

Using Evolutionary Algorithms to Test Software

Prof. Andrea Arcuri

Kristiania University College



Are software applications doing what
are they supposed to do?

Ariane 5 – ESA



On June 4, 1996, the flight of the Ariane 5 launcher **ended in a failure.**



\$500 millions in cost

Software bug

Fatal Therac-25 Radiation

1986, Texas, person died



Power Shutdown in 2003

Nearly 50 millions persons affected in Canada/US



2010, Toyota, bug in braking system, 200 000 cars recalled



Knight Capital Group 2012

\$460 millions lost in 45 minutes of trading due to bug



And I could go on the whole day...

- As of 2013, estimated that software testing costing **\$312 billions** worldwide
- In 2016, 548 recorded and documented software failures impacted **4.4 billion** people and **\$1.1 trillion** in assets worldwide

But what about every-day life in Oslo???

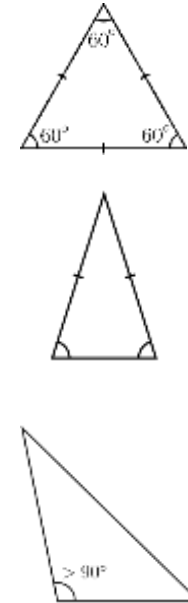
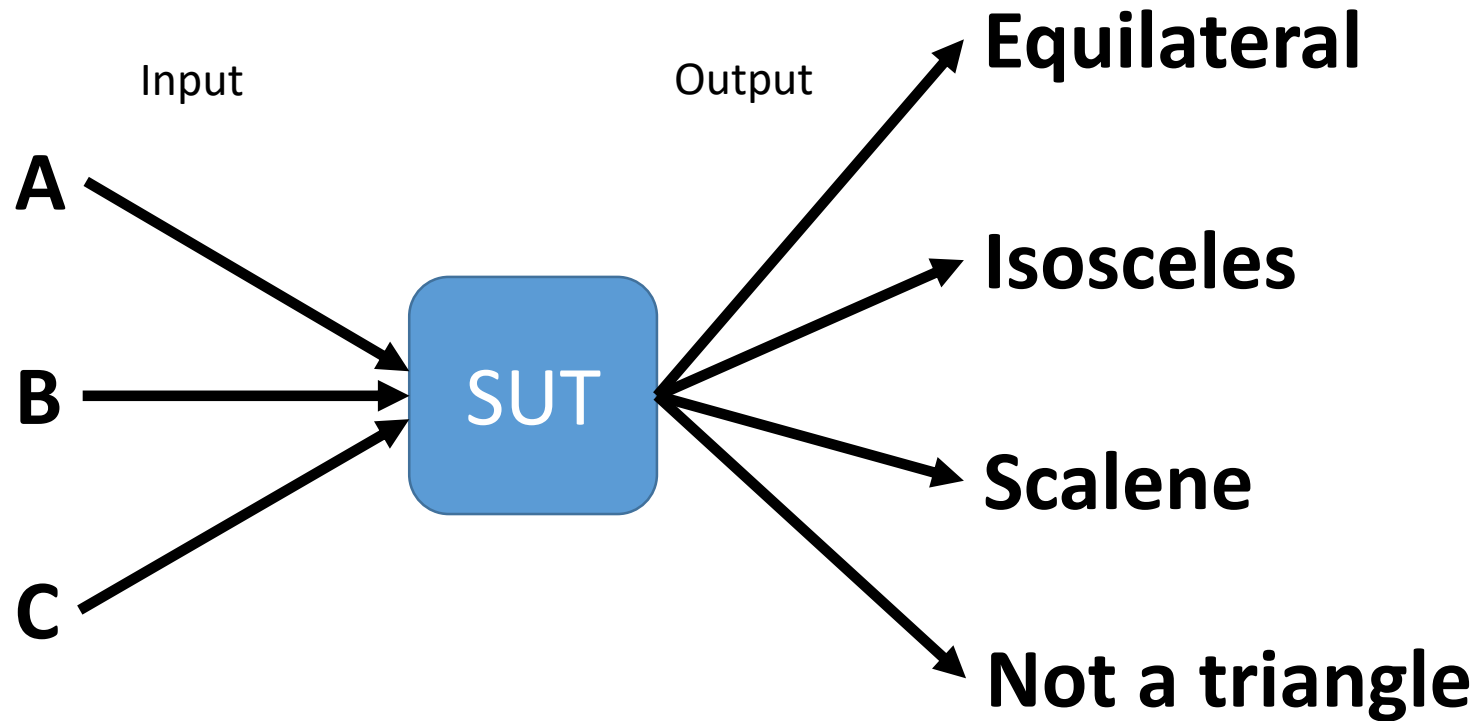


