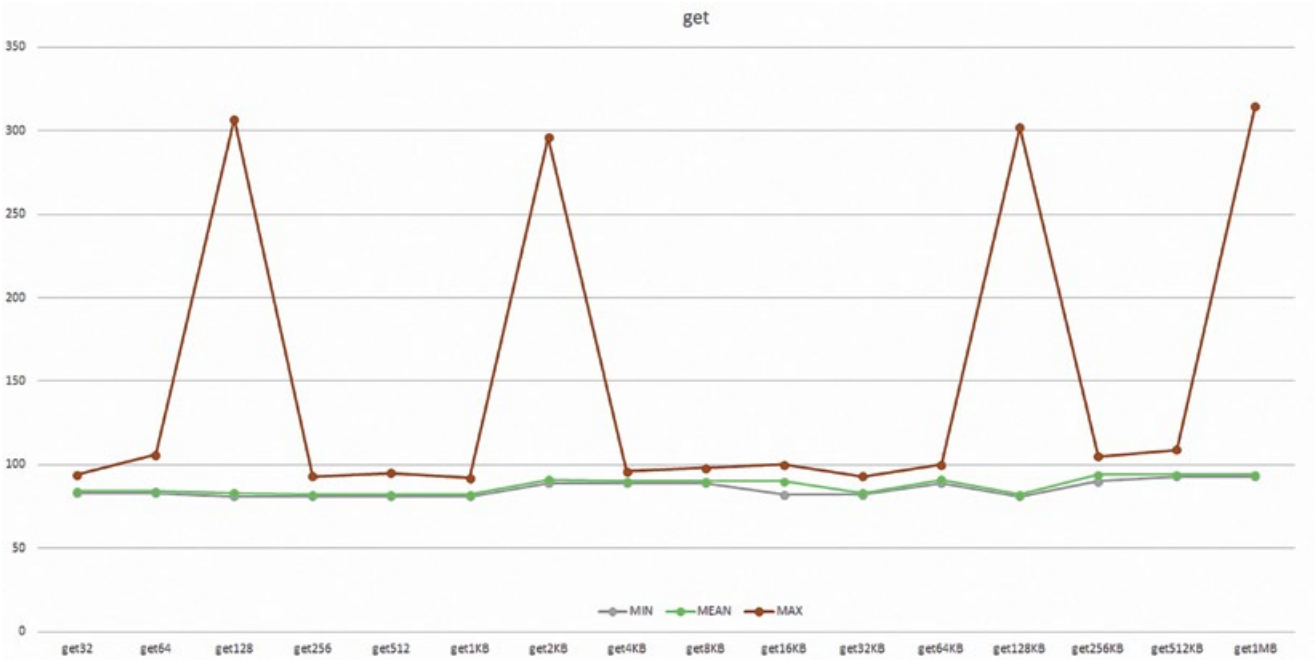
The application uses a smart contract method to query data, as added by the "TX with Storage" test. Each key and value has a fixed size and is filled with random bites. This way, the keys are kept unique. The smart contract queries the value, associates it with the key and reports it back to the application. Please note that this operation is only reading the data – it doesn't generate any change to the ledger, and as such is served by a single peer.



