# My Career - A Retrospective



## TCG Software

Joined as a fresh graduate out of College - 2003.

### ABI Prism 7000

We used to maintain the PCR machines by Applied biosystems. That was my first job and I was a fresher - and I was taught about almost everything - from Visual Source Safe - to MFC to Visual Studio - to all the nitty gritties of windows and naturally - how to get a hold in the business of software.

This project was running for more than 5 years - and no team member chose to dive deep and gather more information about exactly what the  PCR machines are being used for.

Given we were maintaining biotech products - I became much interested in the nitty gritties of the bio-engineering starting with genetics, DNA and microbiology and what not. The seniors in the team found my efforts to comprehend the actual bio-engineering amusing.

6 months into it, and the Applied Biosystems decided to : "check our knowledge" on the basics of the "bio-engineering" and it was at tha point the team lead decided to let me present my findings about how the bio-tech is actually working out.

The presentation was much appreciated - and it saved the day for the team. Client was not expeting any of us know anything much about their actual work - it went a long way to strengthen their trust in our ability to deliver the business value.

See :

1. https://en.wikipedia.org/wiki/Polymerase_chain_reaction
2. https://conquerscientific.com/cq-product/abi-7000-real-time-pcr-system/

Lesson Learned : Figure out the path of the money - what makes money, what is the competitive advantage one is having.

## Microsoft

Hired as MSFT India had a strength of 150 in 2003 and was expanding.  Was interviewed in 2003 dec, and joined 2004 March. Did have a 3 months notice period, which TCG decided to continue with.

### WPF Designer

Team was building prototypic WPF for collaboration with SAP.  The editor of choice was Notepad. It was my responsibility to test the system - and it was impossible to do w/o a designer. Hence went on creating a designer - which was frowned upon by manager. Later after talking to multiple colleages - got hold of 10 mins time in a sync up meeting and demonstrated the designer tool.

It was loved by everyone - and the team immediately threw a party for the same in 11 PM night. I was awarded 2 times and also got a patent for the same work.

**Patent** : https://patents.justia.com/patent/20070130205

**Lesson learned** - prototype and demonstration to everyone in question helps.

## Auto Debugger

In Windows 7 , our teams problem was finding out issues which is making an App crash / not working in windows 7  build - but the same working in Vista. We used to follow step by step process to debug the same. It was teams idea to write a debugger while running two virtual machines to check where the flow of execution is differing.

This was deemed too much effort, and management was not really pro for it because of the RnD involved.

We continued doing it as a side project in weekends and the late nights - and finally demonstrated it in Microsoft Tech Excellence forum. We were judged 3rd. The system later used heavily for debugging "compatibility" issue in Windows Systems.

**Lesson learned** - You are never best. Even your best efforts are .. possibly will be overshadowed by better folks, that does not mean your efforts mean nothing it just means there always will be smarter folks than any of us.

## Windows Extra Downloads

During the Vista Service pack timeframe it was necessary to find out all possible patches microsoft ever released to do testing on the different variations. Unfortunately the whole team had no idea about what needs to be done. Eventually the problem was thrown in the open where I suggested a scraping from windows update global portal. No one in principle agreed - albeit no one had other choice. A Ruby Script using Watir was written to automate IE - and we could find more than 1,500 different patches  - for the same.

Recieved reccomendation from Distinguished Engineer for the same - for quick and dirty way to save the day.

**Lesson learned** - Sometimes, many times, innovation is just equivalent to taking inititative and completing it. Smarts sometimes has very less to do with it.

## Windows SE Planning

At 2010 plan was to think how Windows sustained engineering will work after 5 years.  Explained the near supremacy of macbooks in the mid and the small size startups over windows. Ostriched for stating clearly that Apple also provides free support and their integration seems to better. Unfortunately it was not Satya's Micrsoft and I was told - the statements I am making are "Anti Company".

Apparently talking about Linux, Unix Mac system - and telling lots and lots of people are using them was Anti Company.

The Senior director explained to me to write it down stating that  - these "matter of opinions" are the things that only time can decide upon.

Used the method in abandance. In next year India Dev Centre stopped Windows SE team. In next 2 years it saw Windows diminished market share to the point - Microsoft declaring the final version of the windows.

Satya's Microsoft adapted a better culture - I am pretty sure this does not happen anymore. The end of Internet Explorer, embracing the Chakra engine are all signs of ego-less maturity.

**Lesson learned** - In many occurences what is "right" matters not, what can not cured, must be endured till time comes. Hence, know the audience.