What to do? **Test** the software

But how to test “*properly*”?

Manual testing is expensive, tedious and of limited effect

Example: Triangle Classification (TC)



- 3 integer numbers (A, B and C) as input representing the length of the *edges*
- 4 possible outcomes
- Does the *system under test* (SUT) give the right answer?

How to test TC?

- If numbers are 32 bit integers, there are $2^{32} * 2^{32} * 2^{32} = 2^{96} = 79,228,162,514,264,337,592,626,226,666$ possible combinations
 - ie, 79 Octillion possible combinations of edge lengths
- Cannot test all of them
- Need to define some *test criteria* to decide a good enough test suite which is:
 1. good at finding bugs
 2. small enough to be manageable

1 Test per Output

- **t0:(A=42, B=42, C=42) => EQUILATERAL**
- **t1:(A=42, B=42, C=5) => ISOSCELES**
- **t2:(A=42, B=43, C=44) => SCALENE**
- **t3:(A=42, B=42, C=12345) = NOT A TRIANGLE**
- *Would such 4 test cases be enough?*
- What if the EQUILATERAL case is implemented with just something as naïve as **“if A==B and B==C then EQUILATERAL”**?
 - (A=-3, B=-3, C=-3) would wrongly return EQUILATERAL instead of NOT A TRIANGLE
 - Just checking basic scenarios is not enough

White-Box Testing

- Code can have bugs
- *To trigger a bug, the code must be executed*
- But code can have very complex control flow
- Some rare “paths” in the code might be executed only in very complex scenarios
- *Goal: in a test suite, have each single line and branch be executed at least once*

```
public Classification classify(  
    int a, int b, int c){  
  
    if(a <= 0 || b <= 0 || c <= 0){  
        return Classification.NOT_A_TRIANGLE;  
    }  
  
    if(a == b && b == c){  
        return Classification.EQUILATERAL;  
    }  
  
    int max = Math.max(a, Math.max(b, c));  
  
    if( (max == a && max - b - c >= 0 ) ||  
        (max == b && max - a - c >= 0 ) ||  
        (max == c && max - a - b >= 0 ) ){  
        return Classification.NOT_A_TRIANGLE;  
    }  
  
    if(a == b || b == c || a == c){  
        return Classification.ISOSCELES;  
    } else {  
        return Classification.SCALENE;  
    }  
}
```

Example

- **if((max == a && max -b -c >= 0) ||
 (max == b && max -a -c >= 0) ||
 (max == c && max -a -b >= 0))**
- In this disjunction of 3 different clauses, if in your test suite the first clause is always true, the other 2 would never be executed
 - so if wrong, you would not know
- This is a TRIVIAL example... real industrial software can be way more complex...
- Writing tests for each path is not only tedious, but can be quite hard as well

Automated Test Case Generation

- **Automatically generate test cases**
- Model software testing as an **optimization problem**
 - Maximize code coverage
 - Find bugs
 - Etc.
- Use optimization algorithms
- Benefits: *cheaper and more effective than manual testing*
- *Hard problem to automate*
 - given a non-linear constraint, there is no guaranteed algorithm that can solve it in polynomial time

