William Zeitler

william@WilliamZeitler.com
323-791-8112
Highland, CA (1.5 hours east of Los Angeles)

Systems Software Developer, C/C++, Linux/Windows

*** Krystallize, Inc.** (Nov 2016 to present)

Primary developer for two C/C++ applications, both core/critical to the company:

1) 'pqos' (C): I inherited a multi-threaded application that aggregates system OS metrics (such as file bytes written/second, CPU% utilization) and can optionally create 'synthetic' artificial loads (loading the CPU by calculating primes, exercising the file system by reading/writing scratch files with random contents, etc.). Pqos aggregates said metrics, packages them into JSON packages, and uploads the encrypted data packets (AES128) to a 'dataplex' of LAMP servers which stores the metrics and associated metadata in Postgresql.

- I substantially refactored pqos to correct algebraic and statistical errors, and dramatically improve stability, performance and maintainability. Corrected numerous memory leaks/stray pointers that only manifest themselves after several hours of execution.
- Refactored the source code and build environment so one Linux executable would execute on: RHEL 7.x, Centos 7.x, Suse 12.SP4, Suse 15, Ubuntu 14.x, Ubuntu 16.x & Ubuntu 18.x, and one Windows executable would execute on Windows 7, 8, 10, WinServer 2008, 2012, 2016, and 2019 without adding/modifying any bits on the target machine (other than copying the executable onto the machine).
- Added the ability for pqos to determine if it was running on one of the big-four cloud providers (AWS, Google, Azure or Softlayer) and retrieve the available metadata, ultimately uploaded in a JSON packet to the data warehouse.

2) 'AWP' (C++): The idea of "Automated Workload Profiling" (AWP) is that suppose a client has an application that they'd like to migrate to the cloud, but installing the application is problematic for security, regulatory or logistical reasons. So the client can use pqos on the original machine to determine what kind of load the original application puts on its machine, then use AWP to determine from the metrics data the parameter vector for the pqos 'synthetic load' engine that approximates that load. Then the client can put pqos and that synthetic load parameter vector (an ASCII config file) on whichever providers/SKUs they want to determine which provider/SKU is the best fit.

I built the algorithms and wrote the application to determine the solution parameter vector given a 12-dimensional metrics data pool using an iterative "Newton's method" approach combined with linear algebra, statistical modeling, finite state machines, etc.. Of particular challenge is that usually statistical modeling requires a lot of data (think statistically modeling the stock market and the fire-hose of data available to do that). For this use case, however, each input metric sample vector is expensive – on the order of 30 minutes each. So if one were to guess-timate the number of iterations needed, one would need 2 points data points per metrics parameter (2 points just to define a linear approximation) suggesting that 24 iterations would be optimistic, giving a total runtime of at least 12 hours. In practice, however, my algorithm/implementation has consistently converged 90% of the time to within 90% of nominal in roughly a dozen iterations.

Pqos/AWP have been successfully tested on machines ranging from 2g memory/2 cores to 6t memory/384 cores.

4) Responsible for maintaining/enhancing the PHP/LDAP/Postgresql on the dataplex (a cluster of four servers) that authenticates the pqos connection, processes the encrypted JSON packets, and inserts the data into the database.

5) Wrote a WebApi (REST) in PHP giving clients basic CRUD capability for their own data. Approximately 25 APIs (plus all the expected user options: download data format (JSON/CSV), which columns are wanted, sort order, etc.). Wrote a set of python scripts to validate every API and a representative sample of its options (85 scripts in all).

6) Dataplex Installers. The 'dataplex' consists of four LAMP servers connected by SSH tunnels, and the LAMP servers require Postgresql, Apache, PHP, LDAP, etc. Wrote a 'meta-installer' that generates all the necessary SSH key pairs and generates the four installers --- one tarball for each server. Each installer installs the necessary prerequisites for that server, updates the various system config files, creates service accounts, creates cron jobs, installs the correct SSH keys (including initializing 'authorized_keys'), installs https/TLS certs, installs the source code – everything necessary to spin up a complete functioning dataplex. Even though the company distributes the dataplex via Vmware images, etc., these scripts are still necessary to reliably build the source images.

