Optimizing Help Desk Size

John Sauter

December 30, 2018

Abstract

Any large organization that uses personal computers needs a help desk to help users deal with computer problems. There is an optimum staffing level for the help desk which maximizes productivity. If the help desk is too small, users waste time waiting for a response when they have a problem. If it is too large, help desk operators spend too much time waiting for a call.

The correct size of the help desk can be defined in business terms. It is the size that maximizes net revenue, all other variables being held steady. This article describes a method for determining the optimim size of the help desk.

Contents

1 Using a Simulation for Prediction

To determine the optimum size of the help desk we will run a simulation of the organization's help desk calls. When the simulation accurately predicts the current conditions, we will vary the size of the help desk to see where the point of maximum net revenue falls.

The simulator needs to know a lot about the organization's use of the help desk. Much of this information can be determined from the call logs kept by the help desk. Other information can be estimated from the organization's financial records, but some can be determined only by watching the behavior of users.

1.1 Simulator Inputs

The simulator needs the following information about the organization's help desk and the way it is used.

user population The number of people who are permitted to call the help desk.

help request average The average time between calls to the help desk by any one user. You should be able to determine this by examining the help desk logs. Take the total number of calls to the help desk in a representative week, divide by the user population, and divide that into 2400, the number of minutes in a week.

user earnings The amount earned for the organization by a user who encounters no problems over the course of a year. This is a difficult value to estimate. It will be larger than the net revenue divided by the number of users, because the users will actually encounter problems. You will probably have to guess at this value, then adjust it to make the simulation results match reality.

help operator cost The fully-loaded cost of a help desk operator, including salary, benefits, and training. You should be able to get a good value for this number from the organization's financial and human resources departments.

problem solution time mean The average time required to solve a user's problem and convey that solution to the user. This value can be extracted from the help desk's problem log.

problem solution time std The steandard deviation of the problem solution time. This value can also be extracted from the help desk's problem log.

discourage min The minimum problem solution time which causes the user, and others in his office, to become discouraged due to the performance of the help desk. When solution times are longer than this, users will waste time trying to solve the problem themselves before calling for help, and, if they are very discouraged, will waste the time of other users in their office in an attempt to avoid waiting for a response from the help desk. This value can be determined by watching user behavior.

discourage factor The average factor by which the user, and others whom he impacts, have their productivity impacted by long solution times. You will probably have to guess at this value, then adjust it to make the simulation results match reality.

help operator clearance time The time needed for the help desk operator to finish logging a call and become available to take the next call. This value won't be in the help desk call logs, so it should be determined by observing the help desk in operation.

leave message time The time needed for a user to leave a callback message at the call center, when all help desk operators are busy. You can determine this value by watching an office full of users as they interact with the help desk.

connect time The time needed to connect to a person, once the decision has been made to place a call. This includes looking up the telephone number, dialing, and waiting for the recipient to pick up. You will need to watch both users and help desk operators to estimate this value.

1.2 Running the Simulator

After you have gathered (or estimated) the above values, enter them into the perl script `help_desk_simulator` and run the script, specifying the current size of the help desk and `--verbose` to get backup details for the calculation of net revenue. Adjust the parameters and algorithms as necessary so that the simulation matches reality. Then run the simulator with larger and smaller sizes of the help desk. to determine if increasing or decreasing its size will help or hurt the bottom line.

As the organization grows, the size of the help desk will need to grow with it. Industry norms call for help desk sizes around 1 or 2 percent of the user size, but this will vary with the nature of the organization. Whenever there is a significant change in the organization this simulation should be run again to see if the size of the help desk should be adjusted.