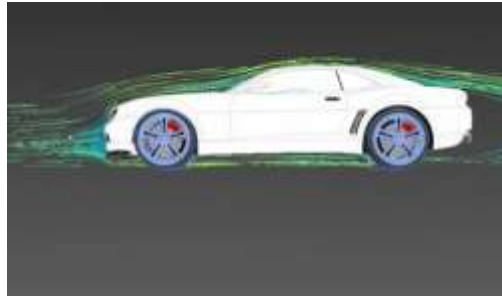
Optimization Problems

- There are a lot of problems in science and engineering for which we do not know any algorithm that can solve them in reasonable (ie *polynomial*) time
 - Such algorithms might exist, but we do not know them yet
- *Brute Force*: try all possible combinations, until find valid solution... but that is *exponential!!!*
 - ie, it could take forever to find any solution
- We need some *heuristics* to address these problems
 - But **no guarantee** that we can find a solution in reasonable time

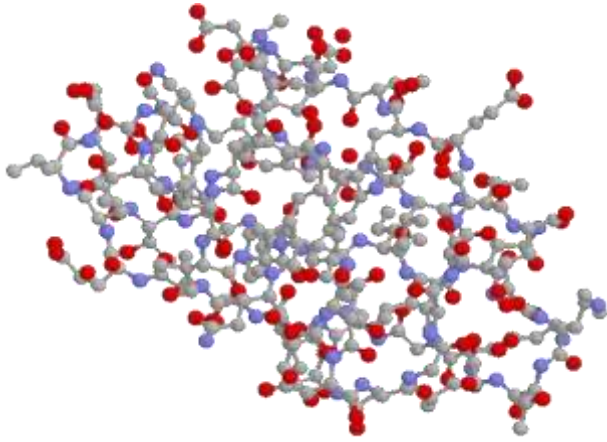
Vehicle Design



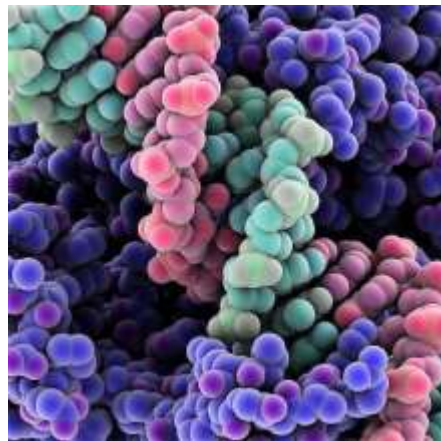
- How to find best shape to reduce air resistance?
- Can have different designs, and then test them in a wind tunnel



Protein Design



How to find the right sequence of amino acids which will result in a protein with some sought properties?



Stock Market



How to find best investment portfolio to maximize profit?



Class Schedule

My Class Schedule | Fall

Start Time 8:00 AM Time Interval: 30

Time	Mon	Tue	Wed
8:00 AM	Breakfast	Breakfast	Breakfast
8:30 AM	Business: Lecture Bldg B, Rm 256	Physics: Lab Bldg J, Rm 309	Business: Lec Bldg B, Rm 2
9:00 AM			
9:30 AM	Applied Math Bldg H, Rm 100		Applied Mat Bldg H, Rm 1
10:00 AM			
10:30 AM			
11:00 AM			

How to find best class schedule for which:

- There is time for all classes
- Classes in same year are not in parallel (ie conflicting)
- Preferences of lectures are taken into account
- Etc.
- ?

