Listening to big data

Or, philately will get you everywhere

Mike Godfrey
Software Architecture Group
University of Waterloo

Overview

- Is clone analysis / empirical SE a Big Data problem?
  - … and should we care?
- Looking hard for the Big Picture
  - And why sometimes that can be a bad idea
- Let's go swimming with the data!
  - Some experiences and some advice

"Big data"

- Three Vs
  - Volume, Velocity, Variety
- Why?
  - Enhanced decision making, insight discovery, and process optimization
- Common problems:
  - Capture, curation, storage, search, sharing, transfer, analysis, and visualization

(More data + simple algorithms) >> (complex algorithms)

- Fantastic talk by Peter Norvig of Google:

  "The unreasonable effectiveness of data"
  http://www.youtube.com/watch?v=yvDCzhbjYWs
- *"Every time I fire a linguist, my scores get better."*
  - [Fred Jelinek, paraphrased]
- But does that work for clone detection / ESE too?
  - Should we all use N-gram algorithms?

Data quality

I SAW THE CODE FOR YOUR COMPUTER PROGRAM YESTERDAY.
IT LOOKED EASY. IT'S JUST A BUNCH OF TYPING. AND HALF OF THE WORDS WERE SPELLED WRONG.
S. Adams E-Mail: SCOTTADAMS@AOL.COM
© 1994 United Feature Syndicate, Inc.
AND DON'T GET ME STARTED ABOUT YOUR OVER-USE OF COLONS.
THEY REMIND ME OF YOU, SIR.

(Big data + simple algorithms)?

- NLP, for example, analyzes unstructured prose
  - Much variation: intent, word ordering, relationships, …
  - NLP often does some pre-processing e.g., stemming
- ESE examines development artifacts with lots of internal structure + external linkage, implicit and explicit
  - Source code text, including comments
  - Version control meta-data
  - Bug reports
  - …
- When you have reliable structure, exploit it!
  - Yes?
  - So maybe big ESE data isn't really big data …

Looking for the Big Picture

Trials and Errors: Why Science is Failing Us

Wired Magazine, December 2011

by Jonah Lehrer

Looking for the Big Picture

A selective attention test

"I used to think that the brain was the most wonderful organ in my body. Then I realized who was telling me this."
— Emo Philips

http://www.youtube.com/watch?v=vJG698U2Mvo

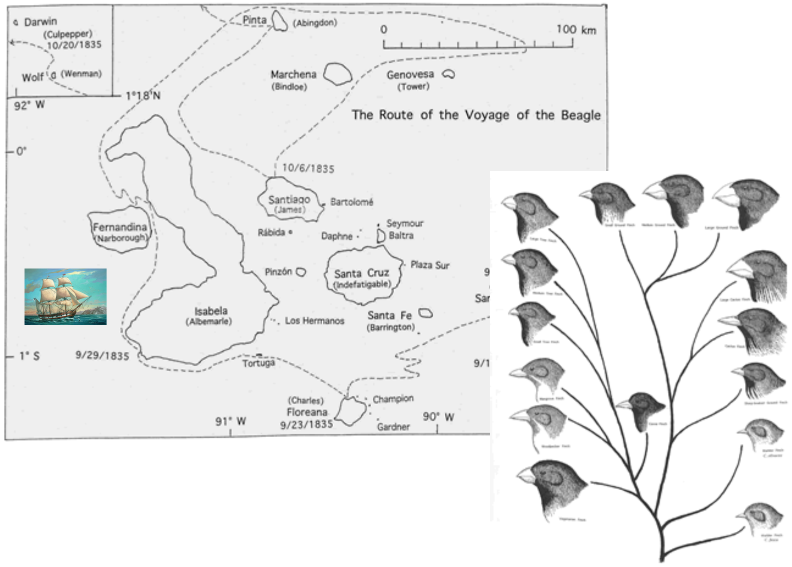
Tim Minchin

http://www.upworthy.com/this-is-the-most-inspiring-yet-depressing-yet-hilarious-yet-horrifying-yet-heartwarming-grad-speech

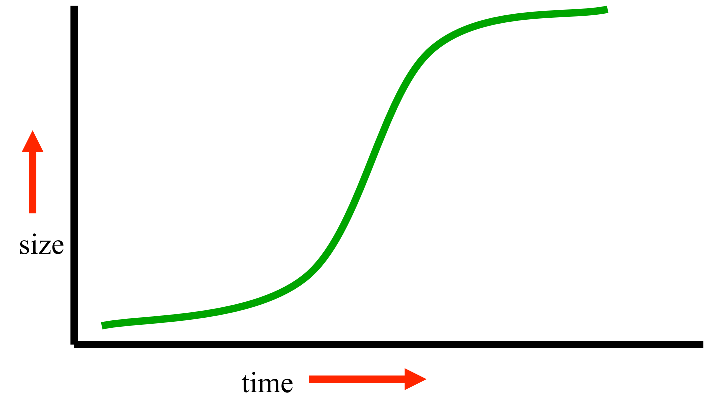
"Physics is the only real science.
The rest are just stamp collecting."