2 The Code

Here is the perl code for the simulator, in case you can't get it from my web site at https://www.systemeyescomputerstore.com. The per-organization variables are at the front. Feel free to modify the algorithms if the organization doesn't fit well into the structure. For example, you might have several populations of users with different help desk call statistics.

```
#!/usr/bin/perl
use 5.010;

use Math::Random;
use Getopt::Long;
use Pod::Usage;
use Data::Dumper;
use warnings;

# You are an IT consultant asked to recommend a staffing level
# for the help desk.  There are complaints that the wait time
# for getting help is too long.

# In order to make a credible recommendation you need to back
# it up with data, and present that data well.  This program
# provides the data you need.
```

```
# The problem can be stated in economic terms: what help desk
# staffing level maximizes productivity.  To calculate this
# we need some information about the company.

my $user_population = 3524;

# The number of computer users in the company.  These are the
# users who might call on the help desk for service.

#my $help_request_average = 200;
my $help_request_average = 1 * 40 * 60;

# The average interval in minutes between calls to the help
# desk by any one user.  You should be able to determine this
# by examining the help desk logs.  Take the total number of
# calls to the help desk in a representative week, divide
# by the user population, and divide that into 2400, the
# number of minutes in a week.

my $help_request_standard_deviation = 60;

# The standard deviation of the interval between calls to
# the help desk by any one user.  To compute this value,
# use the help center log, which should record the arrival
# time for each call.  You have already computed the average
# interval between calls above.  Now compute, for each call,
# the difference between its arrival time and the average
# arrival time.  Square the differences, add them up,
# divide by the number of calls, and take the square root.

my $user_earnings = 348330000 / 3524;

# The amount earned for the company by a user who encounters
# no problems over the course of a year.  This can be approximated
# by dividing the gross annual earnings of the company by
# its employee count.  However, that amount will need to be
# increased because users do have to call the help desk.
# Actually, the gross annual earnings divided by the number
# of users should be an output from this program, to verify
# that the inputs and metholody are accurate.

my $help_operator_cost = 100000;

# The fully-loaded cost of a help desk operator, including
# salary, benefits, and training.
```

```
my $problem_solution_time_mean = 20;
my $problem_solution_time_std = 5;

# The mean and standard deviation of the number of minutes needed
# to solve a problem, once the user is in contact with the help
# desk operator.
# This information should be available from the help desk log.

my $discourage_min = 30;

# The minimum problem solution time which would cause the user,
# and others in his office, to become discourage by the
# performance of the help desk.  When solution times are
# longer than this, users will waste time trying to solve
# the problem themselves before calling for help, and will
# ask other users for help, wasting their time also.

my $discourage_factor = 0.1;

# The average factor by which the user, and others whom he
# impacts, have their productivity impacted by long solution
# times.

my $help_operator_clearance_time = 1;

# The number of minutes needed for the help desk operator to
# finish logging a call and become available to take the next
# call, after he has solved the user's problem.

my $leave_message_time = 1;

# The number of minutes required to leave a callback message
# at the call center.

my $connect_time = 1;

# The number of minutes needed for the user to connect to the
# help desk operator once having placed the call.  Also the
# number of minutes needed for the callback to connect.  This
# includes dialing time and waiting for the recipient of the
# call to pick up.

my $year_length = 50 * 5 * 8 * 60;

# The number of minutes in a work year: 50 weeks per year times
# 5 days per week times 8 hours per day times 60 minutes per hour.
```

```