TimeEdit * WESTERDALE OSLO AET * TIMEPLAN * TIMEPLAN						
DAG < DKT >	NA - 22.12.2017	ENDRE SØK	Algoritmer og datastrukturer (PG4200-17), Enterpriseprogrammering 2 (PG6100-17)			
TID	EMNE	KLASSE, STUDENTGRUPPE, STUDENT	AKTIVITET, PROJEKT	LÆRER	ROM	
u 40	Ti 03.10.2017					
	13:00 - 15:00	Margrethe Øia Thorsen	Øving		Salerom FU110	
	Fr 06.10.2017					
	09:15 - 12:00	Enterpriseprogrammering 2 (PG6100-17)	Programmering 15	Forelesning	Andreas Arcuri	Undervisningsrom F205
u 41	Ti 10.10.2017					
	08:15 - 12:00	Algoritmer og datastrukturer (PG4200-17)	Intelligente systemer 16, Programmering 16, Spilprogrammering 16		Auditorium VU06	
	Fr 13.10.2017					
	09:15 - 12:00	Enterpriseprogrammering 2 (PG6100-17)	Programmering 15	Forelesning	Andreas Arcuri	Undervisningsrom F205
u 42	Ti 17.10.2017					
	08:15 - 12:00	Algoritmer og datastrukturer (PG4200-17)	Intelligente systemer 16, Programmering 16, Spilprogrammering 16		Auditorium VU06	

RPG Equipment



In RPGs, how to find best combination of wearable items to maximize attack and defense under the constraints of maximum weight and item slots available?

Properties of Optimization Problems

- 2 main components: *Search Space* and *Fitness Function*
- **Goal:** find the best solution from the search space such that the fitness function is minimized/maximized

Search Space

- Set X of all possible solutions for the problem
- If a solution can be represented with 0/1 bit sequence of length N , then search space is all possible bit strings of size N
- Search space is usually huge, eg 2^N
 - Otherwise use brute force, and so would not be a problem

Fitness Function

- $f(x)=h$
- Given a solution x in X , calculate an heuristic h that specifies how good the solution is
- Problem dependent, to minimize or maximize:
 - Minimize air resistance
 - Maximize protein structure properties
 - Maximize Return Of Investment
 - etc.

Optimization Algorithms

- Algorithm that explores the search space X
- Only a tiny sample of X can be evaluated
- Use fitness $f(x)$ to guide the exploration to fitter areas of the search space with better solutions
- Stopping criterion: after evaluating K solutions (or K amount of time is passed), return best x among the evaluated solutions
- Many different kinds of optimization algorithms...
 - But as a user, still need to provide the representation and $f(x)$

Search Operator

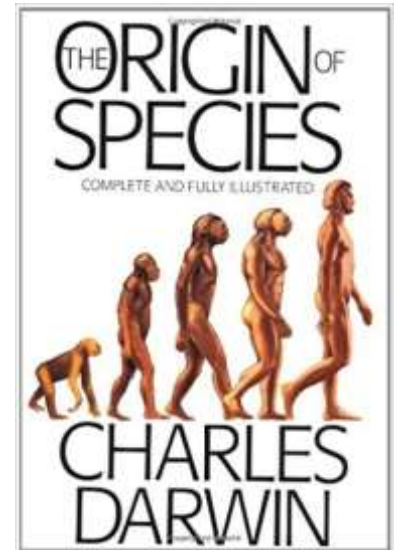
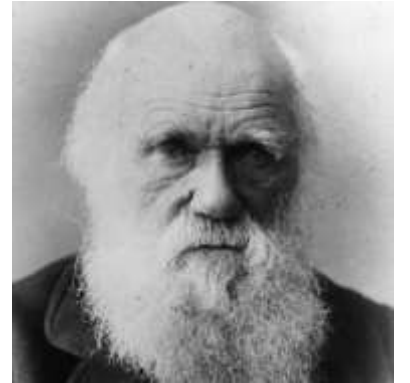
- $s(x) = x'$
- An operator that, from a solution x , gives a new one x'
- Still need to evaluate its fitness, ie $f(x')$
- The optimization algorithm will use the search operators to choose which new x' in X to evaluate
- The search operator will depend on the problem representation
- Example: flip a bit in a bit-sequence representation

