# Avraham “Abe” Bernstein | Professional CV

*Version: 2.0.0*

*Last update: 2017-07-27T07:24:14Z*

*Copyright © 2017 Avraham Bernstein, Jerusalem ISRAEL. All rights reserved.*

*License: Except where otherwise noted, this work is licensed under the Creative Commons License* [*CC BY-ND 4.0*](https://creativecommons.org/licenses/by-nd/4.0/)



Secure photo[[1]](#footnote-2) of the author, Avraham Bernstein c. 2010.

### 0.1 Contact Info & Links

**email/skype/pgp:** [Avraham.Bernstein@gmail.com](mailto:Avraham.Bernstein@gmail.com)

**pgp-id:** [E6647D2F](https://pgp.mit.edu/pks/lookup?op=vindex&search=0x86EFCDAEE6647D2F)

**geo-location:** Jerusalem ISRAEL, **tz:** UTC +2:00/+3:00 [winter/summer]

**tel-IL-mobile/whatsapp:** +972.54.641-0955 **[preferred]**

**tel-US-mobile:** +1.845.402-0023

**www-home:** <http://purl.org/Avraham.Bernstein>

**linkedin:** <https://www.linkedin.com/in/AvrahamBernstein>

**cv-full-html:** <http://purl.org/Avraham.Bernstein/cv.html> **[preferred]**

**cv-abbrev-html:** <http://purl.org/Avraham.Bernstein/cv-abbrev.html>

**cv-full-docx:** <http://purl.org/Avraham.Bernstein/cv.docx>

**cv-abbrev-docx:** <http://purl.org/Avraham.Bernstein/cv-abbrev.docx>

### 0.2 Table of Contents

[Avraham “Abe” Bernstein | Professional CV 1](#__RefHeading___Toc1927_1823469188)

[0.1 Contact Info & Links 1](#__RefHeading___Toc1929_1823469188)

[0.2 Table of Contents 2](#__RefHeading___Toc1931_1823469188)

[1.0 Summary 3](#__RefHeading___Toc1933_1823469188)

[2.0 Work Experience 4](#__RefHeading___Toc1935_1823469188)

[2017 - present: Consultant: Cybersecurity 4](#__RefHeading___Toc1937_1823469188)

[2011-17: Security Manager & Architect: OTT Internet Pay TV System: Secure Player 4](#__RefHeading___Toc1939_1823469188)

[2016-16: Consultant: Cybersecurity Protection of a Small Business With Extremely High Security Concerns 8](#__RefHeading___Toc1941_1823469188)

[2010-12: Part-time CTO: US Agricultural Trading Company with Nigeria 8](#__RefHeading___Toc1943_1823469188)

[2010-11: VP R&D: Urban Traffic Vehicle Route Guidance 9](#__RefHeading___Toc1945_1823469188)

[2010-10: Consultant: Transparent Technical Negotiator of Restricted Hi-Tech Trade with China 9](#__RefHeading___Toc1947_1823469188)

[2009-09: Consultant: Bioinformatics PCR Algorithms 10](#__RefHeading___Toc1949_1823469188)

[Methodology 10](#__RefHeading___Toc1951_1823469188)

[2004-09: Security Research & Business Development 11](#__RefHeading___Toc1953_1823469188)

[2002-03: CTO: Blind Accessibility H/W & S/W to Enable Viewing of Digital Images & Maps 12](#__RefHeading___Toc1955_1823469188)

[1999-02: Manager Cable Modem & Router Network Utilities S/W Development Group 13](#__RefHeading___Toc1957_1823469188)

[2001-01: Consultant: Network Management System (NMS) for FSO Devices 13](#__RefHeading___Toc1959_1823469188)

[2000-01: Contract Programmer: Win32 Asynchronous Network Driver DLL for a VisualBasic Project 13](#__RefHeading___Toc1961_1823469188)

[1998-99: Contract Programmer: GCC Compiler Port for a 128-Core Stack Machine 14](#__RefHeading___Toc1963_1823469188)

[1997-98: Contractor: S/W Architect & Implementation of Conoscopic Interferometer Workstation 14](#__RefHeading___Toc1965_1823469188)

[1995-96: CEO & CTO: S/W Architect & Implementation of US DOD Mil-Spec Automated Testing System: Night Hawk Fire Control System 14](#__RefHeading___Toc1967_1823469188)

[1991-94: CEO & CTO: S/W Architect & Implementation of S/W Toolchain For DSPG PINE CPU 15](#__RefHeading___Toc1969_1823469188)

[1989-91: S/W Architect & Implementation: Shop Floor Production Control (SFPC) System: BARI II 15](#__RefHeading___Toc1971_1823469188)

[1988-88: S/W Architect & Implementation: Quadriplegic PC Accessibility 16](#__RefHeading___Toc1973_1823469188)

[1987-88: Senior Programmer & VAX/VMS Sysadmin: 3D Printer: Solider 17](#__RefHeading___Toc1975_1823469188)

[1986-86: Programmer: Automated Leather Sewing Workstation 17](#__RefHeading___Toc1977_1823469188)

[1984-85: Programmer & VAX/VMS Sysadmin: Hebrew/English Word Processor: Glyph 17](#__RefHeading___Toc1979_1823469188)

[1983-84: Programmer: Real-Time 17](#__RefHeading___Toc1981_1823469188)

[1981-83: Programmer & IBM CP/CMS Sysadmin 17](#__RefHeading___Toc1983_1823469188)

[1979-80: Programmer/Economist 17](#__RefHeading___Toc1985_1823469188)

[1977-78: Intervenor/Economist 17](#__RefHeading___Toc1987_1823469188)

[4.0 Education 18](#__RefHeading___Toc1989_1823469188)

[4.1 Formal Education 18](#__RefHeading___Toc1991_1823469188)

[1979: York University, Canada: MA Economics & Applied Mathematics 18](#__RefHeading___Toc1993_1823469188)

[1977: University of Toronto - Rotman School of Management (MBA Program): No Degree 18](#__RefHeading___Toc1995_1823469188)

[1976: University of Toronto: BA Economics & Applied Mathematics 18](#__RefHeading___Toc1997_1823469188)

[4.2 Continuing Education 18](#__RefHeading___Toc1999_1823469188)

[5.0 Spoken Languages 19](#__RefHeading___Toc2001_1823469188)

[6.0 Computer Languages, SDKs, and Operating Systems 19](#__RefHeading___Toc2003_1823469188)

[6.1 Programming Languages Musings 19](#__RefHeading___Toc2005_1823469188)

[7.0 Patents Under Development 21](#__RefHeading___Toc2007_1823469188)

[8.0 Personal 21](#__RefHeading___Toc2009_1823469188)

## 1.0 Summary

I am a computer scientist and technologist. I work at the CTO level. I am innovative and creative. I have a strong background in economics. I communicate well both orally and in writing.

I am a master S/W engineer and architect of industrial strength systems. I am “hands-on”. Typically due to the complex and novel nature of the problems that I solve, first I need to build [proofs of concept (POC)](https://en.wikipedia.org/wiki/Proof_of_concept) before completing architectural and design specifications. After 35 years I still feel it is the most creative profession that I could have chosen.

For the past 13 years, I worked primarily in the field of cybersecurity with a specialty in [S/W obfuscation](https://en.wikipedia.org/wiki/Obfuscation_(software)), i.e. anti-reverse engineering techniques. I have also worked in a broad range of other application domains where I have quickly come up to speed, and functioned at an expert level. These domains include [bioinformatics](#consultant-bioinformatics-pcr-algorithms), [cryptography](#patents-under-development), [transportation](#vp-rd-urban-traffic-vehicle-route-guidance), [blind vision](#cto-blind-accessibility-hw-sw-to-enable-viewing-of-digital-images-maps), [accessibility](#sw-architect-implementation-quadriplegic-pc-accessibility), [VLSI CPU design](#ceo-cto-sw-architect-implementation-of-sw-toolchain-for-dspg-pine-cpu), and [factory automation](#sw-architect-implementation-shop-floor-production-control-sfpc-system-bari-ii). For example just a few years ago, after only a few months in the new (for me) field of bioinformatics, I developed a new [PCR Ct](http://www.thermofisher.com/il/en/home/life-science/pcr/real-time-pcr/qpcr-education/pcr-understanding-ct-application-note.html) calculation algorithm which overcomes the problem of [PCR inhibition](https://en.wikipedia.org/wiki/Polymerase_chain_reaction_inhibitors). I am in the process of patenting this algorithm. I was fortunate that my mentor was the one who developed the original algorithm - which is the one most cited in this field in Google Scholar.