Ernest Rutherford (1871-1937)

Father of atomic physics
Nobel prize for … chemistry

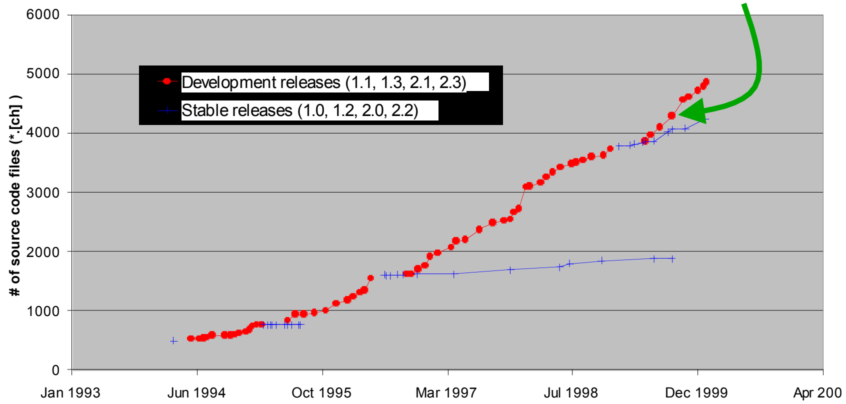


The "S" curve of successful growth

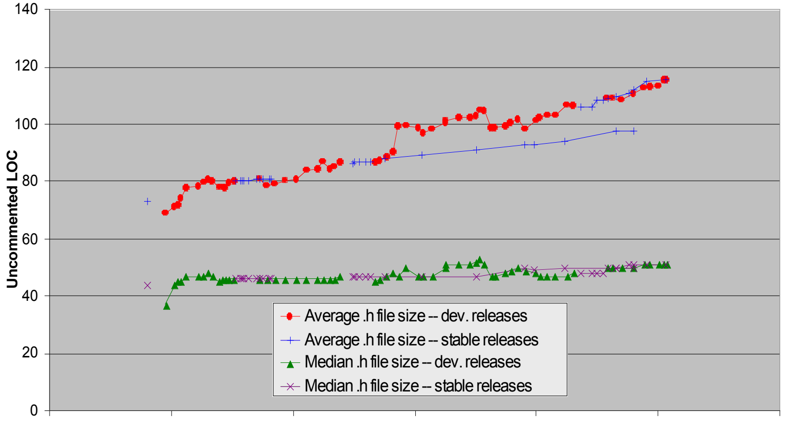


Linux kernel: Growth of kernel src tree (# of files)

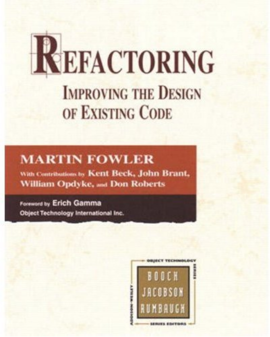
$y = .21*x^2 + 252*x + 90,055 \quad r2=.997$



Linux kernel: Average / median `.h` file size



Source code cloning


REFACTORING
IMPROVING THE DESIGN OF EXISTING CODE
MARTIN FOWLER
With Contributions by Kent Beck, John Brant, William Opdyke, and Don Roberts
Foreword by Erich Gamma
Object Technology International Inc.
BOOCH JACOBSON RUMBAUGH

> *"Number one in the stink parade is duplicated code. If you see the same code structure in more than one place, you can be sure that your program will be better if you find a way to unify them."*
>
> – "Bad Smells"
> [Beck/Fowler in *Refactoring*]

'Cloning considered harmful' considered harmful

1. Forking
   - Hardware variation
     e.g., Linux SCSI drivers
   - Platform variation
   - Experimental variation
2. Templating
   - Boilerplating
   - API / library protocols
   - Generalized programming idioms
   - Parameterized code
3. Post-hoc customizing
   - Bug workarounds
   - Replicate + specialize