Nature Inspired Algorithms

- Nature is good at solving many different problems
- Idea: get inspiration from natural phenomena to create effective optimization algorithms
- E.g., *carbon-to-diamond process*: high temperature in Earth's mantle, cooled slowly while raising up to surface
 - *Simulated Annealing Algorithm*
- E.g., behavior of *ants seeking a path* between their colony and a source of food, based on pheromone trails
 - *Ant Colony Optimization Algorithm*

Theory Evolution

- Charles Darwin, "*The Origin of Species*", 1859
- Theory describing how different species *evolved* from unicellular organisms

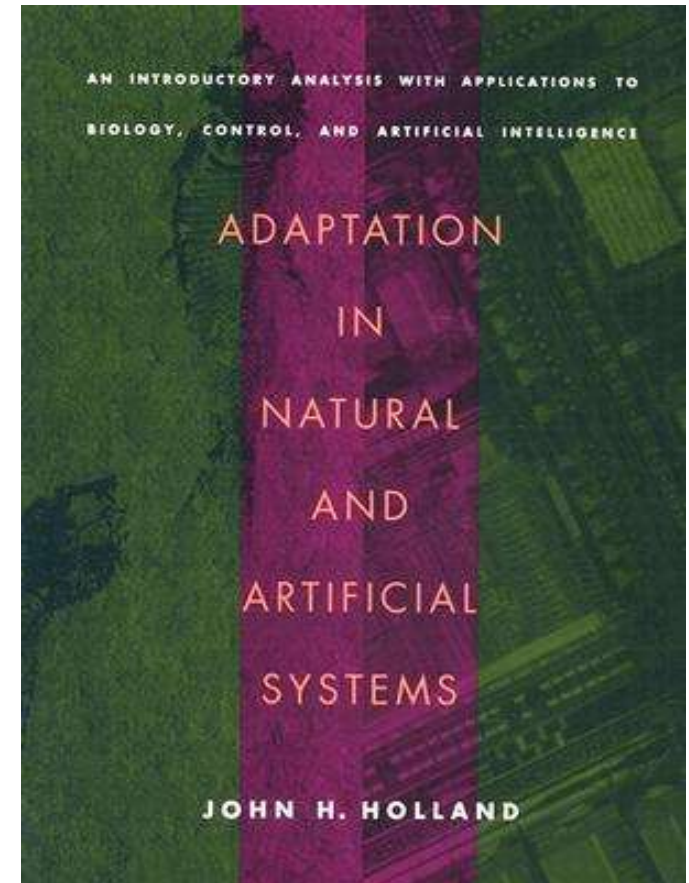


Evolutionary Algorithms (EAs)

- EAs are optimization algorithms based on the theory of evolution
- **(1+1) EA**: the simplest EA
- **Genetic Algorithms**: the most popular EA
- But there are more...

Genetic Algorithms (GAs)

- 1950s: *Turing* proposed use of evolution in computer programs
- 1970s: John Holland created GAs to address optimization problems
- Simulate evolution of an entire *population* of individuals, which procreate sexually



GA Components

- *Chromosome*: the actual representation of the individuals, eg binary string
- *Population*: keep track of several individuals
- *Generation*: individuals mate and reproduce
- *Selection*: how mating is done, based on *fitness function*
- *Mutation*: offspring might have some of their “DNA” mutated
- *Crossover*: offspring inherit genetic material from the parents

Chromosome

1	1	0	0	0	1	0	1	0	0
---	---	---	---	---	---	---	---	---	---

- A solution to an optimization problem will be represented with some data
- In a computer, any stored data can be seen as 0/1 bitstring
- Bitstring is a common representation, but there can be customized for any addressed domain

Population

- Keep track of several individuals which we try to optimize
- At the end of the search, return the best solution in the population



Generations



- The search will be composed of 1 or more *generations*
- At each generation, select individuals for reproduction
- Create a new generation of same size K
- Kill the previous generation
 - Yes, evolution is really cruel...