Invariably I am able to apply knowledge and [meta-knowledge](https://en.wikipedia.org/wiki/Metaknowledge) that I have acquired from other fields, and apply it to the new one. A common technique that I use is to create a [domain specific language (DSL)](https://en.wikipedia.org/wiki/Domain-specific_language) that allows me to write a formal specification of a problem. With a little ingenuity the DSL can be used to create a working command line application, and it usually forms the basis of an automated testing framework. I successfully used a DSL to describe a [shop floor production control system](#sw-architect-implementation-shop-floor-production-control-sfpc-system-bari-ii), along with a compiler that executed/implemented the shop floor. And I used a DSL to describe a [CPU architecture](#ceo-cto-sw-architect-implementation-of-sw-toolchain-for-dspg-pine-cpu), along with a compiler that automatically generated a simulator/debugger, assembler, and C compiler. A common programming technique that I use is [metaprogramming](https://en.wikipedia.org/wiki/Metaprogramming) with templates that allows me to automatically generate code by creating intelligent macros regardless of the underlying target programming language.

I am an [expert generalist](http://99u.com/articles/7269/picasso-kepler-and-the-benefits-of-being-an-expert-generalist), and an [autodidact polymath](http://autodidactpolymath.com/autodidact-polymath-definition/), i.e. a self-learner in new fields who achieves expertise quickly. I know how to find and utilize mentors who are experts in the new field. I am also a good mentor myself. I slavishly attend to my [continuing education](#continuing-education).

## 2.0 Work Experience

### 2017 - present: Consultant: Cybersecurity

*@Self-Employed, Jerusalem:*

1. Developing my own [obfuscating](https://en.wikipedia.org/wiki/Obfuscation_(software)) [compiler](https://en.wikipedia.org/wiki/Compiler) for C/C++ and for [Web Assembly (WASM)](http://webassembly.org/). Still in stealth mode.
2. Collaborating with the compiler company Semantic Designs [Legacy Software Migration](http://www.semdesigns.com/Products/Services/LegacyMigration.html) tools for projects in Israel.
3. I am a mentor for the Jerusalem [Mass Challenge](http://israel.masschallenge.org/) start-up hub.

### 2011-17: Security Manager & Architect: OTT Internet Pay TV System: Secure Player

*@Viaccess-Orca, Ra’anana - a subsidiary of Orange FR, and @Discretix/SansaSecurity, Netanya - now merged into ARM:*

