Implementation & Testing of a Book Lookup System for the Robert E. Kennedy Library

By

Casey C. Sheehan

Computer Science Department
College of Engineering
California Polytechnic State University
Fall 2016

Date of Submission: 9 December 2016

Advisor: Franz Kurfess

Contents

ABSTRACT

The goal of this senior project centered around improving the quality of student and teacher experiences when visiting the library. The task of finding a book amongst the shelves is an arduous one, which I felt could be improved upon through implementation and testing of a Book Lookup system for the Cal Poly Robert E. Kennedy Library. Development for this project was done using a Python framework. Testing and earlier designs were also created using JavaScript and PHP. Repeated tests were conducted on the accuracy of the software and its ability to decrease user search-time when compared to conventional methods.

LIST OF FIGURES

LIST OF TABLES

Project Overview

Background

Cal Poly’s Robert E. Kennedy currently houses in excess of three hundred shelves worth of books among its five floors. Patrons of the library looking for a specific book must record a book’s call number--an alphanumeric identifier given to each book--and compare it shelf by shelf with the listed ranges of call numbers on each. This process can be tedious, and is one which can be vastly improved through the precise manipulation of existing technologies to better suit the present status of the library.

Intended Users

End users of a book lookup system such as this are almost entirely classifiable as members of the campus community--i.e. Cal Poly students, faculty, and staff. While faculty and staff may be more familiar with gathering references from the library, many students may have never had to access a book using these means.

Difficulty and Relevance

Utilization of skills and technologies pursuant to the interest of implementing such a system is wholly relevant as the basis for a Senior Project. To create such a system requires technical refinement in the form of creating normalization and search algorithms to achieve the system’s overall goal. Additional consideration needs also be placed in the thorough testing of such a system. And, finally, the benefit of the project would be readily apparent to the campus community.

Requirements and Features

The initial outlined features for the book lookup system were stated previously in the project's proposal. During implementation and testing, the requirements themselves changed noticeably in order to address certain changes to the library itself during the course of the program's creation. Nonetheless, the initial specifications of the book lookup system remain a strong basis for introducing its intent:

- Approachable UI
    - Limited user interaction to achieve goals
    - Graphical representation of numeric data (shelf and call number)
- Improved Efficiency
    - System should be tested against conventional methods to demonstrate a decrease in the time taken to find a
- Integration with current lookup System (Polycat)
    - Uses Polycat API and GET requests to fetch information such as call number, section, etc.
    - When book is available in library, Polycat shows a single button titled, "Find it in the Library"
    - Clicking the "Find it in the Library" button will open up a new window, in which a map of the library floor which the book is located on will appear
        - The current book's shelf will be highlighted, and information at the bottom of the page will display other information.
            - Other information is floor number, shelf number, and call number
    - When book is not available in the library (i.e. digital copy only), no button is shown
    - System should stay updated as polycat Databases change
        - GET requests done in real time
- Design should be consistent with current themes and templates
    - Where applicable, use of Cal Poly applicable web themes should be used
- Approach should be simple and lightweight
    - Minimal reliance on third-party software for ease of version control
- System should be editable as shelf locations and ranges change
    - Administrative UI should have ability to modify database for call ranges
- Softwares used should be compliant with any testing tools
    - System should not use technologies/languages unsuited for environment
    - Testing libraries need developed

After spending two quarters implementing and testing the Book lookup system, I feel it fitting to add the following addendums to the requirements above:

- Above all else, System should be flexible and easily changed
    - No use of embedded code to represent map locations
- System administration done continuously throughout development cycle
    - For such a system to work, all changes to the library need to be thoroughly documented as they occur

Justification for these modifications may be seen within the Evaluation Section.