Selection



- The fittest individuals will have higher chances of reproduction
- Different strategies for parent selection
- Tournament Selection: sample T individuals *randomly* from the population, and choose the best among them
- *Fitness function is used to determine who is better*

Sexual Reproduction



- Select 2 parents, which give birth to 2 offspring
 - Note, ignoring gender here...
- Offspring will share genetic material with their parents, via the *crossover* operation (aka xover)
- After xover, offspring still have chances of getting mutated

Crossover

Parent 1

1	1	0	0	0	1	0	1	0	0
---	---	---	---	---	---	---	---	---	---

Parent 2

0	1	0	1	0	1	0	0	0	1
---	---	---	---	---	---	---	---	---	---

Choose a splitting point, eg the middle
of the chromosomes

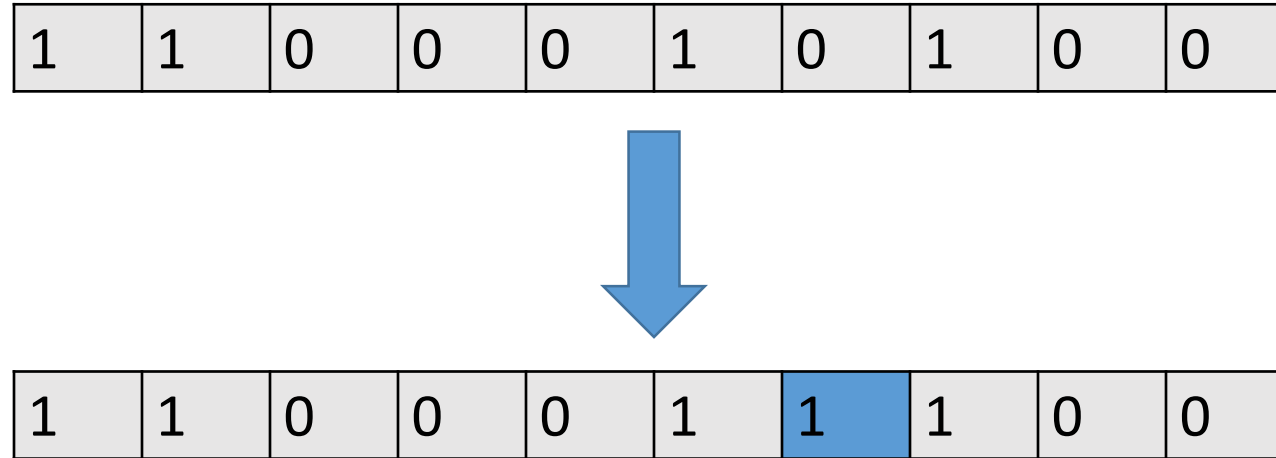
Offspring 1

1	1	0	0	0	1	0	0	0	1
---	---	---	---	---	---	---	---	---	---

Offspring 2

0	1	0	1	0	1	0	1	0	0
---	---	---	---	---	---	---	---	---	---

Mutation in 0/1 Sequence



- Mutation: given N bits, each bit can be flipped with probability $1/N$
 - Eg 10% probability in the above example
- *On average*, only 1 bit is flipped per mutation operation

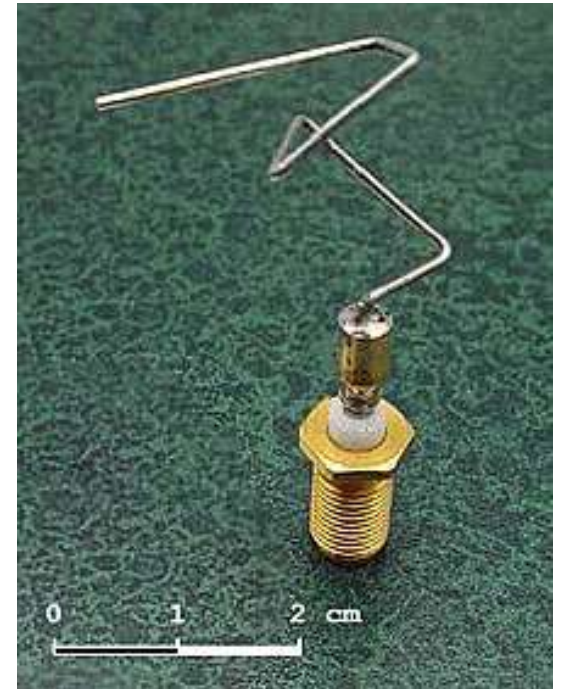
Does this “crazy” stuff actually work???