# Given the parameters above, we compute the company's net revenue.
# Net revenue is the amount earned by the users, minus the amount
# paid for the help desk operators.  We use the above parameters to
# simulate a series of days, to avoid any startup distortion, then we
# simulate 50 weeks, each containing 5 8-hour days.  This gives us
# the net revenue for a year.

# A summary of the simulation is written to a file, for presentation,
# and the net revenue is written to system output, for use by a
# procedure which tries various numbers of help desk operators to find
# the value which maximizes net revenue.

my $verbose = 0;
#
# larger numbers provide more detailed simulation trace.
#

my $man = 0;
my $help = 0;
#
# Parse options.
#
my $getopt_result = GetOptions (
  "help|?" => \$help, "man" => \$man,
  "verbose+" => \$verbose) or pod2usage(2);
pod2usage(1) if $help;
pod2usage(-exitstatus => 0, -verbose => 2) if $man;

#
# Accept parameters.
#
my $help_operator_count = shift (@ARGV);
if (!defined($help_operator_count) or $help_operator_count eq "") {
  die "first argument must be the number of operators at the help desk.\n"
}

#
# Create the users.  Each user is a hash, and there is an array of all
# users.
#
if ($verbose > 1) {
  printf "Creating users.\n";
}

my @user_list;
```

```
for (my $user_number = 1; $user_number <= $user_population; $user_number++) {
  my $user_record = {
    number => $user_number,
    has_a_problem => 0,
    gone_home => 0,
    waiting_until => 0,
    waiting_action => 0,
  };
  $user_list[$user_record->{number}] = $user_record;
  if ($verbose > 3) {
    printf "Created user %d.\n", $user_number;
  }
}

#
# Create the help desk operators.  Each operator is a hash, and there is
# a hash of all help desk operators.
#
if ($verbose > 1) {
  printf "Creating help desk operators.\n";
}
my @help_operator_list;
for (my $operator_number = 1; $operator_number <= $help_operator_count;
  $operator_number++) {
  my $operator_record = {
    number => $operator_number,
    waiting_for_call => 1,
    waiting_until => 0,
    waiting_action => "",
  };
  $help_operator_list[$operator_record->{number}] = $operator_record;
  if ($verbose > 3) {
    printf "Created help desk operator %d.\n", $operator_number;
  }
}

#
# The call queue is implemented as an array.  User numbers are
# shifted in and out, making it a queue.
#
my @callback_queue;

#
# When a help desk operator is waiting for a call, he is placed in a queue
# to make finding him quicker when a call comes in.
#
```

```
my @waiting_list;
for (1 .. $help_operator_count) {
  my $operator_number = $_;
  push @waiting_list, $operator_number;
}

#
# The log is a list of hashes.
#
my @log_list;

#
# For efficiency, we have an array of stacks which record
# which users and operators are scheduled for activity,
# indexed by minute.
#
my $user_schedule = 0;
my $operator_schedule = 0;

#
# make the schedule empty.
#
for (0 .. $year_length) {
  $user_schedule[$_] = [];
  $operator_schedule[$_] = [];
}

#
# The current time, in minutes since the start of the year.
#
my $time_now = 0;

#
# Schedule a problem to call in for each user.
#
for (1 .. $user_population) {
  my $user_number = $_;
  my $user_record = $user_list[$user_number];
  my $wait_time = int(random_normal (1, $help_request_average,
    $help_request_standard_deviation));
  $wait_time = 1 if ($wait_time < 1);
  if ($verbose > 3) {
    printf "Wait time for user %d is %d.\n", $user_number, $wait_time;
  }
  if (($wait_time + $time_now) < $year_length) {
    $user_record->{waiting_until} = $wait_time + $time_now;
```

```
    $user_record->{waiting_action} = "calling";
    push (@{$user_schedule[$wait_time + $time_now]}, $user_number);
    if ($verbose > 2) {
      printf "User %d will have a problem at %d.\n",
        $user_number, $time_now + $wait_time;
    }
  }
}

#
# We advance the clock through the day, taking appropriate actions.
#
if ($verbose > 1) {
  printf "Start of day.\n";
  if ($verbose > 3) {
    for (1 .. $user_population) {
    my $user_number = $_;
    my $user_record = $user_list[$user_number];
      printf "User %d: %s at %d.\n", $user_number, $user_record->{waiting_action},
        $user_record->{waiting_until};
    }
  }
}
for ($time_now = 1; $time_now <= $year_length; $time_now++) {
  if ($verbose > 3) {
    printf "Time: %d minutes.\n", $time_now;
  }
  while (my $user_number = pop @{$user_schedule[$time_now]}) {