Evaluation Criteria

In order to evaluate the book lookup system upon its completion (or, perhaps, before), the following criteria were identified:

1) Reduced search latency--quantifies the extent to which, if at all, the book lookup system improves the average search time for a particular book
2) Accuracy of algorithm--how often the system is able to correctly predict a book's shelf location
3) Scalability and Durability--qualitative analysis over whether such a system is scalable and easily changed to suit need

Technologies Used

Implementation of the book lookup system occurred primarily using two different frameworks; initially, a JS/PHP was used in an effort to adjoin any aspect of the book lookup system to the existing library website. The code base for the project at that point also made use of HTML and CSS. Coding was primarily done using Adobe Dreamweaver. Adobe Illustrator was also used in the creation and manipulation of SVG files for the library floor maps. These SVGs have since been redesignated as flat images (more on that below), and edited again using Illustrator. In terms of technologies provided and used by Cal Poly, the library site is run using Wordpress, and the PolyCat subsite utilizes the MooTools JS library. The site's embedded use of of google analytics has also been considered thus far. Additionally, referential code bases were also found in the publically available source code for similar book lookup systems used by Brown University and Ryerson University. The Ryerson implementation has influenced this in that both have been designed to make use an internal SQL database.

Second-wave implementation of the Book-Lookup System (BLS) involved honing the normalization and sorting algorithm in order to further improve reported accuracy.During this wave of implementation, less focus was placed on the end-user aspects of the system, as during creation it became apparent that the combination of the BLS with existing library systems was would require constant, time-consuming interaction in order to maintain. Again, more information regarding these findings is available in latter sections. To this end, then, the

second-wave implementation made use of MySQL, Python and CGI scripting within a LAMPP framework.
Returning for a moment to the discussion of maps, the aforementioned library SVG files are used by the main library maps page, located at http://lib.calpoly.edu/maps/ . These maps display information about currently available library seats, computers, and study rooms. An example SVG map is shown below:

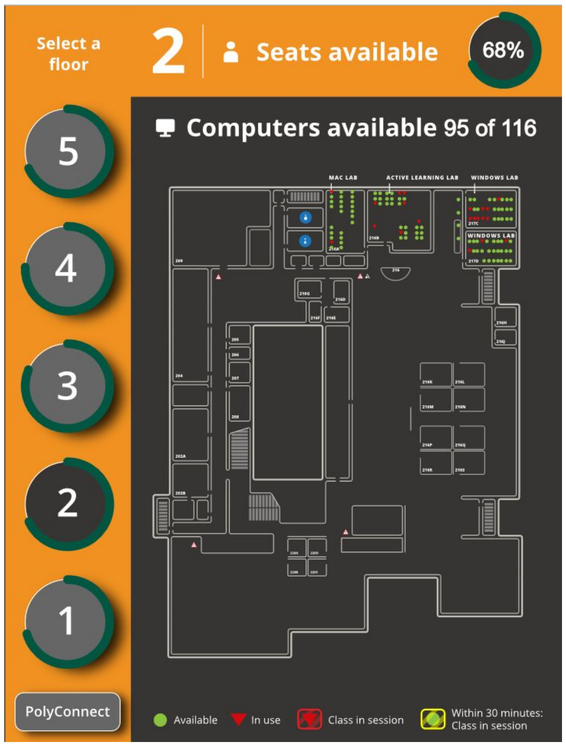

Figure 1: A Robert E. Kennedy Library SVG Map

Another form of campus map found during this project was an internally-used library floorplan. These maps, unlike the SVGS, accurately show where each shelf is located. For this reason, as well as for the sake of relative simplicity in implementation, these floorplans were chosen:

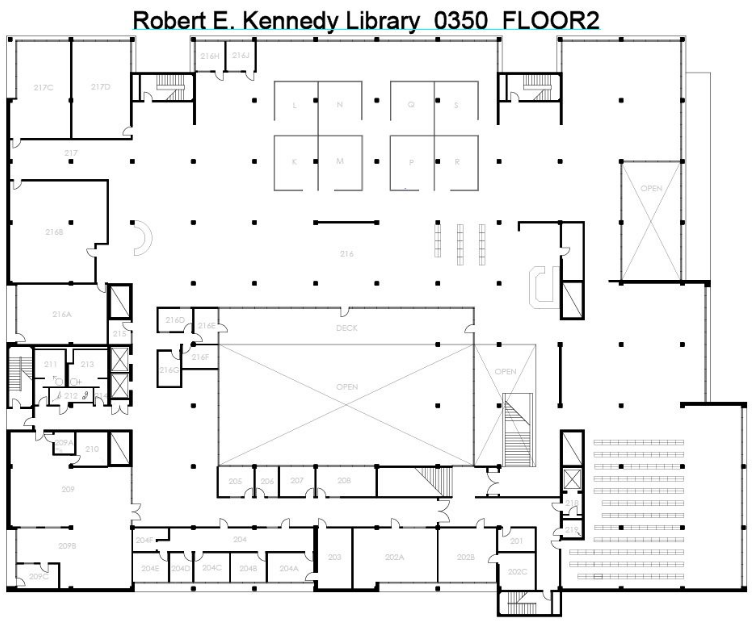

Figure 2: An Internally-Used Library Floorplan

Latter portions of this project also involved significant unit-testing of inputted call-numbers, matched to their intended normalized counterparts. To this extent, regex101, a unit-testing website, was utilized. Further unit testing was conducted using output scripts of CSVs containing the normalized shelf range call numbers, which was compared to intended values using the software Beyond Compare 4.

Research

Initial research into centralized around answering the following three questions:

1. What systems (if any) exist that accomplish this task?
2. What is the underlying basis by which these systems work?
3. How might such a system be developed for use by our library?

Answering the first required some research into existing codesets. As mentioned before, two such sets were found: that of Brown University, and that of Ryerson University. These two offered staunchly different approaches to the problem at hand. The Brown University System utilizes a process known as normalization to compare book call numbers to stack call number ranges. Before discussing normalization, however, it is important to first understand how call numbers function.

Call numbers are alphanumeric in nature, and are often segmented into three or four sections. A typical call number, as it appears on a book, looks something like the following:

DS
559.45
.H35
1986

The first line is made up of a sequence of characters indicating general topic. In this case, DS represents general Asian history (USG). The second line contains a series of numbers which, when read as a whole, designate a specific topic--in this case, the Vietnam War. The third line designates the author's last name--Halin. Finally, the fourth line is the book's publishing date. This fourth line is not necessarily included. However, books by the same author and on the same topic are organized chronologically.

Normalization, then, is the process of making these call numbers comparable to one another. More specifically, a lookup implementation needs to consider whether a given call number falls *between* two call numbers referencing the start and end of a shelf. Since not all call numbers utilize exactly the same format, a generalized technique is needed to render all of them useable in such a manner. The Brown University approach uses a technique of padding unused areas within a call number to render them comparable. For example, the call number "A 123.4 .c11" would normalize to, " A 012340C110" (Brown). Note that the section characters are offset by a space from the remaining alphanumeric characters. This allows general topic to be more readily used as a search index through a database of shelf ranges, as is the technique utilized by Brown. The Ryerson approach to the comparability problem was not as elegant or unilateral. Any system implemented for the Kennedy library needs to be able to accommodate an ever-changing collection of shelves and call numbers. Therefore, an elegant implementation of normalization, such as that of Brown University, was deemed necessary.

For reference, a table displaying a series of exemplary call numbers and their normalized counterparts is included below:

Call Number	Normalized Call Number
DS556.8 L25 2000	DS 055680L250 000 000 2000
HM 22 F8 D774 1992	HM 002200F800D774 000 1992
N1 A43 V.1	000100A430 000 000 V1
ND 1650 E471	ND 165000E471

Table 1: Example Call Numbers and their Normalized Counterparts

Returning to the differences between the Brown and Ryerson implementations, Brown University utilizes the Django framework, while Ryerson utilizes a PHP/SQL approach. Further, the Brown system's graphical representation is done almost entirely with pure HTML/CSS, while the Ryerson implementation makes use of powerful javascript overlays. Visually and subjectively, the Ryerson system delivers a better representation of book location.