Picture 7. Local machine



Picture 8. Cloud instances

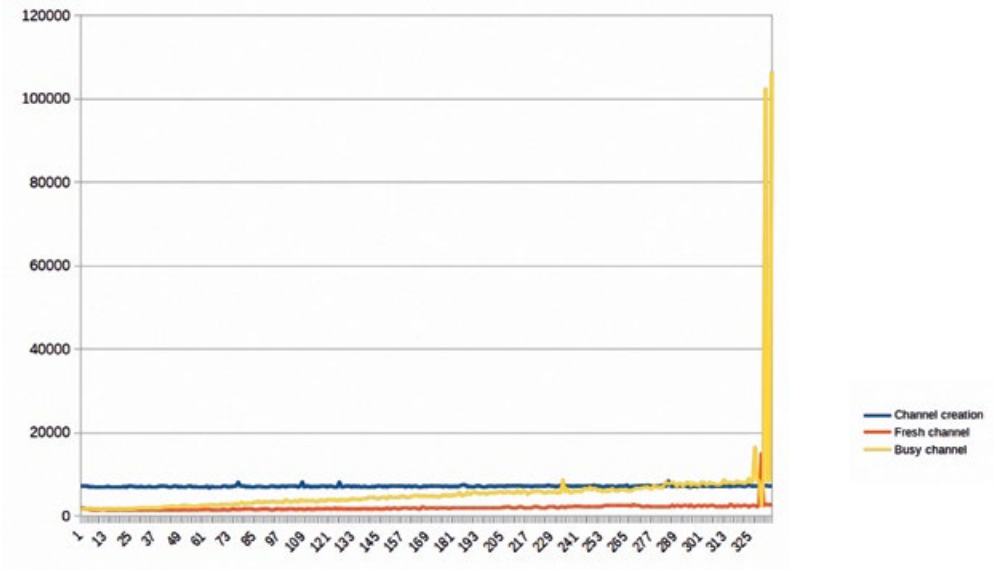


Picture 9. AWS Regional instances

Also, we run tests to estimate the maximum number of channels.

The channels tests:

NETWORK TOPOLOGY	HARDWARE CHARACTERISTICS	LOAD
1 CA + 2 Peers + 3 Orderers + 4 Kafka + 3 Zookeepers	2GB Ram + 1 VCPU	10 new transactions to each fresh channel; +10 new transactions to the first of each fresh channel



The graph above is the result of experiencing 325 channels system fails and timeout errors from the Orderer side. Also, read and write operations depend on the channel's number: if the channel's number increases, then performance typically decreases.

CORDA DESIGN

TEST SCENARIOS:

Corda by R3: P2P for data Isolation. Each transaction can be signed by different set of participants. We simulated 2 scenarios:

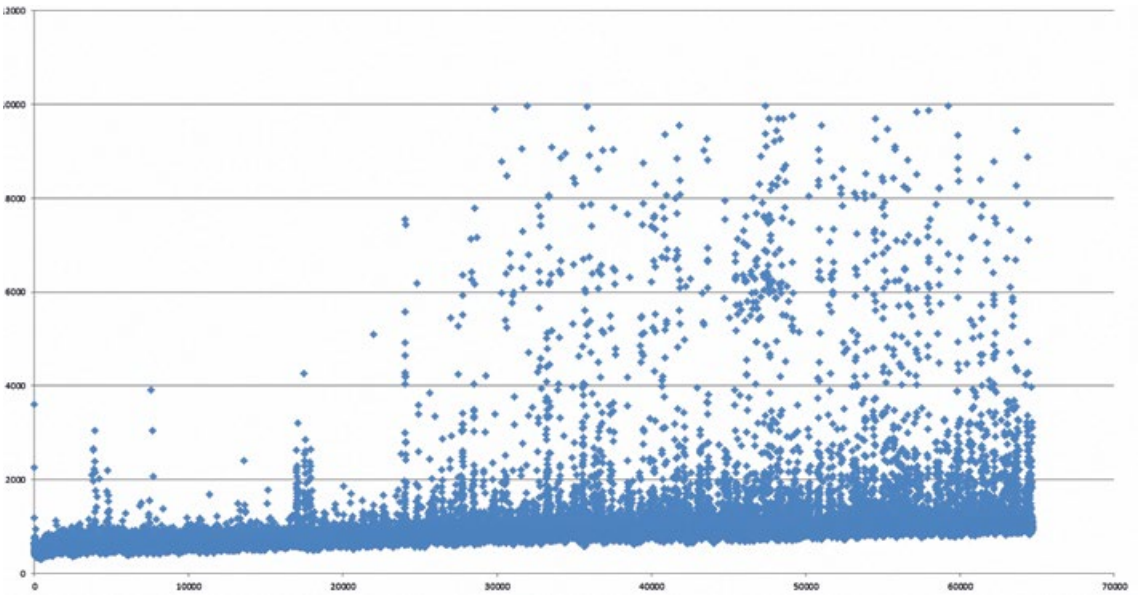
1. Single signature: only the transaction owner has to sign before distribution.