###

# D.E.Shaw & Co - Arcessium

Joined out of MSFT as "Lead" - Performance Engineering. Soon figured out the firm has lots of use for me. Setup the automation infra - mostly worked with declarative computation for Business needs - optimize resources, do more with less.

## API Unit

Head of India ( Senior Director ) had a vision of making granular data driven unit tests - to increase branch coverage for all interfaces across business. People argued more resources were needed - and teh situation was an impasse. API Unit was proposed where app input data were passed via spreadsheets / text files and using reflection the methods were called.

Input data were generated automatically from atomic rudimentary data via combinatorics.

Initally viewed as "just a novel technique" the uses of API Unit spread across all teams - and the resultant 80% above branch coverages were  achieved using fractions of the developer costs.

Arguments against such a technique were about the "disjointness" of the data from the actual test code - which had no "real" test code to begin with. Eventually peopel came to realize that people are already doing it in spring and hibernate.

**Lesson learned** - Aligned project with business priorities almost surely will win, if the business is truthful about the priority.

## CHAITIN

Objective was to automate UI tests faster - and make them robust for the internal PnL applications. Three schools of thoughts were there. One spreadsheet based non Turing complete approach existed, majority was favouring a webdriver based approach, including the business heads. Explained to them that a Turing Complete DSL based approach is a better alternative - albeit w/o prototype it was hard to to sell.

Had to travel to NYC to demonstrate the capabilities and convince the stakeholders including all SVPs for the same. They agreeed to a tight deadline for demonstration - which was made possible by the dedication of the team - and was immensely successful at that time for next 5 years.

Being said that - my senior director remarked that he did the DSL business before - the trouble with them is that w/o the creators/incubators they falls off eventually.

This actually happened. While all the old tests were all written and run using CHAITIN, the new tests are not being written there anymore.

**Lesson learned** - Aligned project with business priorities even if they win initially - long term sustainablity of such priorities is necessary.

### GECO

Tax computation was a problem and verification was a larger problem - a problem that used to take more than 2 weeks to compute. Tax computation can be thought as a Verification of matrix manipulation with business rules, upon which the proposal of a GECO - Generic Expression Comparator.  Initially frowned upon because "A Good Theory is Hard to Grasp" people started seeing results with a working prototype that reduced the time of tax computation & verification from 2 weeks to 1 single day - every night the computations used to run.

After 3 months of rigorous cross verification GECO became the accepted way to verify tax computation. It used to run on close to 20 billion dollars in profit.

A reference presentation about power of DSL in formal verification :

https://www.slideshare.net/nogamondal/formal-methods-in-qa-automation-using-dsl

**Lesson learned** - Results speak a lot than any presentation about potential value adds.

## LinkedIn

Joined as the first ever Staff Engineer hired from India in India.

### Layout Testing

Galen is one of the "layout" testing framework. Prototype with Galen was being worked upon in LinkedIn when I joined and all the Staff declared the prototype "not worthy" because of immense amount of  configuration required. Many declared the problem and the project "dead end". Given the problem a project "Layman" was created - which took basic ideas about declarative composition of UI from CSS itself - and with custom expression engines with JSR-223.

A demonstration of the capabilities along with detection of faults made the tool a favourite for the developers and was planned to be included in the pipeline for the Mobile build along with daily tests running for break in mobile web.

This required a lot of discussions across various orgs and teams - because a failure of the tests would mean from P1 to P2 issues in production. After a lot of discussions and demonstrations it was allowed to seat in, producing almost daily issues in the mobile web.

**Lesson learned** - Build consensus as you go, in case of varying business priority

### Declarative Experimentation

LinkedIn as the social media for "career" relies on a lot of experiments to funnel back it's not so active user bases to it's site/app. One such "funnelling back" mechanism is notification systems - which sends push notifications, emails and sms from LinkedIn. The whole infrastructure was hosted in the Ajkaban cluster, the worlds largest hadoop system with more than 10,000 physical machines.

The jobs were long running Ajkaban workflows with pig scripts. The trouble was "small modification" would require code change, unit test, run , test, deployment. An experiment would be at least 2 weeks away. And PMs loved to experiment - and they would be delayed every time.

The data workflows were not ready for rapid experimentation. This is a stament no one wanted to hear about. At the same time PMs were not really happy about it.

The proposition we came up with was let the gradle workflows will remain as it is, but we can introduce script injection in them using UDF - user defined functions. With that, and with hosted scripts and paramterized script locations - a workflow would become "programmable". While the PMs agreed to it principally, they were against "Over Engineering".