After analyzing the first two questions in the context of a comparison between two existing systems, the answer to the third question began to take shape in the form of an initial system design. The two major design challenges, therefore, were to create a well-modeled database representing library shelf information, and to make this database model comparable to incoming call numbers. This syllogism of the design process proved inapprope, however, as the more relevant question of *how* the call numbers would be retrieved arose.

This realization turned research toward understanding the PolyCat API. PolyCat is the web service that, along with find.lib.calpoly.edu, provides the search functionality for Cal Poly's digital and physical collections. After discussing the design of PolyCat with library technical staff, it became apparent that there was not much of an API for the system, per se. In lieu of spending further time developing such an API for the sole purpose of this project, a prospective workaround was found for this issue. More information on this solution comes in the following section, which discusses the overall project workflow in greater detail.

Ultimately, initial research into a prospective book lookup system yielded promise and valuable understanding of the underlying techniques needed. With the first two questions solidly answered and a system design hypothesized, questions at this point give way to action in the form of code.

System Design & Architecture

At present the system has two relevant architectures, one antiquated and one functional.

The antiquated architecture utilizes a web framework of PHP, JavaScript, and CSS. User input is recorded on a web form requiring them to input the call number (non-case sensitive). From there, the information is dispatched via a file, lookup_request.js, which calls the normalization function on the inputted data and passes it to another function responsible for normalized matching. Both the normalization function and comparison algorithm are maintained in a file called parse_lookup_request.php. From there, the comparison algorithm is executed, querying a mySQL table titled "call_ranges" in order to compare the given call number to the ranges of each shelf. When a match has been detected, the program terminates, and information from the relevant table row is collected. The comparison algorithm returns a several pieces of information, which are passed to the browser via the address bar: the normalized call number for testing purposes, as well as the shelf number, floor number, and coordinates. On load, the shelf and floor number arguments are parsed in order to create a display box. The x and y coordinates are taken and used to create a highlighted shelf overlay.