1. I was head of security for their [Over-The-Top (OTT)](https://en.wikipedia.org/wiki/Over-the-top_content) Internet pay TV product for 6 years. I joined the project shortly after its inception. When I left there were 40M subscribers, and there were no successful hacks in the field.
   * The company fired me because they were in financial shock due to the sharp drop in their primary source of revenue from [smart card](https://en.wikipedia.org/wiki/Smart_card) sales for their [Conditional Access (CA)](https://en.wikipedia.org/wiki/Conditional_access) product which are being annihilated by the pure OTT operators such as Netflix and Amazon Prime.
2. I was responsible for end-to-end security policy and architecture, and inclusion and implementation of security related programming utilities.
   * We used the [Jira](https://www.atlassian.com/software/jira/agile) system for agile project management.
3. The supported client devices were Android and IOS phones and tablets. The subscriber ID was based upon the built-in Android device ID, and the built-in IOS device-application ID, respectively.
4. We supplied a reference application to our customers who were our [CA](https://en.wikipedia.org/wiki/Conditional_access) broadcaster/operator customers who needed to provide an [OTT](https://en.wikipedia.org/wiki/Over-the-top_content) option to their subscribers. They customized our reference application to their needs. We did not have a direct relationship with the subscribers. Our operators typically took a very long time to do their integration and QA before they would release/push their application to their subscribers. The fundamental technical flaw with this business model was that when we found a serious bug or a security problem in the core modules, the operators would typically delay for 6-12 months before pushing the update to their subscribers. Eventually I architected a solution that provided good enough security (see [below](#new-config)), while still satisfying the QA needs of our customers.
5. The original [DRM](https://en.wikipedia.org/wiki/Digital_rights_management) was [PlayReady](https://en.wikipedia.org/wiki/PlayReady). Later we supported Google [Widevine](http://www.widevine.com/), and the [Viaccess-Orca](http://www.viaccess-orca.com/content-protection.html) proprietary DRM.
6. For security reasons, all client S/W was implemented in native machine code, i.e. in C/C++, except for the UI which was implemented in Java and Objective-C, for Android and IOS respectively.
7. Some S/W security tools we purchased, namely the [InterTrust/WhiteCryption](https://www.intertrust.com/products/application-security/) [obfuscating](https://en.wikipedia.org/wiki/Obfuscation_(software)) compiler and white-box encryption tools.
   * *Prima facie* these tools seemed very expensive. The combined package costs $100+K, plus the standard 15% annual maintenance fee. However having been involved in the development of similar tools at Cisco-NDS (see below), they are very complex and can easily consume 5-10 man-years of development. Therefore buying them off-the-shelf made economic sense. When we first purchased the obfuscating compiler, the product was new, and had bugs. On account of my background having built a similar tool myself, I worked closely with the vendor’s architect to help debug their product, and suggested and specified feature improvements.
   * The primary technique used in state-of-the-art *binary* obfuscation is to randomly change program flow by using opaque conditions and jumps that fool the compiler that seem to grossly change the flow, even though the final result of program execution is the same. Note that it is the job of our application build process to compile and link in a way that minimizes the use of global symbols.
   * White-box encryption relies upon wrapped keys and mathematical transformations that make it very difficult to recognize that a well known cryptographic algorithm is being executed, e.g. AES or SHA-1.
8. Many S/W security tools I architected and implemented in-house:
   1. Designed a sophisticated Android root detection mechanism.
      * Obvious signs of a rooted Android device can be determined easily from the the device’s [build properties](https://developer.android.com/studio/build/index.html). But other rooting techniques are not as clear, so I developed a [fuzzy logic](https://en.wikipedia.org/wiki/Fuzzy_logic) scoring algorithm. Based upon subscriber feedback, my rate of [false positives](https://en.wikipedia.org/wiki/False_positives_and_false_negatives) was extremely low.
      * Unfortunately near the end of my tenure, the big studios, e.g. Disney and Sony, relented on their former requirement to refuse to play on rooted devices due to significant loss of royalties from inexpensive off-the-shelf rooted devices from the Far East that represented over 25% of subscriber devices. Therefore the root detection mechanism had to become much more sophisticated. Either I had to detect the existence of a hacker configured device, or alternatively I had to recognize dynamic behavior that an actual attack was underway, e.g. memory tampering, the device was streaming, a debugger process was running, or the device was attached to the Android adb utility. However all Android developers and field engineers must use adb. Therefore I created a temporary (i.e. maximum 72 hour) device ID specific adb exception mechanism in the cloud for authorized devices. Since the vast majority of attackers use adb in order to launch, or at least to prepare, an attack, this anti-adb mechanism is one of the most important security features.
      * A major security hole has been created by the new Chrome O/S feature that uses a [Virtual Machine (VM)](https://en.wikipedia.org/wiki/Virtual_machine) to execute Android 7.0+ apps. We can continue to detect and to block other VM environments, but from a marketing perspective once the popularity of Chrome O/S reaches a significant threshold, it may no longer be possible to block it.
      * Because the root detection mechanism has been diluted, it became necessary to add security functionality to the server side. For example it is easy to detect that a subscriber who is in the 95th percentile of monthly content viewing hours, or one who views content simultaneously from multiple IP addresses, or whose IP address is volatile from geographically disparate regions. And I worked with our data scientists to collect and log internal data from the client that could predict abusive usage patterns. With 40M subscribers, this give us a huge amount of “big data” to mine.
   2. Specified for WhiteCryption an encrypted [UPX](https://upx.github.io/)-like [DSO](https://en.wikipedia.org/wiki/Library_(computing)) (dynamic shared object library = .so file) loader that required run-time authorization in order to load and decrypt - without the knowledge or support from the original supplier of the DSO.
      * Normally [UPX](https://upx.github.io/) automatically decrypts/unpacks a [DSO](https://en.wikipedia.org/wiki/Library_(computing)) upon load, without any special run-time authorization. This new mechanism prevents an attacker from doing a “reconnaissance” load.
   3. Invented a [DSO](https://en.wikipedia.org/wiki/Library_(computing)) library mechanism for Android/Linux which officially exports no symbols!
      * Implemented using the gcc/clang compiler [constructor attribute](https://stackoverflow.com/questions/2053029/how-exactly-does-attribute-constructor-work), along with multi-threading.
      * Eventually could be applied to communication with vendor [DSO](https://en.wikipedia.org/wiki/Library_(computing)) libraries without their support or knowledge, by binary editing of the the ELF file similar to the UPX mechanism (immediately above).
   4. Architected, but did not implement, a [DSO](https://en.wikipedia.org/wiki/Library_(computing)) library randomizer using [ELF](https://en.wikipedia.org/wiki/Executable_and_Linkable_Format) editing utilities. This would allow a different version of the package to be downloaded into a client device, say according to the 4-bit hash of the device ID.
   5. Designed a dynamic mechanism for testing code integrity and read-only data integrity.
   6. Designed a light weight *distributed* encryption library for situations where performance consideration precluded use of AES. In order to preclude a single point of attack, the library was small enough to allow every module that needed it to have its own private and customized copy. This library was implemented as an .h file collection of static and inline functions and variables. Every instance had its own unique static and dynamic [initialization vector (IV)](https://en.wikipedia.org/wiki/Initialization_vector), and every instance had its own unique masks for the cryptographic constants that it used. (This uniqueness was ensured by using precompiled preprocessor random number tables that were designed to be different according to the preprocessor module ID selector of including C/C++ module). Otherwise if the constants were not masked, then simply looking up the constants on the Internet would easily reveal the algorithms. The masking technique had to be extremely efficient, but it also had to outsmart the optimizing compiler’s [constant folding](https://en.wikipedia.org/wiki/Constant_folding) feature.
   7. Architected a secure and efficient data collection and anomaly logging utility, where the data was sent to the web servers for off-line analysis.
   8. Designed a mechanism for light weight randomized encryption of function arguments to foil stack probing so that no two sets of encrypted function parameters were ever the same even when the in-the-clear parameters were in fact identical. Implementation was via a light weight [homomorphic encryption](https://en.wikipedia.org/wiki/Homomorphic_encryption) algorithm that I invented.
   9. Designed a light weight randomized obfuscation of all [libc](https://en.wikipedia.org/wiki/C_standard_library) calls - in order to blind dynamic reverse engineering tools which use the libc calls as an anchor.
   10. Architected, but did not implement, a replacement of the built-in [libc](https://en.wikipedia.org/wiki/C_standard_library) calls with an obfuscated version of [musl](https://www.musl-libc.org/).
   11. Architected, but did not implement, an efficient secure logging/debug utility. We did not want to allow our foreign commercial agents to have access to the in-the-clear log messages. And without dynamic configuration/filtering of the log messages, the massive debug communication traffic would cause playback [jitter](https://en.wikipedia.org/wiki/Jitter).
   12. Architected, but did not implement, a secure static and dynamic configuration mechanism including a cryptographic key ladder mechanism. Now only the single operator specific master key had to be “baked in”, by using obfuscation, into the release of the operator specific [DSO](https://en.wikipedia.org/wiki/Library_(computing)).
       * This configuration file was a replacement for the original awkward mechanism that baked all keys into the operator specific DSO. Since the operators resisted frequent updates of their DSOs, we were not able to make agile and frequent key changes on the servers.
       * This mechanism would also allow us to make agile changes to the Android security database, e.g. to the list of packages commonly found on hacker devices, and to the list of device drivers used for O/S builds designed to run on virtual machines.
       * Generally our customers, i.e. the operators, were far less resistant to pushing relatively frequent database changes to their subscribers - as opposed to code updates. But it required that we modify our application architecture to make it data driven wherever possible.
9. In a nutshell, my primary job was to protect the content, content decryption keys, and subscriber keys, on the client devices. I made it nearly impossible for an attacker to engage in *static* [reverse engineering](https://en.wikipedia.org/wiki/Reverse_engineering), and made it extremely difficult to use the state-of-the-art *dynamic* reverse engineering tools such as the [IDA Hex-Rays](https://www.hex-rays.com/) decompiler/debugger.
   * Due to the challenging and novel nature of the features that I architected, I usually built a working proof-of-concept (POC) before handing the final specification to the programmers. Typically my implementation was language was C, as opposed to C++, because the project was comprised of both C and C++ modules, so using a C interface was the lowest common denominator.
   * Due to the large number of preprocessing and postprocessing utilities in our build process, I had to do my own [devops](https://en.wikipedia.org/wiki/DevOps) “plumbing” changes in order to do end-to-end testing of my POCs.
10. Unfortunately the bottom line with respect to security is that no matter how well protected the client S/W is, you have to expect that it will eventually be hacked. Without pushing a new randomly obfuscated version, say every 60-90 days, no protection scheme will work.

### 2016-16: Consultant: Cybersecurity Protection of a Small Business With Extremely High Security Concerns

*@Anonymous:*

1. The client works on an extremely politically sensitive topic.
2. Their WordPress web site was being regularly attacked.
3. Their researchers need to remain anonymous - for their own personal safety.
4. Their file sharing system is based upon Google Drive.
5. Their PCs and smartphones are not hardened.
6. No one used multi-factor authentication.
7. The only PC operating system with which they are familiar is Windows.
8. First I did a risk analysis, and then started them on a series of incremental steps to improve their security situation.

### 2010-12: Part-time CTO: US Agricultural Trading Company with Nigeria

*@Anonymous:*

My job was to find hi-tech agricultural products that the company needed to purchase, or that I felt would be worthwhile for them to purchase. I would go to the annual [Israel Agritech](http://agritechisrael.org/) exhibition, considered to the be the best in the world, to hunt for products and ideas. For example:

1. Portable field hospitals and inexpensive quick-to-assemble modular housing units for flood victims.
2. Inexpensive moisture proof storage bags that can store 500-1000 kg of grain in humid environments with almost no spoilage, whereas previously nearly 75% of the crops placed in field storage facilities were destroyed from moisture induced rot.
3. Mold resistant produce containers (up to 20 kg) for storage in humid refrigeration facilities in order to extend shelf life.
4. Bacteria resistant milk cartons for non-refrigerated storage.
5. Kits to convert SUVs and light trucks to become off-road bush vehicles.
6. Expert on industrial scale cassava processing which was formerly the major food staple in Nigeria before they joined OPEC, and may soon again become their major staple when the oil stops flowing because the government will no longer have the financial resources to subsidize imported wheat.
7. Cement research expert to consult for the largest cement factory in Africa.
8. Establish a private local area GSM cell phone tower with a 10 km radius range in order to support a cassava plantation/factory to be built in a remote jungle location where cassava naturally grows. The factory was well outside the geographic regions supported by the local mobile telephone companies. The efficiency of the plantation would be greatly increased provided that all the workers had mobile phones even if the communications were restricted to the 300 sq. km. region surrounding the factory.
9. Prepared a bid for refurbishing an abandoned Nigerian government 200 MW natural gas electrical generation station near the cassava plantation. Negotiated with the Israel Electric Corporation (IEC) to submit a joint bid, but eventually they backed out and joined another bidder.