The same was done - via JSON based DSL with JavaScript injecting. It worked - and allowed PMs flexibility to quickly experiment on notifications. They could experiment within a day now - from starting to end and see the results in the next day.

Overall, people were happy.

**Lesson learned** -  Sometimes Quick & Dirty wins.

### Generic Aggregation for Services

LinkedIn had an elaborate way of doing aggregation - with ParSeq and DECO frameworks. Any simple aggregation between multiple webservices  required 3 weeks of work. Proposed a generic aggregation mechanism based on SQL and Script Injection.

The prototype was done in a night's time and demonstrated : https://github.com/nmondal/super

Other staff engineers were mighty impressed - one mentioned I should look into https://en.wikipedia.org/wiki/Yahoo!_Query_Language

My manager also was impressed and declared that is why I am a Staff Engineer, but being said that he decided not to pursue it - because it goes to a different organisation - infra. He suggested to move me into Infra.

I bailed out, because in my mind those distinctions are fuzzy. But that is now how the organization thought, so I let it go.

We did not give up on the problem, however. 6 years later in Compass we built a full fledged system that competes against GraphQL specification.

**Lesson learned** -  Quick & Dirty works till an extent - the extant problem might be too large for a single persona. Teams are there for a reason.

## BayesTree

Bayestree is a startup that is in the recommendation engine space.

## Initial Prototype

The first problem to create the prototype product was to take past historical tickets for any organization and then given a new ticket predict the solutions possibles from the past solutions. Proposal from the Engineering ( me ) was to think of this as a "search" problem - which was instantly frowend upon by the CTO as well as the AI Team. The prototype developed did not work - against a tight deadline for the founding team to take the system for deomnstration in all across the U.S.A.

At this point from Engineering side the only options business was looking for something that would save the Company even before it officially gets rolling. A rudimentary jaccard distance based solution was developed - and then demonstrated - a python flask back end API was created to expose the engine and a rudimentary UI was also developed to live showcase the demonstration. The whole system was put inside a virtual hard drive of a virtual box machine.

That virtualbox machine became the de facto "demonstration" of the capability for nearly 1 year - while the ML team eventually caught up. In the end the idea remained founded in finding different measure of similarity in strings - we did eventually discover one that works better, at that time ( 2016 ) even then Google's similarity measures.

Here is the patent : https://patents.justia.com/patent/20210342536

**Lesson learned** - Trust experts, but verify. Trust outcome over pedigree. Accept expert opinion when it produces outcome.

## DSL for Web Services

Creating haphazard services and long running process is one of the key ingradients of system integrations. While working as a contractor for Reliance we discovered that it is necessary to decouple the core AI Engine from rest of the offering. The same requirement popped up almost everywhere, we needed an infra that lets people develop and deploy webservices at a blazing fast speed for rapid protoyping.

Question was - what works? JSR-223 came in rescue. We decided to have a web service based on spark-java. Yaml configurations will define routes, auth and path forwarding and proxying and reverse proxiying.

Routes will point to JSR-223 scripts - compiled and run as java compiled modules - one can write in any language having a JSR-223 compatible engine - from JavaScript to JRuby to Jython. Our preferred language of choice was ZoomBA - which I built as a prototypic language for rapid business development after comparing against MVEL and Groovy.

Engine loads those files as functions and executes - ( compiles ones and used that compiled module ). Development and system integration became fast, from weeks it became hours. Customers will ask us to customize things - expected us to take days, if not weeks, would be done in mere hours. JOLT web server became an essential infrastructural ingredient in BT's every offering.

Scripting For Java Platform : https://en.wikipedia.org/wiki/Scripting_for_the_Java_Platform

Latest Modification of spark-java for JDK 17 being done here (based on Jetty 11 ) : https://github.com/nmondal/spark-11

ZoomBA manual : https://nmondal.github.io/assets/pdfs/zoomba.pdf

## Search Backed Classification

When working as a potential contractor to Reliance industries - they proposed the problem of 0.1 million email classification a day to forward it to the appropriate group. Our product was a reccomendation system backed by our custom distance implementation for text search.

The core team put their foot down and decided not to pursue the oppertunity. It was told that we should not do it, moreover it was also told that we are on our own, apprently if we choose to get it done.

Given we had very less time, theorized and came up with a prototype based on stochastic sampling - by using the search engine we already had.

Interestingly, not only the algorithm ran, it ran falwlessly to the point company filed a patent around it:

https://patents.justia.com/patent/20210365810

More information : https://arxiv.org/abs/2011.13832

This "hack" not only saved the "face" of the company, it ensured we recieve sufficient funds to continue the company itself.