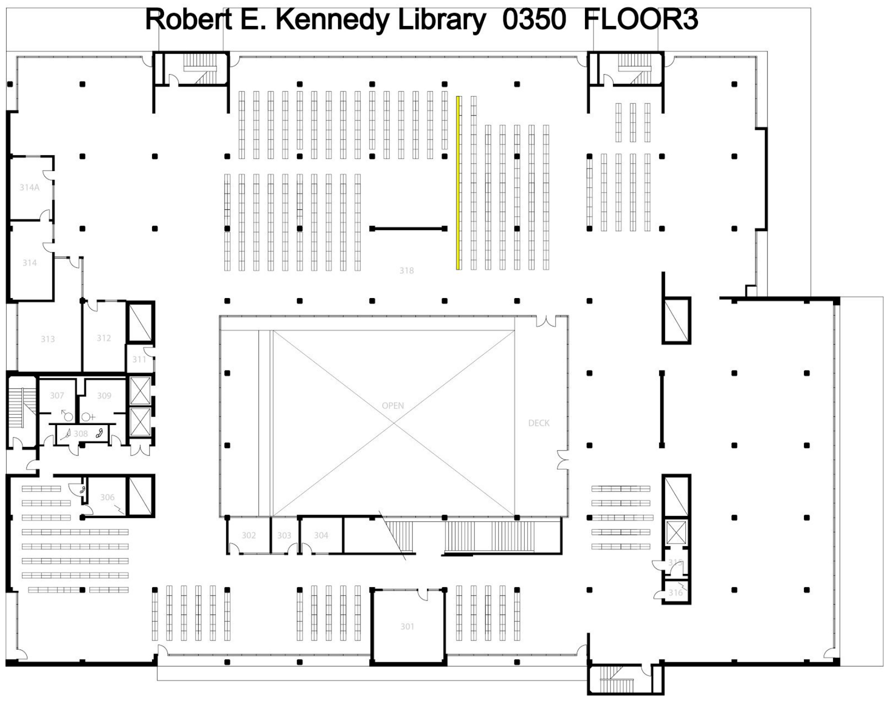

Figure 3: A Floor Plan Map with Highlighted Shelf

After completion of phase two of evaluation of accuracy testing, this approach was forgone in order to attempt to improve overall accuracy and allow for easier modification. The current framework was created within Python. A first file, callnumber.py, is responsible for objectively representing a normalized call number. Within callnumber.py is also the algorithm responsible for normalization. A second file, normalize_stacks.py, is responsible for taking in a data set representing non-normalized shelf call numbers. Each call number is then normalized, and output to a second data set for later use. In order to improve response time and reduce overhead, this normalized data set is stored as a CSV file, as opposed to a conventional mySQL table.

A third program, user.py, is responsible for querying user input of a call number. From there the inputted call number is normalized, and then sent to the comparison algorithm in order to determine the actual floor and shelf. A sample of the terminal output of user.py is included below.

```
Enter call number to normalize and search: DB 661 S45
Normalized Call Number is:  DB 066100S450
Searching...
match on fl  3 row  7
```

Table 2: A sample response from User.py

Validation and Evaluation

As mentioned within the section of the same name, the evaluation criteria for completion of the BLS centered primarily around two aspects: (1) Accuracy of the matching algorithm in finding the correct shelves and (2) End-user reduction of search time. To test the former, three rounds of Accuracy testing were completed. Accuracy testing involved performing tests on the reported shelf for samples of twenty books. These test occurred during weeks 2, 4, and 9 of the Fall Quarter 2016.

Accuracy Testing Phase 1

The initial analysis of the matching algorithm showed a reported success rate of 16 out of 20 instances; 16 out of 20 times, the correct shelf and floor were reported for a given book. Of the four remaining incorrect, 3 were off by a factor of 1 shelf, while the final book produced an error during testing. This error was a result of a notable prefix unique to this book: Ovr, indicating an oversized book.

Accuracy Testing Phase 2

Two weeks after the initial test, I again tested the accuracy of the algorithm, which at that point had received modification in order to be able to receive oversized books, and further utilized further comparison between call numbers based on the third alphanumeric sequence within the normalized number strings. This second test produced a correct response in 14 out of 20 books tested. When searching for a reason as to why this number would in fact decrease, despite attempts at algorithmic improvement, it became apparent that the shelf numbers within the library had actually changed significantly since the original database propagation. In fact, as of this writing, the fourth floor currently contains in excess of 20 newly vacant shelves. Vacancies within shelves was an unforeseen consideration; up until this point the underlying data structure had relied on a full sequence of shelf values with no gaps. The results of this second test revealed the need to restructure and propagate the underlying database.

Further, as the shelves changed in amount within the library, so too did many of their physical locations change. In several cases, many of the underlying x and y coordinates used to map the location of specific shelves displaying incorrectly, or else highlighting the wrong shelf.

Accuracy Testing Phase 3

To address the needs resulting from testing phase two, several changes were made to the BLS implementation. First, the database was modified to allow for vacant ranges. Additionally, shelf call numbers were again reanalyzed through a tertiary study of each in order to ensure accuracy. A script was also constructed to reliably take these call number ranges and normalize them for use within the comparison algorithm. Scripting of this task allowed for much easier manipulation of data without need for excessive administrative oversight.

At this point, too, the attempt to utilize x and y coordinates within the database was abandoned. Implementation at this point instead focused on improving accuracy of reported shelving numbers. After updates were completed to the shelf numbers database, the final round of accuracy testing provided correct results in 18 out of 20 instances.