### 2010-11: VP R&D: Urban Traffic Vehicle Route Guidance

*@TeleQuest, Jerusalem:*

1. I designed algorithms and a computational infrastructure similar to what [Waze](https://www.waze.com/) does today.
2. I implemented these algorithms on Tokyo traffic simulations which is the most traffic congested city in the Western world, and who happen to have a superb traffic data collection system.
3. I built [AWS](https://aws.amazon.com/) cloud infrastructure to run real-time traffic simulations.
4. The simulation S/W was written in Java. Due to the huge size of the simulations, I needed to optimize the Java code - in particular to control garbage collection.
5. The results of the Tokyo simulations were that with a threshold of just 20% guided vehicles on the roads, we could achieve up to a 50% performance decrease in travel times for guided vehicles, with a side effect of 40% decrease for non-guided vehicles. These results were the exact opposite of the competition which generated their own secondary traffic jams once the guided vehicle threshold reached 20% due to their simplistic opportunistic algorithms. I managed a group of 6 people including programmers and mathematicians.

### 2010-10: Consultant: Transparent Technical Negotiator of Restricted Hi-Tech Trade with China

*@Anonymous:*

1. The Chinese wanted to buy a restricted gyroscope for their super high speed trains from Honeywell that is normally used for ICBMs. I proposed to Honeywell that they insert the gyroscope into the center of a tamper proof lead block that weighed say 50-100 kg. For example if the soft membrane surrounding the gyroscope were penetrated with a blade or drill or laser then it would release acid or a binary explosive. Honeywell would be responsible for developing the tamper proof mechanism. They did not have to reveal to the Chinese how they built the mechanism. All they had to do was to give the Chinese a fixed price quote for the custom work. Clearly such a device could not be used on a missile where every extra gram of weight counts, but where the extra weight was of no consequence to the train. Honeywell agreed.
2. The Chinese wanted to buy the restricted Green Hills real-time operating system. The US government restricts the export of Green Hills source code, but not products that contain Green Hills’ binaries. So I asked Green Hills if I could have the board support package (BSP) developed in Israel which requires access to Green Hills source code, but ship the resulting binaries to China. Green Hills agreed.
3. The Chinese wanted to buy a drone system from a subsidiary of Israel Aircraft Industries (IAI) which was designed for commercial/industrial use. The system had cruise missile like navigation capabilities with the ability to do terrain contour matching. Since the device inherently required GPS navigational capability, I suggested to IAI that they modify their software so that it would not work within 500 km of the Israeli border, along with a back-door jamming channel - due to the fact that it is not uncommon for the Chinese to resell this type of equipment to the Iranians who regularly announce their religious obligation to wipe Israel off the face of the earth.
4. An Israeli company which develops expensive VLSI design software, e.g. $50-100K per workstation, was afraid that their Chinese agent would cheat them of their royalties. The initial deal was to sell an older version of the software for a one-time flat fee of 20 licenses, in order that the Chinese dealer would be allowed to sell an unlimited number of the older licenses. But I specified how they should re-architect the next version of their software to be converted to Software As A Service (SaaS) on cloud servers. Each licensed subscriber has a workspace on the cloud where the heavy lifting and “secret sauce” are executed, while the client workstations only act as the UI. It provides excellent security against license theft. This is the new economic and technical model that Microsoft Offices 365 uses for the majority of their subscribers which has become one of their most profitable products.

### 2009-09: Consultant: Bioinformatics PCR Algorithms

*@Syntezza, Jerusalem:*

1. Within 3 months of joining the company and quickly learning the basics of bioinformatics, I discovered a new algorithm using Artificial Intelligence (AI) and Data Science techniques for handling [PCR inhibition](https://en.wikipedia.org/wiki/Polymerase_chain_reaction_inhibitors) - which I am in the process of [patenting](#patents-under-development).
2. The client’s product was a [MRSA](https://en.wikipedia.org/wiki/Methicillin-resistant_Staphylococcus_aureus) (= a lethal staphylococcus bacteria that is antibiotic resistant and thrives in hospitals) [PCR](https://en.wikipedia.org/wiki/Polymerase_chain_reaction) (= DNA amplification/duplication technology) detection kit. Typically the kit would be used in hospitals to determine whether a newly admitted patient was a MRSA carrier. A PCR test can return results within 2 hours, whereas the “gold standard” [Petri dish](https://en.wikipedia.org/wiki/Petri_dish) test, which was developed over 100 years ago, takes at least 36 hours. Given that MRSA is infectious, and potentially fatal, it is important for hospitals to be aware ASAP which patients are carriers.
3. When I started the project, the client’s PCR kit had only a 50% detection rate due to inhibition problems. This result was grossly unacceptable for any medical test. The investors had lost confidence, and were about to pull out.
4. I improved the test’s accuracy to 95%, which was 10% better than their competitors from the pharmaceutical giants. I saved the client from liquidation.
5. I stood on the shoulders of the person who developed the original [PCR Ct](http://www.thermofisher.com/il/en/home/life-science/pcr/real-time-pcr/qpcr-education/pcr-understanding-ct-application-note.html) function analytical detection algorithm, [Dr. Tzachi Bar](https://scholar.google.co.il/citations?user=XokRoVQAAAAJ&hl=en), who was my mentor, and who got me up to speed so quickly.

#### Methodology

1. I had access to a database of about 800 MRSA patient nasal sample PCR test results, along with about 20 samples from pure laboratory MRSA colonies that contained a known quantity of the bacteria. All the patient samples had also been analyzed via the “gold standard” Petri dish test. The Petri dish results were required to calibrate the PCR algorithm/results, and would be the basis of government health services (e.g. FDA) subsequent approval of the kit. As is typical in random patient samples, less than 10% were MRSA positive.
2. First of all, I noted an outrageous data anomaly, namely that the original kit failed to detect as positive many of the samples from the laboratory colonies. After investigating further, and reading the [PCR thermocycler](https://en.wikipedia.org/wiki/Thermal_cycler) manual, I realized that the position of the negative and positive control sample test tubes had been systematically placed in the wrong test tube slot positions - according to the original incorrect experimental protocol. Note that as is typical in these kinds of tests, every batch must contain both a positive and negative control. (For this particular device, there can be a maximum of 36 test tubes in a batch). The PCR algorithm uses these controls for calibrating itself. Given that we were fortunate to the have raw results from the thermocycler, I was able to reconstruct the results by using the correct positive and negative controls.
3. Even though as a result of recalibration, the results significantly improved, still they were poor. Further investigation showed that only the samples whose graphs resembled the [logistic function](https://en.wikipedia.org/wiki/Logistic_function) (= a flat “S” shaped curve) expected for the standard mathematical analysis of [biological assays](https://en.wikipedia.org/wiki/Bioassay), had results that were consistent with the Petri dish results.
4. More than 50% of the graphs of the patient samples did not exhibit the flat “S” shape curve due to PCR inhibition resulting from the use of ineffective preliminary chemistry during sample preparation, and therefore could not be analyzed by the industry standard mathematical techniques used to estimate the Ct.
5. I used patented techniques (which I cannot yet describe) *that did not calculate the Ct* to overcome the above 2 problems.
6. I split the samples into 2 random groups of equal size. I made sure that the 2 groups contained the same number of positive, negative, and laboratory colony samples.
7. One group was used for training, while the second group was used to test the predictive capability of the model.
8. As I stated above, the predicative accuracy of the new algorithm was over 95%.