    $user_record = $user_list[$user_number];
    if ($time_now != $user_record->{waiting_until}) {
      die "schedule problem.\n";
    } else {
      if ($verbose > 3) {
        printf "%d: user %d finished waiting.\n", $time_now, $user_number;
        if ($verbose > 3) {
          print Dumper($user_record);
        }
      }
      given ($user_record->{waiting_action} ) {
        when("connecting") {
          if ($verbose > 2) {
            printf "%d: user %d connecting.\n", $time_now, $user_number;
          }
          my $operator_number = shift (@waiting_list);
          if (defined($operator_number)) {
            if ($verbose > 1) {
```

```
        printf "%d: user %d gets immediate attention from operator %d.\n",
          $time_now, $user_number, $operator_number;
      }
      my $operator_record = $help_operator_list[$operator_number];
      if ($operator_record->{waiting_for_call} == 0) {
        die "Problem with waiting list.\n";
      }
      $operator_record->{waiting_for_call} = 0;
      $operator_record->{current_user} = $user_number;
      my $problem_solution_time = int(random_normal (1,
        $problem_solution_time_mean, $problem_solution_time_std));
      $problem_solution_time = 1 if ($problem_solution_time < 1);
      $operator_record->{waiting_until} = $time_now + $problem_solution_time;
      $operator_record->{waiting_action} = "completing_call";
      push (@{$operator_schedule[$time_now + $problem_solution_time]},
        $operator_number);
      $user_record->{waiting_until} = 0;
      $user_record->{waiting_action} = "";
    } else {
      # All operators are busy, leave a callback message.
      if ($verbose > 1) {
        printf "%d: user %d will leave a callback request.\n",
          $time_now, $user_number;
      }
      $user_record->{waiting_until} = $time_now + $leave_message_time;
      $user_record->{waiting_action} = "leave_message";
      push (@{$user_schedule[$time_now + $leave_message_time]},
        $user_number);
    }
  }
  when("leave_message") {
    if ($verbose > 2) {
      printf "%d: user %d leaves a callback request.\n",
        $time_now, $user_number;
    }
    push (@callback_queue, $user_number);
  }
  when("calling") {
    if ($verbose > 1) {
      printf "%d: user %d calls for help.\n", $time_now, $user_number;
    }
    $user_record->{has_a_problem} = 1;
    $user_record->{problem_started_time} = $time_now;
    $user_record->{waiting_until} = $time_now + $connect_time;
    $user_record->{waiting_action} = "connecting";
    push (@{$user_schedule[$time_now + $connect_time]}, $user_number);
```

```
      }
    }
  }
}
while (my $operator_number = pop @{$operator_schedule[$time_now]}) {
  my $operator_record = $help_operator_list[$operator_number];
  if ($operator_record->{waiting_until} != $time_now) {
    printf "%d: %d.\n", $time_now, $operator_record->{waiting_until};
    die "problem with operator schedule.\n";
  } else {
    if ($verbose > 3) {
      printf "%d: operator %d finished waiting.\n", $time_now, $operator_number;
      if ($verbose > 3) {
        print Dumper($operator_record);
      }
    }
    given ($operator_record->{waiting_action}) {
      when("starting_call") {
        my $user_number = $operator_record->{current_user};
        my $user_record = $user_list[$user_number];
        if ($verbose > 1) {
          printf "%d: user %d gets a call-back from operator %d.\n",
            $time_now, $user_number, $operator_number;
        }
        my $problem_solution_time = int(random_normal (1,
          $problem_solution_time_mean, $problem_solution_time_std));
        $problem_solution_time = 1 if ($problem_solution_time < 1);
        $operator_record->{waiting_until} = $time_now + $problem_solution_time;
        $operator_record->{waiting_action} = "completing_call";
        push (@{$operator_schedule[$time_now + $problem_solution_time]},
          $operator_number);
      }
      when("completing_call") {
        my $user_number = $operator_record->{current_user};
        my $user_record = $user_list[$user_number];
        $user_record->{has_a_problem} = 0;
        push (@log_list, {
          user_number => $user_number,
          help_operator_number => $operator_number,
          elapsed_time => $time_now - $user_record->{problem_started_time},
        });
        $user_record->{waiting_until} = 0;