2. Multiple signatures: three participants have to sign off on the transaction before distributing it.

INVOKE + QUERY operation

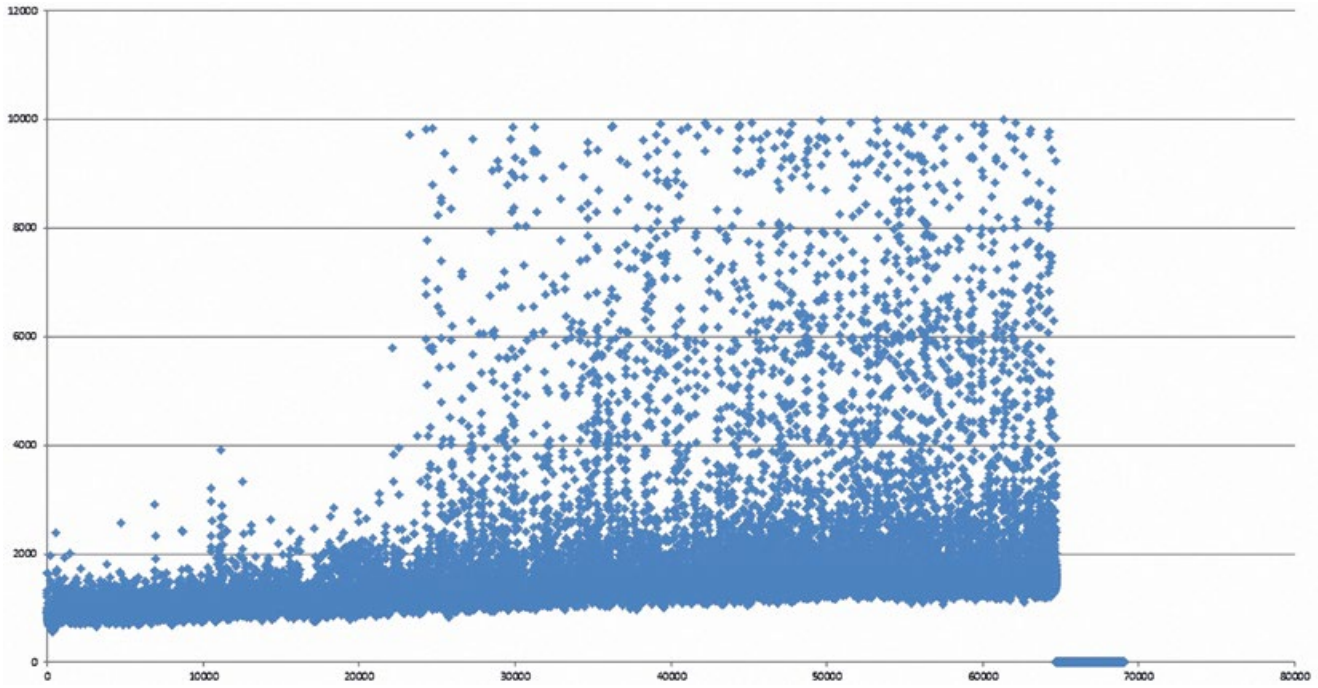
- Submit new update transaction
- Read state
- Flow invocation
- Gather signatures of all participants (depends on scenario)
- Write state
- Transaction distribution