### 2004-09: Security Research & Business Development

*@Cisco-NDS, Jerusalem: NDS primary business was providing* [*Conditional Access (CA)*](https://en.wikipedia.org/wiki/Conditional_access) *pay TV security systems to satellite TV broadcasters:*

1. I worked on a wide variety of security related projects. My background task was to do code security reviews. Typically secure coding is achieved by [adhering to best programming practices](https://www.youtube.com/watch?v=eL5o4PFuxTY).
2. I was a member of the architecture team for their in-house [LLVM](https://llvm.org/) [obfuscating](https://en.wikipedia.org/wiki/Obfuscation_(software)) compiler.
3. I developed techniques using Virtual Machine (VM) technology to crack [Digital Rights Management (DRM)](https://en.wikipedia.org/wiki/Digital_rights_management) schemes, and to subvert the random number generators which are the core initialization process for all cryptographic algorithms.
4. I worked with the operational security team to track hackers, and to provide stealth techniques for our researchers.
5. I arranged for world class security researchers to give seminars, (1) a week long reverse engineering seminar, and (2) a 2 day seminar how to hack with virtual machines.
6. I wrote the technical bid for securing the 2008 Olympics Beijing Olympics TV broadcasts.
7. I researched business cases for providing security services to the computer gaming industry, and for printer ink cartridges. For example sales revenue from premier games can gross $60M in their first 2 weeks, by which time they are typically hacked. Improving security so that the hack can be delayed for another 2 weeks, can increase revenue by 50%.
8. I advised senior management on the technical and business merits of potential acquisitions.
9. I did a very interesting and important non-security related project for senior management, by [data mining](https://en.wikipedia.org/wiki/Data_mining) the company’s bug database, that showed that 25% of S/W development manpower was wasted on fixing bugs. And I showed them simple techniques that could reduce this number by 80% .
10. I designed a hybrid simulator/emulator debugger for legacy [Set-Top Boxes (STB)](https://en.wikipedia.org/wiki/Set-top_box) that originally could be debugged only with printf statements to log files. My new debugger allowed source code on the PC to be debugged using the [MS Visual Studio IDE](https://en.wikipedia.org/wiki/Microsoft_Visual_Studio) debugger while still viewing the results on the STB. Implementation was accomplished by reverse engineering the STB middleware API. 80% of the middleware ran natively on the PC, while the STB low level H/W specific portions were implemented via an agent on the STB that was accessed via API calls that were implemented as [Remote Procedure Calls (RPC)](https://en.wikipedia.org/wiki/Remote_procedure_call).
11. I wrote an automated testing system for a satellite content delivery system for huge content, e.g. delivering ultra high definition movies to cinemas, and print newspapers for remote publishing. I created a [Domain Specific Language (DSL)](https://en.wikipedia.org/wiki/Domain-specific_language) in order to execute the satellite operations. After studying the Win32 source code of the satellite ground control station, I detected a major conceptual flaw which the architect refused to believe (because testers are not supposed to understand Win32 internals!). So I wrote a progressive test that brought the satellite to its knees at only 25% of its rated capacity. Afterwards the development team used my tool to develop their own unit test scripts, and to execute a system sanity test before checking-in any changes to the source control system.

### 2002-03: CTO: Blind Accessibility H/W & S/W to Enable Viewing of Digital Images & Maps

*Virtouch, Jerusalem:*

1. My task was to change the company’s focus from providing tools for teaching blind children Braille skills to instead provide tools for blind adults to use at work.
   * The fact of the matter is that government social services agencies around the world are much more concerned about funding tools that allow the blind to integrate into the work force, and thus become economically self sufficient, versus funding educational tools for blind children.
2. Previously the company had developed a [tactile mouse](http://www.avrahambernstein.com/resources/vtplayer.png) that was far too expensive for our target audience (~$350).
3. I abandoned the mouse because (a) it was too expensive, and (b) the blind find it very difficult to use mice because mouse movement must be coordinated with the cursor on the screen which they cannot see.
4. I developed new algorithms that allowed the blind to read [GIS](https://en.wikipedia.org/wiki/Geographic_information_system) maps using SVG and XML combined with audio cues (by using pitch and volume), and by using an off-the-shelf graphics tablets and stylus (that were much less expensive than our tactile mouse).
5. The blind prefer graphic tablets because they inherently have accurate spatial knowledge of their position on the tablet because they can feel it.
6. The S/W needs to make a rectilinear mapping of the tablet into a desktop window.
7. The same techniques could be used for displaying mathematical functions and geometry.
8. I *almost* saved the company from liquidation. My research was awarded a European FP6 grant of $0.5M Euro that required *matching funds*. But the investors refused to put up the matching funds due to the company’s long history of financial failure in the children’s market.
9. The most important business lesson that I learned was that building special purpose H/W for the disabled is a recipe for bankruptcy; but if you can repurpose off-the-shelf H/W, typically designed for gamers, then you can financially succeed in the accessibility/disability market.
10. The most important personal lesson that I learned was how satisfying it was working with the blind volunteers, and supplying them with a tool that allowed them to “see”.

### 1999-02: Manager Cable Modem & Router Network Utilities S/W Development Group

*@Vyyo, Jerusalem:*

1. I managed a team of 3-6 programmers.
2. I designed a NMS and MIB for cable modems and routers, and I designed the associated embedded SNMP agents.
3. I designed a hybrid IP connection for cable modems where there was no physical cable upstream channel. Instead the upstream channel used a telephone modem (ATA), while the downstream channel used the cable modem. Head end network equipment for both interfaces was supplied by the cable operator. My solution was to dynamically modify the arp table of the edge router. For typical surfing, the effective downstream rate was as fast as a pure cable solution. The company applied for a patent.
4. I greatly improved the efficiency of the laboratory modem speed stress testing by a factor of 10-100 by using a “steepest descent” search algorithm instead of a binary search algorithm. Reduced testing time per modem from hours to minutes.
5. I designed a virtual testing lab with 64K modems and 512K PCs via multiplexing the physical connections. The test lab had only 256 physical cable modems, 4 physical PCs with 8 network connections each, 1 cable router, and 2 24-port layer-2 programmable switches. By dynamically editing the PC MAC addresses, and by dynamically editing the MAC filters on the network switches, I was able to stress test the router into believing that it faced 64K modems, and the modems into believing that each one was shared by 16 PCs.

### 2001-01: Consultant: Network Management System (NMS) for FSO Devices

*@MRV / Jolt, Jerusalem.*

1. The [free space optics (FSO)](https://en.wikipedia.org/wiki/Free-space_optical_communication) repeaters enable fiber optic cables to be extended through the air via lasers for distances of up to 5 km.
2. These devices were made of digital H/W with no need for a CPU.
3. In order to provide network management capability, I selected an inexpensive micro-controller that could interface with the FSO H/W, had an Ethernet port, and a built-in Java interpreter.
4. I designed the MIB, and an embedded program that could interface with the H/W.
5. I designed a simple HTTP server that acted as an [SNMP proxy agent](http://www.mg-soft.com/mgSnmpProxyAgent.html).

### 2000-01: Contract Programmer: Win32 Asynchronous Network Driver DLL for a VisualBasic Project

*@Inex-Zamir, Jerusalem.*

### 1998-99: Contract Programmer: GCC Compiler Port for a 128-Core Stack Machine

*@Fourfold, Jerusalem:*

1. This was a very challenging [gcc](https://gcc.gnu.org/) port because the architecture had no registers, or alternatively an infinite number of registers, while RAM access was highly unusual in order to accommodate the 128 cores.
2. I needed to add pragmas for co-processing.
3. The machine instruction set was [FORTH](https://en.wikipedia.org/wiki/Forth_(programming_language))-like, so it presented some unusual optimization challenges, because it was so unlike the standard [CISC](https://en.wikipedia.org/wiki/Complex_instruction_set_computer) and [RISC](https://en.wikipedia.org/wiki/Reduced_instruction_set_computer) instruction sets.
4. The resulting C compiler worked and produced efficient code.
5. I noted early in the project that the method for emitting target object code was well suited to an object oriented design, even though at the time the gcc compiler collection only interfaced with C code. In order to lower the “impedance” of using C++, I created a static “wrapper” for the C++ classes that was acceptable to a C project.
6. The final source code was extremely repetitive. Therefore I developed a preprocessor in [TCL](https://en.wikipedia.org/wiki/Tcl) in order to automatically generate much of the code.