        $user_record->{waiting_action} = "";
        if ($verbose > 1) {
          printf "%d: user %d has his problem solved.\n",
            $time_now, $user_number;
```

```
          }
          # See if this user will have another problem
          # before the simulation completes.
          my $wait_time = int(random_normal (1, $help_request_average,
            $help_request_standard_deviation));
          $wait_time = 1 if ($wait_time < 1);
          if ($time_now + $wait_time < $year_length) {
            $user_record->{waiting_until} = $time_now + $wait_time;
            $user_record->{waiting_action} = "calling";
            push (@{$user_schedule[$wait_time + $time_now]}, $user_number);
          }
          $operator_record->{waiting_until} = $time_now +
            $help_operator_clearance_time;
          $operator_record->{waiting_action} = "finished_call";
          push (@{$operator_schedule[$time_now + $help_operator_clearance_time]},
            $operator_number);
        }
        when("finished_call") {
          if ($verbose > 2) {
            printf "%d: help desk operator %d is ready to take another call.\n",
              $time_now, $operator_number;
          }
          $user_number = shift (@callback_queue);
          if (defined($user_number)) {
            if ($verbose > 2) {
              printf "%d: help desk operator %d takes a message from user %d.\n",
                $time_now, $operator_number, $user_number;
            }
            $operator_record->{current_user} = $user_number;
            $operator_record->{waiting_for_call} = 0;
            $operator_record->{waiting_until} = $time_now + $connect_time;
            $operator_record->{waiting_action} = "starting_call";
            push (@{$operator_schedule[$time_now + $connect_time]},
              $operator_number);
          } else {
            # no message in the queue--wait for a call.
            $operator_record->{waiting_for_call} = 1;
            $operator_record->{waiting_until} = 0;
            $operator_record->{waiting_action} = "";
            push @waiting_list, $operator_number;
          }
        }
      }
    }
  }
  # See if there are any waiting help desk operators who can be matched up with
```

```
  # queued calls.
  my $all_done = 0;
  while ($all_done == 0) {
    my $operator_number = shift @waiting_list;
    if (defined($operator_number)) {
      my $user_number = shift @callback_queue;
      if (defined($user_number)) {
        my $operator_record = $help_operator_list[$operator_number];
        if ($verbose > 2) {
          printf "%d: help desk operator %d takes a message from user %d.\n",
            $time_now, $operator_number, $user_number;
        }
        $operator_record->{current_user} = $user_number;
        $operator_record->{waiting_for_call} = 0;
        $operator_record->{waiting_until} = $time_now + $connect_time;
        $operator_record->{waiting_action} = "starting_call";
        push (@{$operator_schedule[$time_now + $connect_time]},
          $operator_number);
      } else {
        # We have operators waiting but the queue is empty.
        unshift @waiting_list, $operator_number;
        $all_done = 1;
      }
    } else {
      # no operators waiting.
      $all_done = 1;
    }
  }
}

# End of the simulation.  Note any lost time due to unsolved problems.

for (1 .. $user_population) {
  my $user_number = $_;
  my $user_record = $user_list[$user_number];
  if ($user_record->{has_a_problem} == 1) {
    push (@log_list, {
      user_number => $user_number,
      help_operator_number => 0,
      elapsed_time => $year_length - $user_record->{problem_started_time},
    });
  }
}

#
# We are finished with the simulation.  Now go through the log and
```

```
# summarize the results.  Also compute the discouragement penalty.
#
my $discourage_amount = 0;
my $sum_time = 0;
my $problem_count = 0;
my $max_solution_time = 0;
my $min_solution_time = 0;

foreach my $log_record (@log_list) {
  my $elapsed_time = $log_record->{elapsed_time};
  $problem_count = $problem_count + 1;
  if ($verbose > 1) {
    printf "Problem %d from user %d solved in %d minutes.\n",
      $problem_count, $log_record->{user_number}, $elapsed_time;
  }
  $sum_time = $sum_time + $elapsed_time;
  if ($max_solution_time < $elapsed_time) {
    $max_solution_time = $elapsed_time;
  }