**Lesson learned** - Necessity is, truly the mother of invention. Engineering excellence is meaningless unless useful to actual users need.

# Amazon

## Vendor Fraud Detection

Amazon is worlds most "Customer Obsessed" Company by it's own admission. It goes to great lengths to act humane towards it's customers. One such thing is the problem of overage.

Consider Amazon asking for 10 iPhones from a vendor. While expected no of iphone will be 10, in reality - the vendor can provide more than 10, even 1000s of iPhone and Amazon will pay for all the 1000 iPhones in full, irrespective that how much it ordered. This phenomenon is termed "Overage".

While sounds pretty innocuous at first, one must consider the whole loop of buy and then sell. What if most of the 1000 iPhones Amazon can not sell and only could sell with a massive discounted price? Clearly there must be a fine line between customer obsession and the willingness to take a "de facto" loss of 1 billion dollar year on year ( then estimated ) due to fraudulant practices.

This was the holy grail for the Amazon VPS - for years people were trying to even crack the problem - for years people were just talking about it. We took the problem heads on - and were told from Senior Director level all the way to our Managers - not to. Some problems are better left alone. The whole problem of interacting with cross continent teams and then convincing them the right choice of action would be a hindrance. Convincing execs, more so. Thus, no one dared to do it.

Amazon is a spreadsheet deiven firm, on two colors, green and red. The objective was to find out how much red would be there once we finish it.

Evidently the problem was not easy and we did work outside our time ( Sat / Sun ) to even gather data from the Amazon Redshift and then moving them into a staging area ( EMR ) to extract out the reasonable signals. We model them as repeated biased coin tossing.

We wrote custom declarative workflow ( called DHARA ) - we wrote statistical models and analyzed them with Python and got in talk with the statistics team in Amazon to check things out - they agreed principally and gave go ahead for the prototype with our modelling.

In the end we could color the vendors in categories Red, Orange, Yellow, Green and found the "Notional Loss" Amazon was suffering per year. It came upto 750 Million dollars a year. Finally the execs had some real estimate of what was the amount of the "overage leak".

The spreadsheet was presented in front of  the execs and was culminated in heated debate about the next steps.

**Lesson learned** -  Business value addition at scale of a billion dollar is a collaborative enterprise cutting across multiple teams and value system.

# Compass

## Improving Android App

India division started with the initial mision of removing disparity between iOS and the Android app. While functional requirements never raised any conflict - the non functional requirements were ill thought of. Key areas of improvements were:

1. Crashes
    1. Due to memory leaks
    2. Due to "wrong usage" of SDK
2. Slow Performance
    1. UI
    2. Data transfer ( aggregation problem )
3. Functional Issues from commits

When we went on improving the performance for the app - the majority were not really happy about it becuase of the potential regression it may cause. In the end of 14 months we achieved the following results:

1. 99.999% available app - to do so we
    1. replaced Android Instrumentation with our own Custom one
    2. Put a library comprising of higher order functions to do a lot of wrap around standard SDK
    3. Put up augmented memor cleaning mechanism by hooking into lifecycles of Android and reflection
    4. Created a better BFF endpoint ( see generic API Aggregation )
2. The app was faster from load time of 5 sec at 50 percentile to 750 ms at 50 percentile
    1. Bunch of libraries were written to measure it even in prod
    2. Custom higher order functions were written to delegate work in the backgrounds if need be
3. Started figuring out how each commits impact the code base ( see RAC below )

The results achieved were unquestionable way to put an end to the raging debates about efficacy of the engineering system for a better non functional behaviour of the application.

**Lesson learned** -  A stitch in time saves nine. Engineering effort, if not spent in the right time, will come back and bite, and more often than not it is not about "capability" it is about willingness - and be fearless about the consequences.

## Risk Assessment of Commits

Agile literally means rapid - agility is the way to achieve things rapidly, and while CI/CD let us create process around it, it does not mandate quality or risk of a single commit over other related areas. With 1000s of tests to run the objective for RAC was to find out risk and mitigate risk for each commit - by smartly analyzing the commit and running the least no of test cases that covers the commit.

This is not really an easy thing to think about from a management point of view.

See more :

1. https://medium.com/compass-true-north/decision-dilemma-risk-taking-955b16135ebe
2. https://www.microsoft.com/en-us/research/publication/crane-failure-prediction-change-analysis-and-test-prioritization-in-practice-experiences-from-windows-2/

Initially it was a really hard story to explain to the execs - most confused this with CI/CD itself but later came to realize the true potential after demos. Engineering managers were the first one to grasp the impact quickly, the PM org 2nd. Eventually it was demonstrated in a org wide meeting where the CTO declared RAC as : "Fundamental Engineering" .