### 1997-98: Contractor: S/W Architect & Implementation of Conoscopic Interferometer Workstation

*@Newport-Optimet, Jerusalem:*

1. The company product was a measurement workstation based upon their [conoscopic](http://www.optimet.com/faq.php) probe.
2. The object to be measured was placed on a platter that could be moved in the XY axes.
3. The probe was static, and measured Z - actually an array of Z points.
4. The workstation also included a video camera.
5. Generally the probe could scan at its maximum speed, although some types of materials, especially reflective, required lower speed.
6. My task was to design the S/W architecture for a Windows NT PC to implement an *automated* workstation that could be used in a factory environment.
7. I created a [domain specific language (DSL)](https://en.wikipedia.org/wiki/Domain-specific_language) implemented in [TCL](https://en.wikipedia.org/wiki/Tcl) for controlling and configuring the workstation.
8. The most interesting and challenging aspect of the project was to give “life”, i.e. semantic meaning, to the millions of raw XYZ data points that were measured.
9. I used [OpenGL](https://en.wikipedia.org/wiki/OpenGL) for rendering graphics. I created a toolkit and domain specific language for the 3D visualizations - unfortunately just *before* the initial release of [VTK](http://www.vtk.org/).
10. My reference development object was a tooth (aka “Timmy the Tooth”) which I painted as if it were a topographic map.
11. An important visualization technique for mass produced widgets on the factory floor, was to compare them against a reference widget.

### 1995-96: CEO & CTO: S/W Architect & Implementation of US DOD Mil-Spec Automated Testing System: Night Hawk Fire Control System

*@Pitkha Outsourcing, Jerusalem, for @Elbit-Elop, Rechovot:*

1. The company product was the weapon fire control system for the Night Hawk laser guided missiles.
2. My task was to create an architecture and implementation of the automated test procedures dictated by the US DOD. The test specification documentation was over 1 meter high.
3. Additionally there would be many *ad hoc* tests that would be required during the development process.
4. The test equipment included external voltage regulators, external heating and cooling equipment, vibrators, etc.
5. The workstation controlling the tests was a Windows NT PC.
6. Instead of creating a monolithic test program in C/C++, I created a BASIC-like [domain specific language (DSL)](https://en.wikipedia.org/wiki/Domain-specific_language) with special drivers for controlling the various pieces of H/W.
7. I made this critical architectural decision because the test specification manual was being regularly revised, plus I needed the flexibility to allow the system engineers to write their own scripts without delving into the underlying C code. And I wanted to avoid having the client call me every time he needed to implement a minor/trivial change that he could easily learn to do himself.
8. The major tests were presented via a UI which in fact emitted DSL script.
9. The system worked as planned.
10. Unsupervised tests ran successfully for up to 72 hours (over holidays).
11. Post mortem: The first version of [TCL](https://en.wikipedia.org/wiki/Tcl) was released just around the time of this project, i.e. *pre-Internet*. Once I learned about TCL, I immediately realized it would have been the ideal platform for creating the test environment. Instead of me having to create my own BASIC-like language with control structures and variable handling, I could have relied upon TCL, and simply added custom primitives for the various pieces of H/W.

### 1991-94: CEO & CTO: S/W Architect & Implementation of S/W Toolchain For DSPG PINE CPU

*@Pitkha outsourcing, Jerusalem, for @DSPG, Givat Shmuel:*

1. I was the S/W architect of a clock accurate DSP CPU simulator along with a complete software development toolchain, i.e. a debugger, C compiler, assembler and linker. Note that the system was developed just *before* the GNU Compiler Collection framework reached maturity, i.e. v2.95.
2. This system enabled working applications to be developed before the chip became physically available. It reduced application time-to-market by 6-12 months.
3. The technological breakthrough was my design of a [domain specific language (DSL)](https://en.wikipedia.org/wiki/Domain-specific_language) implemented in [lex/yacc](https://en.wikipedia.org/wiki/Yacc) that described the CPU architecture.
4. The associated DSL compiler automatically generated the source code for the complete toolchain that enabled it be automatically rebuilt within an hour in the face of almost daily changes to the VLSI architecture - especially the pipeline.

### 1989-91: S/W Architect & Implementation: Shop Floor Production Control (SFPC) System: BARI II

*@Digital Equipment Corporation (DEC), Herzliya, for @Iscar, Tefen:*

1. Iskar Matkash in Tefen IL is a fully automated factory that produces thousands of different cutting blades using a sintering process. The raw materials go through many stages of operations. In many cases after undergoing intermediate processing, the partially processed material can still be diverted to multiple final products - similar to stem cells. The factory contains hundreds of automated workstations, stands, stacks, guided vehicles, and conveyor belts. The product or intermediate product is placed on pallets. The pallets are moved from one stand on a workstation to a stand on another workstation, or temporarily to a storage stand or stack.
2. My task was to create a computer program that automatically operated/orchestrated the factory.
3. When my co-architect and I started this project, we had zero background in industrial engineering. We were supplied with a mentor who brought us up to speed.
4. Eventually after months of discussions we created an architecture that was a textbook object oriented taxonomy - a “factory object kingdom”. The top level object was a “production instruction”.
5. We defined the attributes and methods associated with each object.
6. We created a descriptive, i.e. *non-procedural*, [domain specific language (DSL)](https://en.wikipedia.org/wiki/Domain-specific_language) that was designed to be user-friendly for the factory engineer.
7. I wrote the language manual.
8. We used the language to configure the factory. We created a UI wrapper for the language which emitted CLI script.
9. We mapped the language to a relational database.
10. We created the implementation architecture in Pascal. Given the inherent object oriented (OO) nature of the architecture, C++ would have been an ideal implementation language choice - but the project management refused.
11. After 18 calendar months, and 6 man-years later, the factory ran perfectly!

### 1988-88: S/W Architect & Implementation: Quadriplegic PC Accessibility

*@Cubital, Herzliya - a charity project funded by the company and the CEO* [*Itzhak Pomerantz*](https://il.linkedin.com/in/itzhak-pomerantz-91500b3)*:*

1. First of all, it important to note that this project took place in 1988 when speech-to-text technology was still in its infancy, and exhorbitantly expensive.
2. The H/W used for this project was the following:
   1. A [light pen](https://en.wikipedia.org/wiki/Light_pen), i.e. an obsolete pre-mouse point and click device that synchronizes with the trace signal of the CRT video display, outfitted with special light weight military optics that increased its effective range from 5 mm from the screen to 800 mm.
   2. A standard accessibility [sip-and-puff](https://en.wikipedia.org/wiki/Sip-and-puff) switch.
3. The light pen was mounted on the user’s head by using a sturdy woman’s plastic hair head band, while the sip-and-puff straw replaced the button on the light pen.
4. The S/W that I developed overlaid a virtual keyboard on top of the screen.
5. In novice mode aiming the pen at a key caused it to illuminate, while clicking on it entered it into the system as a virtual key stroke.
6. The problem with novice mode was that the virtual keyboard obstructed more than half the screen.
7. In expert mode, the virtual keyboard was hidden, but when the pen was aimed at an individual key, it would pop-up and become illuminated.
8. This system was used to enable Shulamit Gabbai, a former school teacher who became quadriplegic by contacting polio (due to a terrible malfunction in the *Or Akiva* drinking water supply which became mixed with sewage), to become a book editor for *Maariv*. She was able to type 30 characters per minute.

### 1987-88: Senior Programmer & VAX/VMS Sysadmin: 3D Printer: Solider

*@Cubital, Herzliya:*

Solider was one of the first 3D printers. It was the size of a shipping container, but it embodied the same principles used in modern 3D desktop printers. Since a 3D printer prints one layer at a time conceptually similar to the way a regular printer prints one page at a time, I designed a VAX/VMS print driver for the 3D printer.