Usability Study

In order to evaluate the BLS' performance based on end-user reduction of such time, a small usability study was conducted in which 5 participants were asked to find a book using both the conventional method as well as the BLS. Measurement was done via recording the time between initial reception of the book's call number, to when the book was finally found. Travel time from the first floor to a book's respective floor was discounted.

After surveying 5 participants, the average search time using the conventional method was recorded at 5 minutes and 37 seconds. On the other hand, using the book lookup system, the average search time dropped to 3 minutes 43 seconds. This constituted an improvement of 1 minute 54 seconds. That is, participants spent an average of 34% less time searching for a book using the BLS than did with conventional methods.

Conclusions

Findings

This first section will be dedicated to the insights gained throughout the implementation. Foremost, unit-testing quickly revealed the applicability of a call number normalization algorithm to decreasing average search time for end users. This aspect was again confirmed within the scope of initial usability surveys, which showed that test subjects were spending on average 1-2 minutes less time finding a given book. These numbers could perhaps be improved further through the use of more noticable labeling of individual shelves.

Each of the three phases of accuracy testing offered some insight into the usefulness and practicality of the matching algorithm itself. The first test demonstrated the initial accuracy of call number normalization to suit the needs of Robert E. Kennedy library. The second accuracy test revealed necessary changes to the initial requirements outlined previously. Finally, the third accuracy test offered insight into unique edge cases (specifically, oversized books) and how to rectify them.

Critiques and Considerations

Given the consistent changes occurring to the library's physical collections, the implementation approach outlined here, when taken as a whole, does not well meet the current environment in which it would be deployed. Specifically, the choice to implement a hard-coded representation of shelf locations, i.e. embedded x and y coordinates for use within a CSS overlay highlighting the relevant shelf, is not an ideal design strategy when taken in consideration with the fact that the library's shelving locations are constantly changing--and have been throughout this Fall quarter.

Another aspect of the BLS that may perhaps be refined lies in the actual images used to represent the library's individual floors. Since the physical layout of the library is fluid, a heavily simplified solution would perhaps be ideal in the interim, as opposed to one focusing on usability and aesthetics.

The decreasing number of physical books and the resultant vacant shelves also brought out another subtle issue: initial construction of the MySQL database representing shelves' call ranges did not allow for vacancies within shelves. This was fixed relatively quickly within the database, but the matching algorithm for normalized call numbers also needed to be changed in order to accommodate this change.

Future Work

The three areas of critique listed above provide an excellent segway into future actions. In order to hone the BLS to better suit the needs of the Robert E. Kennedy Library, the following actions ought to be taken:

1. Physical book locations to be removed from the Database
2. Image-maps to be replaced with pure HTML/CSS representations
3. Further modification of the matching algorithm

Taking a moment to speak on the first two, the utilizing pure HTML/CSS for the library maps would ameliorate the need for constant and frustrating changes to DB x and y location values. Instead, the matching algorithm could simply pass as an argument the unique html identifier of the target shelf to be utilized within an asynchronous javascript function to append a unique css class to the target shelf, which would stylize it has highlighted. In this way, mapping could be completed entirely independent from that matching algorithm. This would also further simplify the means by which the database entries are updated as shelves change. All that would need to be done to modify the data structure using this approach would be to recompile the normalize_stacks.py file using a modified CSV to reflect the new shelves.

The matching algorithm can also be further improved with the addition of a standardized means of labeling each shelf. To explain, the underlying data structure would need to be modified so as to have the ability to actually represent two ranges: the primary start and ending call numbers, as well as optional ranges for oversized books. Since oversized books were the continuing outliner within unit testing, it seems apparent that they will need their own unique representation since they do not follow a consistent pattern with respect to the primary shelves' call numbers.

In some ways, it is difficult to be so critical of a system which I have spent two quarters developing, and yet it is only by acknowledging these aspects that the system may hopefully find its way into use by the campus community.