Usage

- *Genetic Algorithms are among the most used kind of optimization algorithms*
- Successfully used in a lot, a lot of different domains
- Goldberg's 1989 book on GAs is *among the most cited books* in the *whole* field of Computer Science

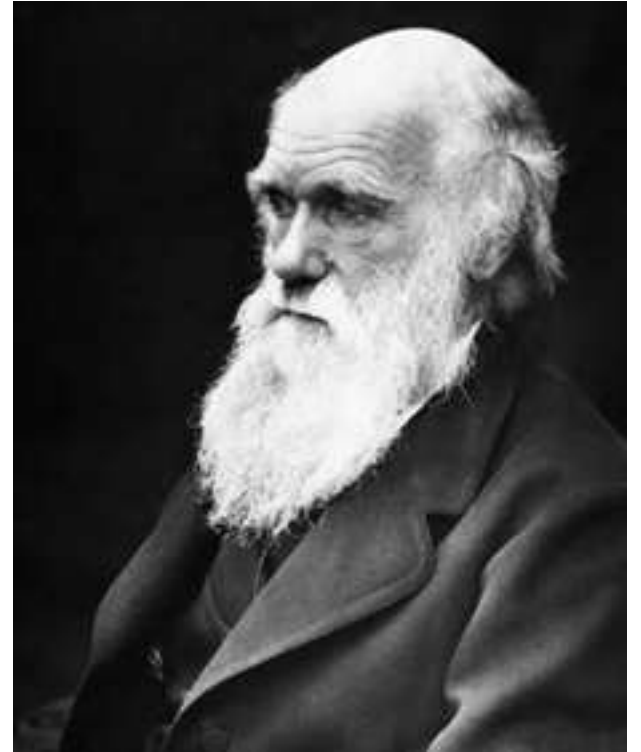
Example

- Evolution of antenna designs
 - Optimize beamwidth and impedance bandwidth
- Best found with an Evolutionary Algorithm
- Used for example in NASA spacecrafts



Search-Based Software Testing (SBST)