### 1986-86: Programmer: Automated Leather Sewing Workstation

*@Orisol, Lod:*

This is the first time I developed a [Domain Specific Language (DSL)](https://en.wikipedia.org/wiki/Domain-specific_language) in order to implement the sewing machine control program. Note that leather is a natural product. Therefore no two pieces of leather are identical, so an automated sewing program must be intelligent. The workstation included a video camera in order to enable real-time feedback. The sewing machine had a maximum speed of over one thousands stitches per minute with a very high torque, so it had to slow down carefully in order not to damage the engine. I added annotations (e.g. speed limits, and minimum and maximum margin widths) to the [AutoCad](https://en.wikipedia.org/wiki/AutoCAD) description of the pattern. I compiled the pattern and annotations into a [soft real time](https://stackoverflow.com/questions/17308956/differences-between-hard-real-time-soft-real-time-and-firm-real-time) control program.

### 1984-85: Programmer & VAX/VMS Sysadmin: Hebrew/English Word Processor: Glyph

*@John Bryce, Jerusalem.*

### 1983-84: Programmer: Real-Time

*@Elta/IAI via DSI, Ashdod:*

This was my first job upon making [Aliya](https://en.wikipedia.org/wiki/Aliyah) to Israel. Before receiving my security clearance, I worked on the development of a data collection system. I wrote a [RTOS](https://en.wikipedia.org/wiki/Real-time_operating_system) kernel for the 8080 CPU because at the time no off-the-shelf alternative was available. My mentor was Menachem Malkosh. It was a formative learning experience. After receiving my clearance, I worked on the radar system for the [Lavi fighter plane](https://en.wikipedia.org/wiki/IAI_Lavi).

### 1981-83: Programmer & IBM CP/CMS Sysadmin

*@Mitre Corp, McLean VA:*

Most of my programming was in [APL](https://en.wikipedia.org/wiki/APL_(programming_language)). The APL [functional programming](https://en.wikipedia.org/wiki/Functional_programming) mathematical vector language is still a relevant paradigm to the present day.

### 1979-80: Programmer/Economist

*@JWWA.com, an economic consulting firm in Washington DC:*

I configured and interpreted Fortran computer simulations of electric power generating systems for the purpose of costing and pricing models that were used at [Public Utilities Commission (PUC)](https://en.wikipedia.org/wiki/Public_utilities_commission) rate hearings.

### 1977-78: Intervenor/Economist

*@Ontario Energy Board (OEB), Toronto:*

1. I was an [intervenor](https://en.wikipedia.org/wiki/Intervention_(law)) at the ECAP’77 costing and pricing hearings.
2. Like all citizens, I had legal standing because I paid an electric bill.
3. I took over the [marginal cost pricing](https://en.wikipedia.org/wiki/Marginal_cost) (= peak load or time-of-day pricing) proposal of the Ontario Hydro (i.e. at that time the name of Ontario’s electric generation and transmission utility) economists who were forced to drop their case due to extreme political pressure from the metallurgy processing industry who at the time relied upon electric blast furnaces.
4. I actively participated in the hearing sessions for about 9 months. I filed submissions, gave expert testimony, and cross-examined opposing witnesses.
5. I argued my position very well. At 22 years old, I was the first public interest intervenor in the history of the OEB to be awarded costs.
6. I published an op-ed in [The Globe and Mail](https://www.theglobeandmail.com/), i.e. at the time Canada’s newspaper of record, explaining the economic and political issues surrounding the case.

## 4.0 Education

### 4.1 Formal Education

#### 1979: York University, Canada: MA Economics & Applied Mathematics

I passed my final examinations in economic theory before I even started the program, so in order to achieve my required course credits the faculty agreed to allow me to be a special student at the Univ. of Toronto graduate faculty of engineering, where I took the majority of my courses. My major project was a computer simulation in Fortran how to cost efficiently operate a hydro-electric dam. I was mentored by an economist from Ontario Hydro (i.e. at that time the name of Ontario’s electric generation and transmission utility).

#### 1977: University of Toronto - Rotman School of Management (MBA Program): No Degree

I “dropped out” of school in the middle of the year after taking an advanced micro-economic theory course which analyzed the Ontario Hydro [marginal cost pricing](https://en.wikipedia.org/wiki/Marginal_cost) submission to the Ontario Energy Board (OEB), in order to take advantage of the unusual opportunity to present Ontario Hydro’s case for them, which they were forced to drop for political reasons ([see above](#intervenoreconomist)). The following year I was able to apply my course credits to an MA Economics program at York Univ ([see above](#york-university-canada-ma-economics-applied-mathematics)).

#### 1976: University of Toronto: BA Economics & Applied Mathematics

The most memorable and still useful courses I took were in statistics, experimental design, game theory, advanced calculus, and microeconomics.

In 1971 at the age of 15, for a highschool computer science course, I wrote a computer program to play a perfect game of 3D 4x4x4 [tic-tac-toe](https://en.wikipedia.org/wiki/Tic-tac-toe) in Fortran on an [IBM 1130](https://en.wikipedia.org/wiki/IBM_1130). The computer had 16 KB RAM, and was the size of a refrigerator. It was arguably my most formative learning experience from which I received the computer programming “bug” which I carry with me to the present day.

### 4.2 Continuing Education

1. Today the field of computer science is changing so rapidly, that without ongoing self-study, one’s formal education becomes obsolete within 5 years.
2. From 1991-96, while I was CEO & CTO of Pitkha Outsourcing, I used to spend one afternoon per week reading at the Hebrew University Jerusalem (HUJI) computer science library.
3. Afterwards with the advent of the Internet, more up-to-date computer science topics were available on the Internet, so going to the library was no longer the most efficient way to keep updated.
4. Since 2005, I have maintained a subscription to the [O’Reilly Safari](https://www.safaribooksonline.com/) on-line tech library.
5. My daily dose of tech news comes from [Slashdot](https://www.safaribooksonline.com/).
6. I regularly watch [TedX](https://www.ted.com/about/programs-initiatives/tedx-program) and [Talks At Google](https://talksat.withgoogle.com/) video seminars.
7. The most fascinating feature of TedX talks is to watch and learn how world class experts in a wide range of fields are able to distill what their special area of expertise to intelligent laymen in only 18 minutes. Whenever I speak, I attempt to emulate the best TedX speakers.
8. I regularly read the tech sections of the Israeli business newspapers [Globes](https://www.globes.co.il/en/) and [The Times of Israel](http://www.timesofisrael.com/start-up-israel/).
9. I have ecclectic interests.
10. I regularly research new topics in depth.
11. My browser bookmarks are my most important professional store of my knowledge. I use Firefox because it has the best built-in bookmarking feature, because it uses tags/labels. I have a well honed tag taxonomy.
12. Therefore I really am an [expert generalist](http://99u.com/articles/7269/picasso-kepler-and-the-benefits-of-being-an-expert-generalist), and an [autodidact polymath](http://autodidactpolymath.com/autodidact-polymath-definition/), i.e. a self-learner in new fields who achieves expertise quickly.

## 5.0 Spoken Languages

1. English (5/5)
2. Hebrew (4/5)
3. French (2/5)

## 6.0 Computer Languages, SDKs, and Operating Systems

Language knowledge in order of expertise, based upon my current frequency of usage:

1. C, TCL, bash + posix text utilities, e.g. awk, sed, etc.
2. C++, python, make, html5, css, markdown, pandoc, jinja2
3. flex, bison, llvm, javascript, java, yaml, json, go
4. forth, lisp, prolog, apl, fortran, opengl, svg, xml schema, relax ng, xslt, perl, C#

Note that I write compilers and [Domain Specific Languages (DSL)](https://en.wikipedia.org/wiki/Domain-specific_language), so learning a new language takes me only a few days.