Citations

Brown University Book Lookup System. Bonnie Buzzwell, Ted Lawless; https://drive.google.com/file/d/0BzPT8o4kv5YySWgwU3h0RFg3VzZIUjJWeWhaMkRGaGF6QkZ3/view

Cal Poly Campus Library Maps, Robert E. Kennedy Library; http://lib.calpoly.edu/maps/

CallNumberLC Normalization Library. Done in conjunction with Brown University and MIT; https://github.com/libraryhackers/library-callnumber-lc

RULA Book Finder. Ryerson Library Staff; https://github.com/ryersonlibrary/rula-finder

The Library of Congress Classification System. University System of Georgia; http://www.usg.edu/galileo/skills/unit03/libraries03_04.phtml

Understanding Library of Congress Call Numbers. University of Buffalo; http://library.buffalo.edu/help/instructional/loc.html

Appendix

1. Normalized Call Numbers for the Library's third floor

```
3,1,DS 000400D660,DS 013500G400F664 000 1998
3,2,DS 013500 000 000 000 64R4851984,DS 055680L250 000 000 2000
3,3,DS 055680M360 000 000 1995,DS 073490N300F640 000 1984
3,4,DS 073490T330L500 000 1996,DT 002000B460 000 000 1975
3,5,DT 002000B460 000 000 1984,DT 065800N430 000 000 1967
3,6,DT 065800Y600,DA 008622D700K730
3,7,DA 009000 000 000 000 041978,E  007800C150C698 000 1976
3,8,E  007800C150C700,E  009900O300H650 000 1995
3,9,E  009900O300H900,E  017600 000 000 000 573V4
3,10,E  017600B590,E  018380S700L480
3,11,E  018380S400T800,E  018400S750J320
3,12,E  018400S750L345 000 2006,E  018593S700M430 000 2006
3,13,E  018593S400N400 000 1973,E  033780B900 000 000 1968
3,14,E  033780C148 000 000 V1,E  045700T427 000 000 1952
3,15,E  045700T427 000 000 1953,E  048700D260 000 000 V1
3,16,E  048700D260 000 000 V2,E  074800S840D600
3,17,E  074800S840N360,E  087300J550
3,18,E  087300L370,F  012850D770 000 000 1996
3,19,F  012850E380,F  037900N553M553 000 2005
3,20,F  037900N553Z454 000 2009,F  081100E400
3,21,F  081100F230 000 000 V1,F  086900S450G400
3,22,F  086900S450H320,F  122100O860B470 000 1989
3,23,F  122100O860D700 000 1986,F  146520T700B590
3,24,F  146520T900C370 000 1997,F  121930A700B300 000 1968
3,25,F  121930 000 000 000 87B39,GB 098000 000 000 000 093
3,26,GB 098000R870 000 000 2013,GN 000400H580 000 000 1966
3,27,GN 000400I520 000 000 1952,GN 063500M400L330 000 1983
3,28,GN 063500M400M250 000 1997,GV 019167W500W450 000 2004
3,29,GV 019170H340 000 000 2009,GV 175100H300A800 000 1976
3,30,GV 175100H830,HA 021400U580 000 000 1969
3,31,HA 074500M640 000 000 2001,HC 017500B665 000 000 2005
3,32,HC 017500B745 000 000 2006,HD 147600T760R880 000 1985
3,33,HD 147600U500B470 000 1985,HD 808100A650B570
3,34,HD 808100A650H370,HE 020300F360 000 000 1973
3,35,HE 020300H250 000 000 1994,HF 204600S450
3,36,HF 233050H400 000 000 2006,HM 002200F800D774 000 1992
3,37,HM 002200F800D779 000 1973,HM 026100A100G340 000 2008
3,38,HM 026100A100G340 000 2008,HN 005700B450