NETWORK TOPOLOGY	HARDWARE CHARACTERISTICS	LOAD	ADDITIONAL CONFIG	COMMENTS
3 Peers + 1 Notary	3 Peers + 1 Notary	65K	-Xmx1536M	Owner's signature only
3 Peers + 1 Notary	3 Peers + 1 Notary	65K	-Xmx1536M	Three signatures



Write degradation: 400ms up to 700/1000ms

Read degradation: <= 10ms up to 400/600ms



Write degradation (signing + distribution + storage): 700ms up to 900/1200ms

Read degradation: <= 10ms up to 400/600ms

PS: ActiveMQ is involved in the transaction's signings and distributions (a possible reason for the high data spread seen above).

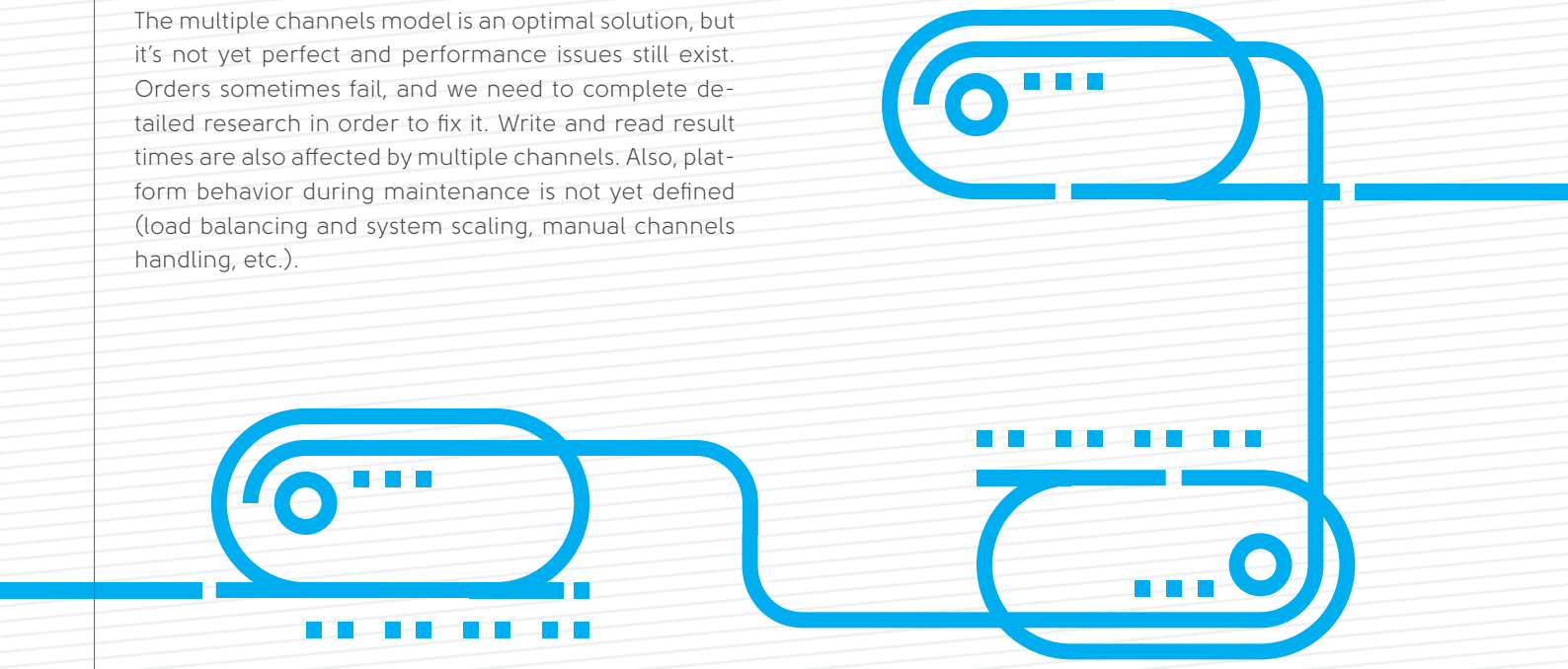
CONCLUSION

The fabric and corda results were different during the tests, so let's analyze the outcomes.