  if (($min_solution_time == 0) or ($min_solution_time > $elapsed_time)) {
    $min_solution_time = $elapsed_time;
  }
  if ($elapsed_time >= $discourage_min) {
    $discourage_amount = $discourage_amount + ($elapsed_time * $discourage_factor);
  }
}

my $max_productivity = ($user_population * $year_length) -
  ($problem_count * $problem_solution_time_mean);
my $actual_productivity = ($user_population * $year_length) -
  $sum_time - $discourage_amount;
if ($actual_productivity < 0) {
  $actual_productivity = 0;
}
my $percentage = 100.0*($actual_productivity / $max_productivity);
my $gross_earnings = ($user_earnings / $year_length) * $actual_productivity;
my $net_revenue = $gross_earnings - ($help_operator_cost * $help_operator_count);

if ($verbose > 0) {
  printf "# Average solution time for %d problems is %d minutes.\n",
    $problem_count, $sum_time / $problem_count;
  printf "# Max time for one problem is %d minutes, minimum is %d minutes.\n",
    $max_solution_time, $min_solution_time;
  printf "# Discouragement of users due to long solution times is %d.\n",
    $discourage_amount;
  printf "# Number of users is %d.\n", $user_population;
```

```
  printf "# Each user is capable of increasing revenue by \$%d per year," ,
    $user_earnings;
  printf " which is \$%.2f per minute.\n", ($user_earnings / $year_length);
  printf "# Maximum productivity with no problems is %d.\n",
    $user_population * $year_length;
  printf "# Maximum productivity considering there were %d problems is %d.\n",
    $problem_count, $max_productivity;
  printf "# Actual productivity is %d, or %.1f%% of maximum.\n",
    $actual_productivity, $percentage;
  printf "# Gross revenue due to users is %d.\n", $gross_earnings;
  printf "# Cost of help desk operators is %d.\n",
    $help_operator_cost * $help_operator_count;
  printf "# Net revenue is %d.\n", $net_revenue;
}
printf "%d %.2f\n", $help_operator_count, $net_revenue;


__END__

=head1 NAME

help_desk_simulator - simulate the oprations of a help desk

=head1 SYNOPSIS

help_desk_simulator [options] help_desk_operators

The parameter is the number of operators at the help desk.

Options:
  --help            brief help message
  --man             full documentation
  --verbose         output progress information--can be repeated for more detail

=head1 OPTIONS

=over 8

=item B<help>

Print a brief help message and exit.

=item B<man>

Print the manual page and exit.
```

```
=item B<verbose>

Print infromative progress and diagnostic messages to standard output.
The option can be repeated for additional output.  High levels of verbosity
are intended for debugging the simulator, and are somewhat cryptic.

=back

=head1 DESCRIPTION

Suppose you are an IT consultant asked to recommend a staffing level
for an organization's help desk.  There have been complaints that the
wait time for getting help is too long.  The help desk is called "helpless".

In order to make a credible recommendation you need to back it up
with data, and present that data well.  This program provides the data
that you need.

The problem can be stated in economic terms: what help desk staffing level
maximizes productivity?  To calculate this we need some data about the
organization.  Where data is not available we make some reasonable
assumptions.  To validate the assumptions we simulate the current number
of help desk operators, and verify that the simulation matches the current
reality.  Having achieved credibility for the simulation, you can then
simulate other numbers of help desk operators until you find the optimum.

=cut
```

Here is a simple BASH script for running the simulator over a range of help desk sizes. The output can be plotted with gnuplot.

```
#!/bin/bash
rm data_*.txt
sleep_time=1
for ((n=1;n<=400;n++))
do
  nice ./help_desk_simulator --verbose $n >data_${n}.txt &
  if [[ $sleep_time -lt 1 ]] ; then sleep_time=1
  fi
  echo "N = $n, sleep_time = $sleep_time"
  nice sleep $sleep_time
  load_average=`uptime | awk -F'load average:'  '{print $2}' |\
    awk -F',' '{print $1}' | awk -F'.' '{print $1}'`
  sleep_time=$(($load_average / 12))
  while [[ "$load_average" -gt 100 ]] ; do
    echo "Load average = $load_average"
```

```
    nice sleep 1
    load_average=‘uptime | awk -F’load average:’  ’{print $2}’ |\
      awk -F’,’ ’{print $1}’ | awk -F’.’ ’{print $1}’‘
  done
done
wait
cat data_*.txt >data.txt
```