Cloning harmfulness:
Two open source case studies

		Apache		Gnumeric	
Group	**Pattern**	**Good**	**Harmful**	**Good**	**Harmful**
Forking	Hardware variation	0	0	0	0
Forking	Platform variation	10	0	0	0
Forking	Experimental variation	4	0	0	0
Templating	Boiler-plating	5	0	6	7
Templating	API	0	0	0	9
Templating	Idioms	0	12	1	1
Templating	Parameterized code	5	12	10	34
Customizing	Replicate + specialize	12	4	15	16
Customizing	Bug workarounds	0	0	0	0
Total		**36**	**28**	**32**	**67**

Apache httpd 2.2.4 - 60 Tokens
Gnumeric 1.6.3 - 60 Tokens

What to do?

- *Swim* with the data
- *Be* the gorilla in the mist
- *Look for lumps* under the carpet & ask "Why?"

Luncheon with the boating party

- While fooling around with RASCAL, I printed N! in the range N=1,…,1000
  - 1000! ≈ 4.02 x 10^{2567}
    … in case you were wondering,
  - A googol (note spelling) is only 10^{100}
    10,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,0
    00,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,0
    00,000
- Here’s what 1000! looks like:

```
402,387,260,077,093,773,543,702,433,923,003,985,719,374,864,210,714,632,543,799,910,429,938,512,
398,629,020,592,044,208,486,969,404,800,479,988,610,197,196,058,631,666,872,994,808,558,901,323,
829,669,944,590,997,424,504,087,073,759,918,823,627,727,188,732,519,779,505,950,995,276,120,874,
975,462,497,043,601,418,278,094,646,496,291,056,393,887,437,886,487,337,119,181,045,825,783,647,
849,977,012,476,632,889,835,955,735,432,513,185,323,958,463,075,557,409,114,262,417,474,349,347,
553,428,646,576,611,667,797,396,668,820,291,207,379,143,853,719,588,249,808,126,867,838,374,559,
731,746,136,085,379,534,524,221,586,593,201,928,090,878,297,308,431,392,844,403,281,231,558,611,
036,976,801,357,304,216,168,747,609,675,871,348,312,025,478,589,320,767,169,132,448,426,236,131,
412,508,780,208,000,261,683,151,027,341,827,977,704,784,635,868,170,164,365,024,153,691,398,281,
264,810,213,092,761,244,896,359,928,705,114,964,975,419,909,342,221,566,832,572,080,821,333,186,
116,811,553,615,836,546,984,046,708,975,602,900,950,537,616,475,847,728,421,889,679,646,244,945,
160,765,353,408,198,901,385,442,487,984,959,953,319,101,723,355,556,602,139,450,399,736,280,750,
137,837,615,307,127,761,926,849,034,352,625,200,015,888,535,147,331,611,702,103,968,175,921,510,
907,788,019,393,178,114,194,545,257,223,865,541,461,062,892,187,960,223,838,971,476,088,506,276,
862,967,146,674,697,562,911,234,082,439,208,160,153,780,889,893,964,518,263,243,671,616,762,179,
168,909,779,911,903,754,031,274,622,289,988,005,195,444,414,282,012,187,361,745,992,642,956,581,
746,628,302,955,570,299,024,324,153,181,617,210,465,832,036,786,906,117,260,158,783,520,751,516,
284,225,540,265,170,483,304,226,143,974,286,933,061,690,897,968,482,590,125,458,327,168,226,458,
066,526,769,958,652,682,272,807,075,781,391,858,178,889,652,208,164,348,344,825,993,266,043,367,
660,176,999,612,831,860,788,386,150,279,465,955,131,156,552,036,093,988,180,612,138,558,600,301,
435,694,527,224,206,344,631,797,460,594,682,573,103,790,084,024,432,438,465,657,245,014,402,821,
885,252,470,935,190,620,929,023,136,493,273,497,565,513,958,720,559,654,228,749,774,011,413,346,
962,715,422,845,862,377,387,538,230,483,865,688,976,461,927,383,814,900,140,767,310,446,640,259,
899,490,222,221,765,904,339,901,886,018,566,526,485,061,799,702,356,193,897,017,860,040,811,889,
729,918,311,021,171,229,845,901,641,921,068,884,387,121,855,646,124,960,798,722,908,519,296,819,
372,388,642,614,839,657,382,291,123,125,024,186,649,353,143,970,137,428,531,926,649,875,337,218,
940,694,281,434,118,520,158,014,123,344,828,015,051,399,694,290,153,483,077,644,569,099,073,152,
433,278,288,269,864,602,789,864,321,139,083,506,217,095,002,597,389,863,554,277,196,742,822,248,
757,586,765,752,344,220,207,573,630,569,498,825,087,968,928,162,753,848,863,396,909,959,826,280,
956,121,450,994,871,701,244,516,461,260,379,029,309,120,889,086,942,028,510,640,182,154,399,457,
156,805,941,872,748,998,094,254,742,173,582,401,063,677,404,595,741,785,160,829,230,135,358,081,
840,096,996,372,524,230,560,855,903,700,624,271,243,416,909,004,153,690,105,933,983,835,777,939,
410,970,027,753,472,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,
000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,
000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,
000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000
```