Fabric demonstrated good performance results for a distributed topology. Peer scaling only affected the system minimally. Reading < 200ms per per transaction can be achieved in a cluster model with a single channel. Similarly, writing < 350-400 ms can also be achieved in a cluster model with a single channel. The multithreaded update reached 50ms per record.

The multiple channels model is an optimal solution, but it's not yet perfect and performance issues still exist. Orders sometimes fail, and we need to complete detailed research in order to fix it. Write and read result times are also affected by multiple channels. Also, platform behavior during maintenance is not yet defined (load balancing and system scaling, manual channels handling, etc.).

Corda offers a good opportunity for data isolation by P2P communications. However, having an H2 database highly limits the platform. This timing results in read operations degrading as the data load increases. The platform has a flexible design and allows change db, but this change isn't achieved easily at this time. Finally, the Multi-sig model decreases performance significantly.



ALEXANDER KOPNIN

Alexander Kopnin has a master's degree in Computer Science with more than five years' experience. As a lead designer in Luxoft Digital, he oversees a team that specializes in solutions based on blockchain and decentralized platforms.


LTS
LUXOFT TECHNOLOGY SERIES
Meet up & network
with the best of the best
LTS is a series of free online events
featuring famous IT lecturers with rock star
status in the IT community. LTS webinars are
designed for IT professionals who are keenly interested
in the latest trends and want to attain expert-level knowledge.
https://career.luxoft.com/lts-luxoft-technology-series/
BRIGHT MINDS
SHARP SOLUTIONS
LUXOFT

LESSONS WE CAN TAKE FROM SKYDIVING FOR SOFTWARE DEVELOPMENT

MIKHAIL DRUZHININ

Now it is official. I have my skydiving "A" license, which allows me to jump out of an airplane on my own. Today I want to share with you what this interesting sport has to offer people who run IT projects.

First, let's figure out what IT projects and skydiving have in common.

SKYDIVING:

1. Has an objective and planned outcome
2. Is limited by time and resources (altitude)
3. Requires upfront planning
4. Can lead to serious issues if mistakes are made during execution
5. Involves having almost no control over environment
6. Depends greatly on those around you: they can either significantly contribute to success or cause failure in everything

Kind of sounds like an IT project, right? With one small difference, of course: a skydive lasts around five minutes and an IT project last five months or, if you're unlucky, five years.So let's come back to the point what leads to a successful ~~skydive~~ project in order to come back in one piece.