O/S knowledge in order of expertise, based upon my current frequency of usage:

1. Linux
2. Android
3. IOS
4. Win32

### 6.1 Programming Languages Musings

It may not be politically correct to say this, but C++ is too [dangerous](https://medium.com/@cscalfani/goodbye-object-oriented-programming-a59cda4c0e53) to use for the average project. *Even though during the the architecture phase of a project, it is critically important to identify common classes and objects, interfaces, and frameworks, I find that C++ object oriented implementations are highly error prone, and are sinkholes for R&D manpower resources.* Inexperienced architects and programmers create class hierarchies that are extremely brittle. In performance sensitive and embedded real-time projects most programmers do not understand the C++ language internals sufficiently in order to realize which language features cannot be used in such an environment. Therefore I prefer to write C code with an object oriented “flavor”. In my professional opinion, it is much safer to develop using the simpler, but tried and true, [encapsulation](https://en.wikipedia.org/wiki/Encapsulation_(computer_programming)), [composition](https://en.wikipedia.org/wiki/Object_composition), and [interface](https://en.wikipedia.org/wiki/Protocol_(object-oriented_programming)) programming techniques combined with C’s built-in, but underused, generic programming syntax - see [C11 generics](http://www.robertgamble.net/2012/01/c11-generic-selections.html) and the gcc/clang [\_\_typeof\_\_ and \_\_auto\_type](https://gcc.gnu.org/onlinedocs/gcc/Typeof.html) operators. My reservations about C++ are shared by such well known people as [Linus Torvalds](https://developers.slashdot.org/story/15/03/10/2038228/was-linus-torvalds-right-about-c-being-so-wrong), and [Rob Pike and Ken Thompson](https://golang.org/doc/faq). Admittedly C has many dangers too, but I feel much less than C++, perhaps because I have 30 years experience programming with it. The biggest problem that inexperienced C programmers face is memory leaks; however using [Valgrind](http://valgrind.org/) combined with well designed unit testing and [fuzzing](https://en.wikipedia.org/wiki/Fuzzing) can catch almost all memory problems.

In spite of my reservations about C++, I am pragmatic. There are some application domains where it is clearly the best choice of implementation language. For example in the case of compiler design, C++ is the best way to implement [ASTs](https://en.wikipedia.org/wiki/Abstract_syntax_tree).

Google created their new [Go language](https://golang.org/) precisely for the above reasons. It is their “go-to” language for new projects. For the vast majority of their applications, they would never even consider C/C++, because the average programmer does not have the high level of expertise required to produce large apps built with C/C++ that are bug-free. First they were a Java shop which they dropped due to its never-ending security bugs, and due to the fact that they could not control the language due to a never-ending intellectual property battle with Oracle who “owns” Java. Then they switched to free open source Python, including hiring [Guido van Rossum](https://en.wikipedia.org/wiki/Guido_van_Rossum), Python’s “benevolent dictator for life”. But they became dissatisfied with Python due to its lack of type safety, problematic package management, and its poor performance. Go is a vastly improved and safer version of C, along with an extremely well designed package manager. Go’s performance is 2-4 times slower than C. It is in the same performance category as Java and C#, while Go is about 2-4 times faster than Python. Typically Go’s performance problems arise due to its automatic garbage collector (GC) - which is a key feature expected in most modern languages. But in most cases an experienced C/C++ programmer who inherently understands how to efficiently control memory allocations can [tune Go’s GC](https://blog.cloudflare.com/recycling-memory-buffers-in-go/). Go’s performance is good enough for the most demanding web server and back-end applications, although in general the GC precludes it from being used for soft real-time applications. Go is good for client applications too. There is excellent support for many UIs. It does not yet have full mobile support, but it is on their road map. Precisely because Go is the Google’s own internal application language, the language will continue to improve. Up until now Go has made a few architectural decisions with which I disagree, namely lack of generics, asserts, and a preprocessor. On the other hand, many other languages have similar problems or worse. I can easily correct these deficiencies by using my own macro/template preprocessor - where my tool of choice is [Jinja2](http://jinja.pocoo.org/). But I am aware that many programmers cannot tolerate preprocessing their source code, unless the preprocessor is connected “at the hip” to the compiler.

For new projects that need to “press the pedal to the metal”, the new safe and efficient system programming languages [Rust](https://en.wikipedia.org/wiki/Rust_(programming_language)) and [Nim](https://en.wikipedia.org/wiki/Nim_(programming_language)) should be considered. My personal preference is for Nim especially on account of its superb built-in [metaprogamming](https://en.wikipedia.org/wiki/Metaprogramming) capability. But Nim is currently supported by only a small team mostly of volunteers, while Rust has much more market traction because it is backed by the Mozilla organization.

Again pragmatism prevails, in spite of my theoretical language preference for Go over Python, or dabbling with Nim and Rust, because there are many application domains where Python has superb 3rd party libraries that would reduce the development time of a proof of concept by at least a factor of 3 (e.g. [NumPy](https://en.wikipedia.org/wiki/NumPy), [SciPy](https://www.scipy.org/), [Matplotlib](http://matplotlib.org/), [TensorFlow](https://www.tensorflow.org/), [data visualization](http://opensourceforu.com/2017/03/python-libraries-data-visualisation/), etc.), so choosing to use Python is very often the most productive decision by far.

Finally, I find for creating specifications for novel and complex ideas, I like to start by brainstorming with [mind maps](https://lifehacker.com/five-best-mind-mapping-tools-476534555) and then move to [TCL](https://wiki.tcl.tk/15542) for creating a formal specification which typically morphs into a [domain specific language (DSL)](https://en.wikipedia.org/wiki/Domain-specific_language).

## 7.0 Patents Under Development

1. [Bioinformatics](https://en.wikipedia.org/wiki/Bioinformatics): (a) An extremely accurate and simple noise reduction and normalization algorithm to improve the accuracy of the standard [PCR Ct](http://www.thermofisher.com/il/en/home/life-science/pcr/real-time-pcr/qpcr-education/pcr-understanding-ct-application-note.html) calculation, and (b) an [Artificial Intelligence (AI)](https://en.wikipedia.org/wiki/Artificial_intelligence) methodology for measuring the quantity of DNA in a bioassay where [inhibition](https://en.wikipedia.org/wiki/Polymerase_chain_reaction_inhibitors) makes it impossible to estimate the Ct because no underlying [logistic function](https://en.wikipedia.org/wiki/Logistic_function) (= a flat “S” shaped curve) exists.
2. [Cryptography](https://en.wikipedia.org/wiki/Cryptography): A set of non-linear cryptographic primitives using [Hamming weight](https://en.wikipedia.org/wiki/Hamming_weight)-like [data dependent permutations](https://link.springer.com/article/10.1007%2Fs00145-001-0012-9?LI=true) which overcomes the well known limitation of using Hamming weights because they have a [binomial distribution](https://en.wikipedia.org/wiki/Binomial_distribution).

## 8.0 Personal

I was born in Canada in 1956. I have lived in Jerusalem Israel since 1983. I am married with 4 children, 2B + 2G, plus many grandchildren. I take physical fitness seriously. Once upon a time I was a judoka, and a classical guitarist. I was an IDF reserve soldier for 15 years, where I served as a combat soldier in the infantry in the Jordan Valley. In spite of the fact that I joined the army when I was 32 years old (Hebrew: *Shlav Betnik*), functionally, but unofficially, I served in the capacity of deputy company commander (Hebrew: *Samech Mem Pe*) which provided me with the opportunity to achieve rich personal growth, and enabled me to learn important managerial and leadership skills.

1. This secure photo was built using an [oilify](http://docs.gimp.org/en/plug-in-oilify.html) filter and a visible email address [watermark](https://en.wikipedia.org/wiki/Digital_watermarking). It uses [digital camouflage](https://io9.gizmodo.com/5510040/designer-reverse-engineers-face-detection-tech-to-develop-camouflage-makeup) that will defeat almost all face matching algorithms - in spite of the fact that my 3 year old grandson has no problem identifying me from this photo. It protects my privacy in two ways. (1) Harvesting it from the Internet, and adding it to a database of photos that will be compared with CCTV surveillance photos will not work. (2) The photo has built-in [two factor authentication (2FA)](https://en.wikipedia.org/wiki/Multi-factor_authentication). The prominent email address watermark prevents a web site from maliciously displaying my photo while attempting to associate/label it with any other name besides mine. Similarly if a site were to maliciously display my photo without my permission in order to imply my association or agreement with them, then it is trivial for anyone who may be suspicious of their claim to “challenge” the site by asking me for a verification email. [↑](#footnote-ref-2)