That's 249 trailing zeros!

And as N grows, they accumulate!

402,387,260,077,093,773,543,702,433,923,003,985,719,374,864,210,714,632,543,799,910,429,938,512,
398,629,020,592,044,208,486,969,404,800,479,988,610,197,196,058,631,666,872,994,808,558,901,323,
829,669,944,590,997,424,504,087,073,759,918,823,627,727,188,732,519,779,505,950,995,276,120,874,
975,462,497,043,601,418,278,094,646,496,291,056,393,887,437,886,487,337,119,181,045,825,783,647,
849,977,012,476,632,889,835,955,735,432,513,185,323,958,463,075,557,409,114,262,417,474,349,347,
553,428,646,576,611,667,797,396,668,820,291,207,379,143,853,719,588,249,808,126,867,838,374,559,
731,746,136,085,379,534,524,221,586,593,201,928,090,878,297,308,431,392,844,403,281,231,558,611,
036,976,801,357,304,216,168,747,609,675,871,348,312,025,478,589,320,767,169,132,448,426,236,131,
412,508,780,208,000,261,683,151,027,341,827,977,704,784,635,868,170,164,365,024,153,691,398,281,
264,810,213,092,761,244,896,359,928,705,114,964,975,419,909,342,221,566,832,572,080,821,333,186,
116,811,553,615,836,546,984,046,708,975,602,900,950,537,616,475,847,728,421,889,679,646,244,945,
160,765,353,408,198,901,385,442,487,984,959,953,319,101,723,355,556,602,139,450,399,736,280,750,
137,837,615,307,127,761,926,849,034,352,625,200,015,888,535,147,331,611,702,103,968,175,921,510,
907,788,019,393,178,114,194,545,257,223,865,541,461,062,892,187,960,223,838,971,476,088,506,276,
862,967,146,674,697,562,911,234,082,439,208,160,153,780,889,893,964,518,263,243,671,616,762,179,
168,909,779,911,903,754,031,274,622,289,988,005,195,444,414,282,012,187,361,745,992,642,956,581,
746,628,302,955,570,299,024,324,153,181,617,210,465,832,036,786,906,117,260,158,783,520,751,516,
284,225,540,265,170,483,304,226,143,974,286,933,061,690,897,968,482,590,125,458,327,168,226,458,
066,526,769,958,652,682,272,
660,176,999,612,831,860,788,
435,694,527,224,206,344,631,
885,252,470,935,190,620,929,
962,715,422,845,862,377,387,
899,490,222,221,765,904,339,901,886,018,566,526,485,061,799,702,356,193,897,017,860,040,811,889,
729,918,311,021,171,229,
372,388,642,614,839,657,
940,694,281,434,118,520,
433,278,288,269,864,602,
757,586,765,752,344,220,
956,121,450,994,871,701,244,516,461,260,379,029,309,120,889,086,942,028,510,640,182,154,399,457,
156,805,941,872,748,998,094,254,742,173,582,401,063,677,404,595,741,785,160,829,230,135,358,081,
840,096,996,372,524,230,560,855,903,700,624,271,243,416,909,004,153,690,105,933,983,835,777,939,
410,970,027,753,472,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,
000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,
000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,
000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000

Helper function #1

```
public int countZeros (int n) {
    if (n < 10) {
        return 0;
    } else if (n % 10 == 0) {
         return 1 + countZeros (n / 10);
    } else {
        return countZeros (n / 10);
    }
}
```

```
rascal> int i = fact(1000);
int: 4023872600770…
rascal> countZeros(i);
int: 472
```