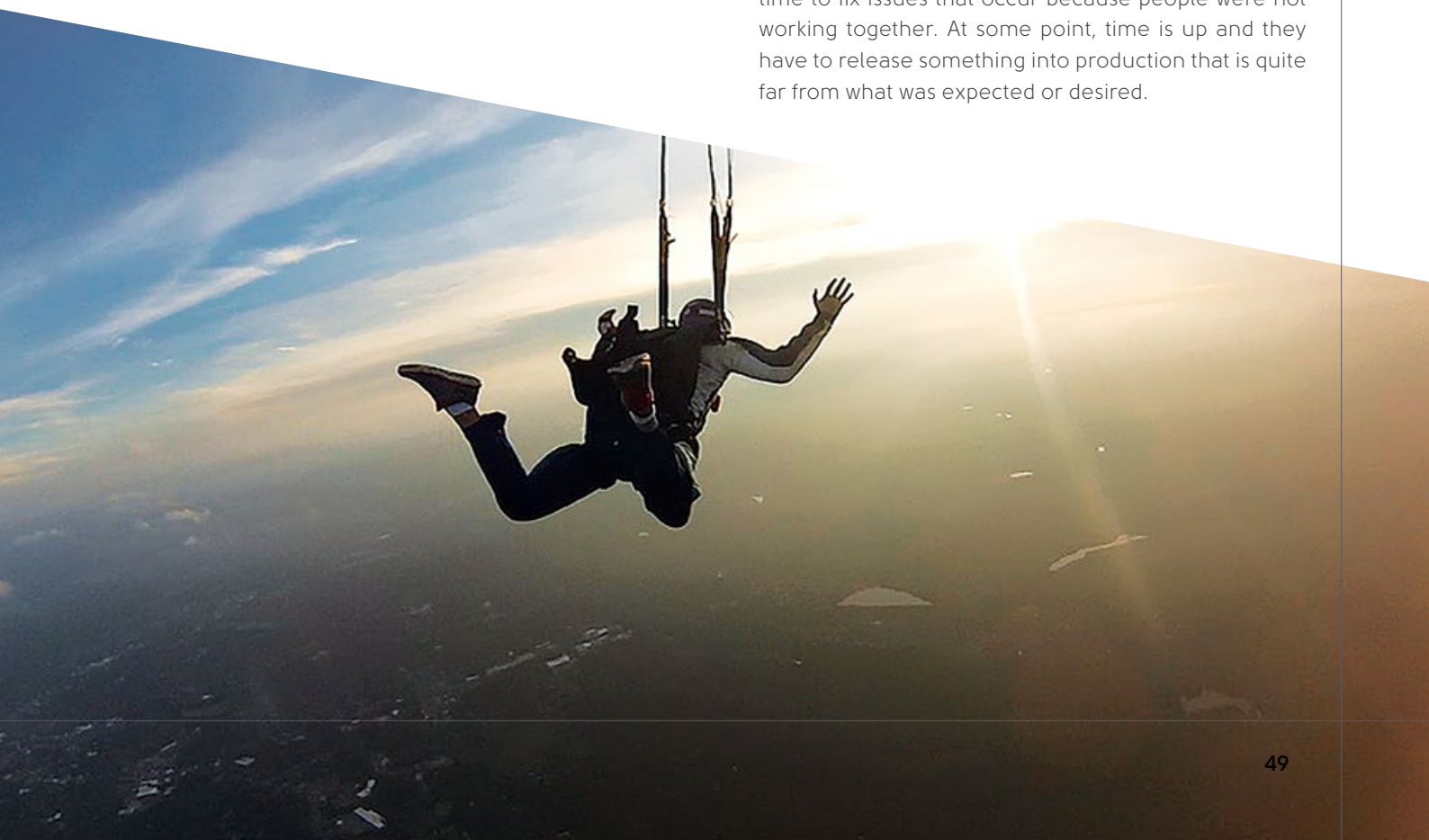
YOU NEED TO KNOW THE ENVIRONMENT AROUND YOU

This includes weather, landscape and who is with you in the airplane. In IT, a common mistake is for managers at different levels to start doing something without taking a moment to collect information about the current state of the organization or system, what is going on and why, and what is the main business direction. Unfortunately, the environment is usually out of our control and we have to adapt to it. Ignoring this simple fact can cost significantly in terms of resources and maybe even failure of project goals.

PLANNING IS ESSENTIAL

Based on information from the environment and what you want to do, you need to plan all phases of the jump, from exit to landing. You're probably not doing it alone, so everybody in your group needs to understand who is doing what. The failure to plan and communicate in a group leads to a mess during execution. Keep in mind that time is very limited and if something goes wrong you will need to spend additional time to fix it instead of doing what you planned to do. In addition, if you fail to check the weather and didn't prepare accordingly, you can end up landing in somebody's backyard (or tree) instead of the designated landing zone.

These issues arise in IT projects, too - people spend time to fix issues that occur because people were not working together. At some point, time is up and they have to release something into production that is quite far from what was expected or desired.