RAC formed the backbone for the agile engineering in mobile and expanded into backend.

**Lesson learned** -  A great engineering success can come from anywhere, be expectant of it. If we wait for something to happen, and try hard, it would eventually happen.

## Generic API Aggregation

The bassis of the project was the axiom:

"Most" of the "back-end" development is trivial to the point of :

1. Exposing CRUD operations over a data store ( db , nosql db etc ) - Data Sources - Atoms
2. Aggregating multiple atoms to create an aggregation endpoint

In pure functional form one can think the first one as parameter passing to the actual functions, and 2nd one is that of function composition of sorts.

These are very bold claims - and they divisive to the core.  It goes to the debate of 1960s where DB technologies were making inroads. The storing and loading of data itself was taken as a big problem - unless data stores settled it and SQL became the norm.

Based on this theory - and it's need to create a back-end-for-front-end layer - we went on to create a GraphQL like infrastructure that via yaml based configuration exposes a real time data pipeline through multiple layers of API calls, with support for automatic parallelization and timeouts and retries on the workflow as well as node level.

Naturally while this is a good theory - reality us sometimes it is better to have actual code - and debate ensues whether to adpot to this SIRIUS infrastructure in a large scale. The mobile team argued that the they are going to use it anyways, it helped them build stuff faster, cheaper - and a cleaner way by maintianing configurations - while SIRIUS engine takes care of the rest.

The backend team argued - only for some trivial APIs such things are possible - it is hardly possible to maintian business logic in configuraiton.

A counter argument came - business logic in configuration based system is the heart of any workflow or rule engine.

Many confused it with non real time workflow systems in Amazon like HERD  ( internal ),  SWF,  Apache Airflow, and Ubers Cadence. The fact is none of these are even true compititors - the only true competition are GraphQL engines - like Hasura.

Primary objective from business was to ensure that we can run 100s of end points by a single team of 3, instead of having 5 endpoints being maintained by  a team of 10.  This naturally culminates job security issues - and debate raced to reach to the top of the organization.

An Estimate by the team showed - we could (at least ) run the show with 60% of the team size that we have ( a minimum 40% reduction).  While everyone principally agreed - they eventually declared ONLY mobile will use it.

Future is unknown on the project - with organization deciding to let go 60% of it's workforce the organization itself is in a tripping point.

Ref : https://www.vice.com/en/article/n7z7zm/real-estate-giant-compass-is-facing-an-existential-cash-burn-problem

SIRIUS is not only a cool piece of tech - it can, and should act as saviour for the organisation at large, if the right choice is being done. Only time can tell.

**Lesson learned** -  Not every battle is worth fighting - eventually we need to let people choose what is good for the organization. This is an alignment problem and human problem, tech is not a part of it. Resistance to new tech is expected and even essential, but that way calculators and not computers will dominate us today. Let the tech add sufficient ROI and let the alignment grow organically.

# Final Words

Throughout my 20 years of journey in the so called field of "software" it has been pretty apparent for me is that "right fact" almost never wins any debate whatsoever.

For any organization there must be a vision. There should be missions to carry that vision. And then the team arranges around it. Once such a mision becomes a reality - getting everyone into same page is just a mere formality, else you are not part of the mision anyways.

During my operations as junior engineers the mission was very simple - follow the product thinking to show value add.

As I grew more and more, it became apparent that the mision I should derive from the vision statement for the top brass in an organization.

An organization is federated. If everyone believes in the same vision - then creating a common mission is easy.

During my 20 years everytime there was a clearn closure to any of the conflicts, it was very clear that it came from the mision and vision.

The rest - it was very clear the debating party did not have the mission and vision clarified. Business does or does not need it is a shallow, narrow way of looking at the world - because it can change within a span of months as seen in Compass, and as seen in Microsoft ( over 2 years ).

An Engineering leader is as good as the team. And the culture of the Engineering team is as good as the leaders leading them.

Culture of how to deal with conflict not only demonstrates the teams but also demonstrates the value system the organization build in.

References :


1. https://engineering.linkedin.com/blog/2016/03/what-is-craftsmanship-and-why-is-it-important-
2. https://relevant.software/blog/build-winning-engineering-culture/
3. https://nmondal.github.io/assets/pdfs/eng.pdf