Helper function #1, v1.1

```
public int countTrailingZeros (int n) {
    if (n < 10) {
        return 0;
    } else if (n % 10 == 0) {
         return 1 + countTrailingZeros (n / 10);
    } else {
        return 0 ;
    }
}
```

```
rascal> countTrailingZeros(i);
int: 249
```

Helper function #2 — Let’s play around

```
public void printLastTwenty (int n){
    for(int i <- [n-19..n]) {
       println ("<i>! has <countTrailingZeros(fact(i))> ”
           + ”trailing zeros.");
    }
}
```

```
rascal>printLastTwenty(1000);
981! has 243 trailing zeros.
982! has 243 trailing zeros.
983! has 243 trailing zeros.
984! has 243 trailing zeros.
985! has 244 trailing zeros.
986! has 244 trailing zeros.
987! has 244 trailing zeros.
988! has 244 trailing zeros.
989! has 244 trailing zeros.
990! has 245 trailing zeros.
```

```
991! has 245 trailing zeros.
992! has 245 trailing zeros.
993! has 245 trailing zeros.
994! has 245 trailing zeros.
995! has 246 trailing zeros.
996! has 246 trailing zeros.
997! has 246 trailing zeros.
998! has 246 trailing zeros.
999! has 246 trailing zeros.
1000! has 249 trailing zeros.
ok
```

Looking for lumps

```
public void findLumps (int n) {
    int iMinusOneFactZeros = 0;
    for (int i <- [1..n]) {
        int iFactZeros = countTrailingZeros(fact(i));
        int diff = iFactZeros - iMinusOneFactZeros ;
        if (diff >= 1) {
            println ("<diff> more zeros at <i>!");
        }
        iMinusOneFactZeros = iFactZeros;
    }
}
```

```
rascal>findLumps(1000);
1 more zeros at 5!
1 more zeros at 10!
1 more zeros at 15!
1 more zeros at 20!
2 more zeros at 25!
1 more zeros at 30!
1 more zeros at 35!
```

```
1 more zeros at 40!
1 more zeros at 45!
2 more zeros at 50!
1 more zeros at 55!
1 more zeros at 60!
1 more zeros at 65!
1 more zeros at 70!
2 more zeros at 75!
```

Looking for lumps

```
1 more zeros at 80!
1 more zeros at 85!
1 more zeros at 90!
1 more zeros at 95!
2 more zeros at 100!
1 more zeros at 105!
1 more zeros at 110!
1 more zeros at 115!
1 more zeros at 120!
3 more zeros at 125!
1 more zeros at 130!
…
1 more zeros at 245!
3 more zeros at 250!
1 more zeros at 255!
```

```
1 more zeros at 495!
3 more zeros at 500!
1 more zeros at 505!
…
1 more zeros at 620!
4 more zeros at 625!
1 more zeros at 630!
…
1 more zeros at 985!
1 more zeros at 990!
1 more zeros at 995!
3 more zeros at 1000!
ok
```

Helper function #3, v1.1

```
public void findLumps2 (int n, int tao) {
    int iMinusOneFactZeros = 0;
    for (int i <- [1..n]) {
        int iFactZeros = countTrailingZeros(fact(i));
        int diff = iFactZeros - iMinusOneFactZeros ;
        if (diff >= tao) {
            println ("<diff> more zeros at <i>!");
        }
        iMinusOneFactZeros = iFactZeros;
    }
}
```

- We can parameterize the threshold to look for jumps of 2, 3, or 4 zeros

Looking for lumps

```
rascal>findLumps2(1000,2);
2 more zeros at 25!
2 more zeros at 50!
2 more zeros at 75!
2 more zeros at 100!
3 more zeros at 125!
2 more zeros at 150!
2 more zeros at 175!
2 more zeros at 200!
2 more zeros at 225!
3 more zeros at 250!
2 more zeros at 275!
…
2 more zeros at 950!
2 more zeros at 975!
3 more zeros at 1000!
ok
```

```
rascal>findLumps2(1000,3);
3 more zeros at 125!
3 more zeros at 250!
3 more zeros at 375!
3 more zeros at 500!
4 more zeros at 625!
3 more zeros at 750!
3 more zeros at 875!
3 more zeros at 1000!
ok
```

```
rascal>findLumps2(1000,4);
4 more zeros at 625!
ok
```

$5^0 = 1$

$5^1 = 5$

$5^2 = 25$

$5^3 = 125$

$5^4 = 625$

$5^5 = 3125$

An analytic solution

Let N be a positive integer.