SAFETY FIRST

You do a gear check when you preparing for a dive, you do it again before boarding and once again in the aircraft. Also you go through emergency procedures at least a couple of times in your head. Of course, you have a reserve parachute which is checked every 180 days, and if you're smart enough, you checked the map to know where your reserve landing zone will be.

Now let's have a look at production systems execution:

- Who knows what to do if production servers go down? Are people trained for it?
- How often did you check to see that the business continuous plan works for real? Do you even have backups?
- Did you ever try out your deployment rollback procedure? Do you check it every time before deploy? Do you have one?
- Do you know what to do if your surrounding environment changes, increasing request time to them in several times?
- Do you monitor what is going on and are you able to react before an issue becomes critical?

I think the majority of IT projects fail on all of these questions, even though a day of production system downtime can cost to your organization significant revenue that could be critical for the business or even kill it.

YOU SHOULD KNOW YOUR PRIORITIES

Priorities for a jump sound simple:

1. You should open your parachute.
2. Open the parachute at the proper altitude.
3. Open the parachute in a stable position.

But these simple rules give you the ability to act fast when you need to and not lose time and altitude trying to do things perfectly when you just need to open the chute. In IT projects, priorities may not be so simple but from what I see, people either don't have them at all, or they are so vague that it makes it hard to make a simple choice between two JIRA tickets.

Worse yet - everything is a number one priority and you're supposed to do everything. In such cases, with a few changes in the environment or a few obstacles, the team has no other option but to work overtime or move release dates forward.

TEAMMATES ARE KEY TO SUCCESS

In skydiving, when time is limited, good teamwork is vital to achieve great things. If teams know one another and know the behavior of their colleagues, they are better able to react to issues and fix them. To build such teams it takes the experience of several jumps before people start working together as one team. Experienced members help build a team more quickly and take on more difficult challenges, but you can't expect that people who have never worked together will do a great job with their very first attempt. For some reason, people don't have a chance to build teams in IT – they aren't motivated to do it and would rather act as a set of independent actors, or they shuffle back and forth so much that they basically don't have the time to learn how their team members work. A key indicator of a well-built team is that it produces coherent work. This means that when you turn to find your teammate's shoulder, it's exactly where you expect it to be. The second thing that is usually missed is the fact that teamwork takes time. If you are solo, you able to do the same maneuvers planned for group, but much faster since you don't need to wait on other people and coordinate with them. In IT, the number one mistake is, "Let's add more people and we'll get it done faster," which completely misses both points above.

RELAX

Stress reduces your ability to think. A diver's first jump is so stressful that it's hard to remember and repeat three simple actions. The major rule is to relax and not make any sudden or sharp moves. Don't delay your actions, but don't rush either - rushing leads to mistakes and it takes 5-10 times longer to recover from a mistake than just simply taking a deep breath. So, relax!

SUMMARY

Actually, all activities in the skydiving world have one objective, which is to make jumps safer and more predictable. The industry is constantly progressing in this area. If you combine knowledge about the environment and safety with proper planning and prioritiza-

tion, teamwork and calmness, your project's execution will be fun - just like one of the scariest sports on the planet. That's probably why I like it - people create a balance between having fun and sticking with rules that save lives.

PS: No skydiver was hurt or injured during the writing of this article.

MIKHAIL DRUZHININ

Mikhail Druzhinin has over ten years' experience in software development. He has participated in full-cycle application development, leading development groups and providing hands-on experience in a variety of programming languages and technologies, including solution design and architecture. He works as a trainer in Luxoft's Training Center, where he conducts trainings in the area of architecture and Java development. His major zones of interest is cloud-based and distributed applications, and DevOps.

LUXOFT **ON THE AIR**

Check out our Youtube channel – **www.luxoft.com/youtube**

LOGEEK MAGAZINE

career.luxoft.com

Editor: Dilya Saramkova
For any comments, suggestions or requests to contribute to the next issue, please write to LGmagazine@luxoft.com