3,39,HN 005700B450 000 000 1965,HN 038500S640
3,40,HN 038500S770,HQ 002100G363 000 000 1985
3,41,HQ 002100G586 000 000 2014,HQ 068200W484 000 000 2002
3,42,HQ 068250K542 000 000 1998,HQ 079200U500C432 000 1993
3,43,HQ 079200U500C4325 000 2013,HQ 115400F447 000 000 1978
3,44,HQ 115400F4473 000 000 1986,HQ 142600W667
3,45,HQ 143800A110K630,HT 1662454300 000 000 000 2009
3,46,HT 16624700 000 000 000 2003,HT 152100R335 000 000 1992
```

```
3,47,HT 152100R340 000 000 1995,HV 088300C200P490 000 2012
3,48,HV 088300C200S730,HV 581000S840 000 000 1994
3,49,HV 581000W240 000 000 1991,HV 654300H8043
3,50,HV 653400J360T360 000 2002,HV 862100R900S370 000 2002
3,51,HV 862100U500G550 000 1984,JA 008400R900J350
3,52,JA 008400R900U800 000 1964,JK 052100K430
3,53,JK 052100M370 000 000 1981,JS 007800K360 000 000 2011
3,54,JS 007800R430 000 000 1997,JZ 151900L380 000 000 2001
3,55,JZ 154100P470 000 000 2001,KF 474550G500
3,56,KF 474800A200 000 000 1972,LB 102830M600 000 000 1988
3,57,LB 102830M625 000 000 2009,LB 282275E780 000 000 1991
3,58,LB 282275H355 000 000 2010,LC 511500H380 000 000 1971
3,59,LC 511980C370 000 000 2006,LD 072960S520H600P480
3,60,LD 072960S520H600P530,ML 012800R600H620 000 1986V2
3,61,ML 012800S250C580,ML 041900M790K600 000 1991
3,62,ML 051900O858A300,MT 000600G5245
3,63,MT 000600G597M900,ML 041000V400V290
3,64,ML 041000W100B1853 000 1975,ML 0422
3,65,N  000100A430 000 000 V1,N  530000L730
3,66,N  530000M865,N  651000M600 000 000 1978V2
3,67,N  651000P370 000 000 1999,N  691900E400E430
3,68,N  691900N670C680 000 2010,N  819900J300C313
3,69,N  819900T300L580 000 2000,NB 049700M600G500
3,70,NB 049700M600H300 000 1966,NC 099700A100G730 000 2011
3,71,NC 099700A100G730 000 2002,ND 023700M240R400 000 1970V2
3,72,ND 023700F260W370 000 1973,ND 053000M400 000 000 V1
3,73,ND 053000N400 000 000 V2,ND 061100M2313
3,74,ND 061100M41969,ND 165000E471
3,75,ND 171100A430F300,NK 108760C300E800 000 1982
3,76,NK 108760N500B680 000 1973,NK 211000C5985
3,77,NK 211000C680 000 000 1986,NK 363400A200I8513 000 2011
3,78,NK 363400A200K800,NX 045650E600P370 000 2013
3,79,NX 045680E600S650 000 2001,NX 082000E850F3313 000 2001
```

2. Completed Regular Expression to capture Call Number Properties (PHP, Python, etc)

```
\s*
([A-Z]{1,3})  # alpha
\s*
(?:        # optional numbers with optional decimal point
  (\d+)
  (?:\s*?\.\s*?(\d+))?
)?
\s*
(?:            # optional cutter
  \.? \s*
  ([A-Z])      # cutter letter
  \s*
  (\d+ | \Z)       # cutter numbers
)?
\s*
(?:              # optional cutter 1
  \.? \s*
  ([A-Z])      # cutter letter 1
  \s*
  (\d+ | \Z)       # cutter numbers
)?
\s*
(?:              # optional cutter 2
  \.? \s*
  ([A-Z])      # cutter letter 2
  \s*
  (\d+ | \Z)       # cutter numbers
)?
(\s+.+?)?       # everything else (ver, months, parts, etc.)
\s*$
```