* Informulus, Inc (2013 to 2016)

Wrote a SAS parser (C++/ANTLR). SAS is a 70s language not at all written in conformance with modern computer language practices. Among its many 'challenges', the worst is that keywords can also be used as variable names (thus you can define variables IF, ELSE, THEN and have a legal SAS statement like "IF ELSE=IF THEN IF=ELSE ELSE THEN=IF;") So standard compiler theory / a classical lex/yacc approach won't work because lex/yacc assumes that reserved words aren't used as variable/function names. I wrote a parser that successfully builds an Abstract Syntax Tree (from which symbol tables and more can be extracted). Successfully tested against a test bed of 180 representative SAS programs taken from industry production environments.

* NetProfession, Inc. (2010 to 2013)

Wrote a CMS (LAMP) that included user-buildable web forms, credit card processing, and blogging (using .htaccess redirects like WordPress does).

* Glass Armonica Performer (2002 to 2010)

The glass armonica is a musical instrument invented by Benjamin Franklin in 1761 that works on the 'wet finger around the wine glass rim' idea, for which Mozart, Beethoven and others of that era composed. There are maybe a dozen players in the world capable of playing the Mozart pieces and I'm one of them. I had the glasses blown (ground them to pitch myself), built an instrument, and engineered a way to safely and economically fly with $40,000 of hand blown quartz glass.

Performances included the Blue Man Group, the Kennedy Center, Europe and Asia; my hands playing the glass armonica were a clue on Jeopardy. A YouTube video of me playing "Dance of the Sugar Plum Fairy"

on the glass armonica has over 5 million views (search for ‘glass armonica sugar plum zeitler’). I can be heard prominently in the HBO film “Taking Chance” immediately following the opening title sequence.

Wrote a 300 page book on the history of the glass armonica (the non-fluff bibliography is 25 pages long). Translated substantial passages from the original sources in Latin, 18th century German and French into English. Trips to rare book reading rooms (locked rooms and gloves required to handle the books) at UCLA, Stanford, the Library of Congress and others were necessary.

*** Boeing** (1999-2002) Senior Developer: C++, IIS/VB/MsSql.

Wrote COM objects in C++ for use in VB programs that performed line-critical statistical analysis of manufacturing processes (generating 40 different statistical analysis charts).

The system also included a “Very Primitive iPad” (“VPI” <grin!>) with which the line mechanics could pull up ‘measurement plans’ and plug in various measurement devices; the VPI would display a diagram of where the measurements were to be made, do sanity checks of the resulting measurement data, and later upload the data to a PC via RS232 which uploaded to MsSql. When I came on board the interface between the VPI and the PC was badly broken. No documentation on the protocol between the VPI and the PC could be had. So I captured the binary data stream and reverse-engineered the protocol. When I started, the upload process took 30 minutes when it didn’t crash; when I finished the upload process took 20 seconds and worked every time.

*** Microsoft** (1997-1999) Developer on the C++ team at Microsoft. As a contractor I was given the ‘scut’ tasks that needed to be done – generally difficult and/or unsexy but essential tasks, such as: grepping the entire MFC and C++ tree for Y2K issues, setup (an unexciting but absolutely essential function), , But seeing from the inside how world-class software (Visual Studio C++) is shipped was one of the great educational experiences of my life.

*** Edmark** (1996-1997) lead developer (Windows/C++) at Edmark, Inc. (grade-school children’s multimedia educational software designed to run successfully in a low-end environment – because kids get the hand-me-down machines from their parents). Pioneered the idea that in addition to their high-end development machines, the developers should have on their desks a representative low-end machine for testing.

The software presented cartoon-like images (e.g. “Sammy the Snake” in “Sammy’s Science House”). A frequent request was for the ability to print ‘coloring book images’ (wire-frames) which kids could color. The art department balked at the time it would take to create the necessary wire-frame images, and there wasn’t space on the CD-ROM deliverable anyway. The engineering department also threw up their hands. I came up with an algorithm to scan the color images looking for color boundaries based on deltas in color Pythagorean distance (R^2 + G^2 + B^2). The resulting wireframe/coloring-book images were surprisingly good, even for problematic image elements like ‘fire’.

Other

***** Mathematics Instructor at Monterey Peninsula College (3 years). Mostly Statistics and Algebra.