Let $k = floor\ (log_5\ N)$

Start a counter at zero, call it *nz*

We want to examine *i <- [1..N]*

- If *i* is *not* divisible by 5, ignore it
- If *i* is divisible by 5, add 1 to *nz*
- If *i* is also divisible by 25, add 1 more
- …
- If *i* is also divisible by 2^k, add 1 more

Final functions

```
public int predictZeros (int N) {
    int k = floorLogBase(N, 5);
    int nz = 0;
    for (int i <- [1..N] ){
        int p5 = 1;
        for (int j <- [1..k]) {
            p5 *= 5;
            if (i % p5 == 0) {
                nz += 1;
            } else {
                break;
            }
        }
    }
    return nz;
}
```

```
public void verifyTheory (int N) {
    int checkInterval = 100; // for printing
    bool failed = false;
    for (int i <- [1..N]) {
        ifact=fact(i);
        int p = predictZeros(i);
        int c = countTrailingZeros(ifact);
        if (p != c) {
            failed = true;
            println (“Found a counter example at i=<i>");
            break;
        } else {
            if (i % checkInterval == 0) {
                println ("<i>! has <p> trailing zeros");
            }
        }
    }
    if (!failed) {
        println ("The theory works for i: 1..<N>”);
    }
}
```

Time to celebrate!

```
rascal>verifyTheory(10);
The theory works for i: 1..10
ok
rascal>verifyTheory(100);
100! has 24 trailing zeros
The theory works for i: 1..100
ok
```

```
rascal>verifyTheory(1000);
100! has 24 trailing zeros
200! has 49 trailing zeros
300! has 74 trailing zeros
400! has 99 trailing zeros
500! has 124 trailing zeros
600! has 148 trailing zeros
Found a counter example at i=625
   predicted zeros = 155
   observed zeros  = 156
ok
```

Looking under the hood

```
// I wrote these little wrappers.

// Log for an arbitrary base
public real logB(real a, real base) {
    return log(a) / log(base);
}

public real floor (real a) {
   return toReal(round (a - 0.5));
}

public int floorLogBase (int a, int b) {
    return toInt(floor(logB(toReal(a), toReal(b))));
}
```

```
rascal>floorLogBase(625,5);
int: 3
rascal>logB(625.0,5.0);
real: 3.9999999999999998757330130880776320985295476764801684…….
```

A bad fix (that kinda works)

```
// I wrote these little wrappers.

// Log for an arbitrary base
public real logB(real a, real base) {
    return log(a) / log(base);
}

public real floor (real a) {
   return toReal(round (a - 0.5 + 0.00001));
}

public int floorLogBase (int a, int b) {
    return toInt(floor(logB(toReal(a), toReal(b))));
}
```

A better, exact solution

```
// Also change predictZeros to call this version
public int floorLogBase2 (int a, int b) {
   int remaining = a;
   int ans = 0;
   while (remaining >= b) {
      ans += 1;
      remaining /= b;
   }
   return ans;
}
```

```
rascal>verifyTheory(1000);
100! has 24 trailing zeros
200! has 49 trailing zeros
300! has 74 trailing zeros
400! has 99 trailing zeros
500! has 124 trailing zeros
600! has 148 trailing zeros
```

```
700! has 174 trailing zeros
800! has 199 trailing zeros
900! has 224 trailing zeros
1000! has 249 trailing zeros
The theory works for i: 1..1000
ok
```

Lessons?

- Explore the terrain, take notes, build intuition, develop theories, test them
  - Refine, repeat
  - Double check
- Build infrastructure with natural “break points”
  - Understandable >> fast, esp. in the beginning
  - The correct way >> the easy way,
    - The correct way may be pretty easy too
- Document and later challenge your assumptions
  - Are you measuring what you think you are measuring?

What history taught me

- Study what you already have and understand
  - Often, your intuition is golden
  - Take it apart and see how it works (e.g., Linux study)
- Challenge pre-conceived notions
  - Create testable hypotheses + evaluate them (e.g., cloning)
- Software archives contain lots of rich data
  - But need to process, link, mine the artifacts
- Need to continually re-examine reasonableness of assumptions
  - Don't blindly trust the numbers; dig and validate!

Listening to big data

Or, philately will get you everywhere

Mike Godfrey
Software Architecture Group
University of Waterloo