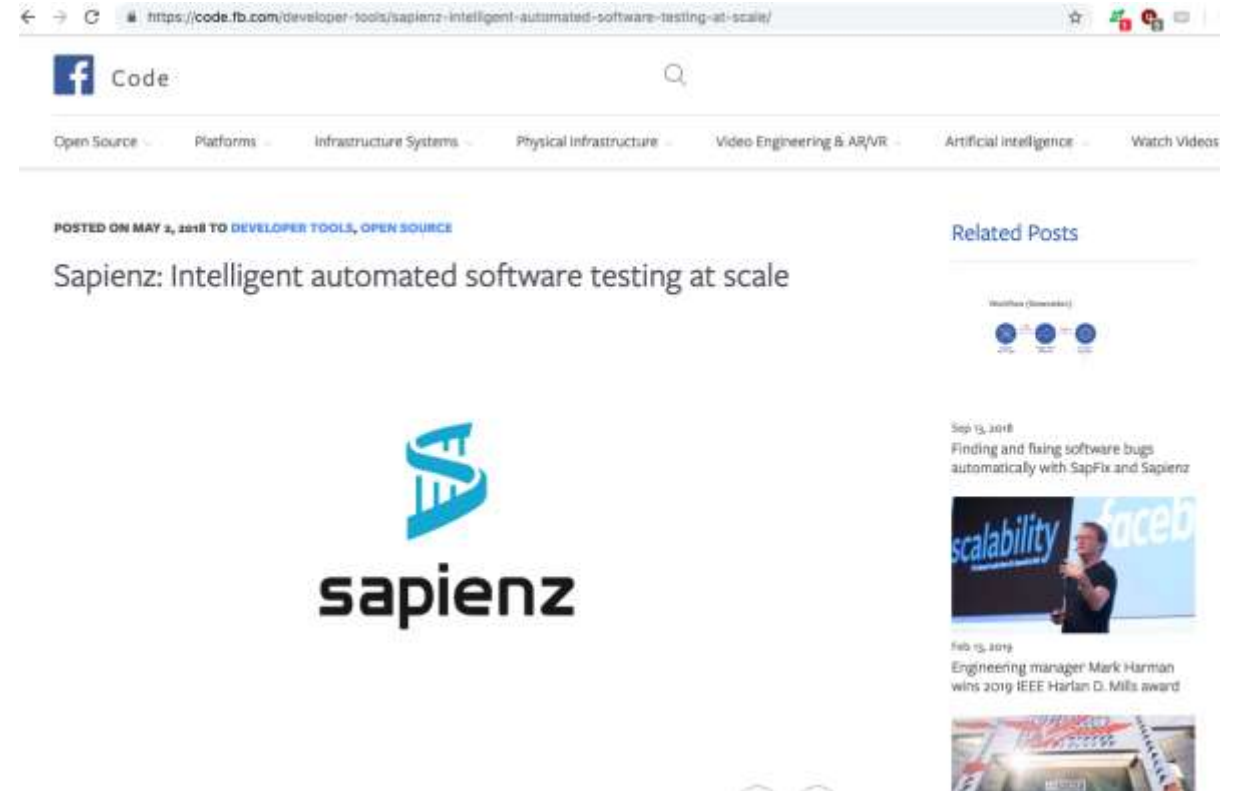
- Biology meets Software Engineering
- *Evolve* test cases
- *Genetic Algorithms*
- *Chromosome*: test cases as sequences of function calls with their inputs
- *Fitness*: based on code coverage, bug finding, etc
 - lot, lot of technical details here...



Success Stories: Facebook

Facebook uses SBST for automatically testing their software, especially their mobile apps

- eg, tools like *Sapienz* and *SapFix*



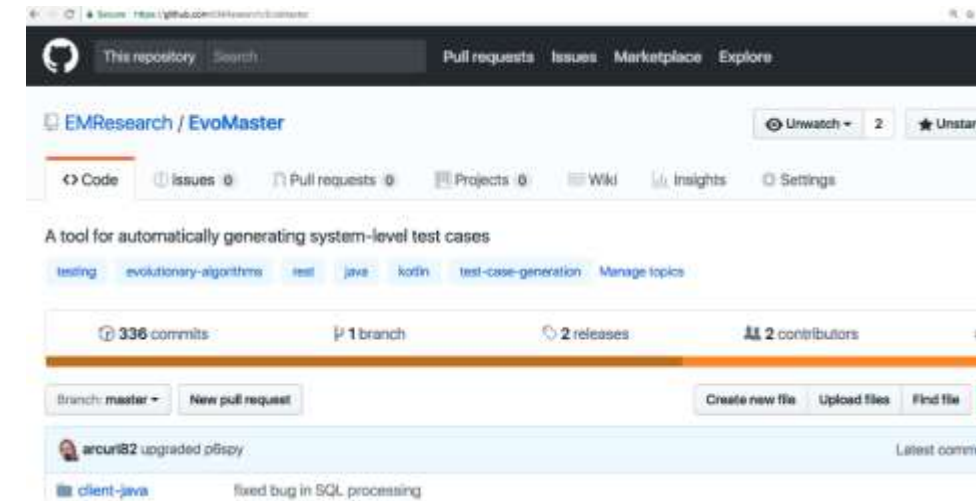
Previous project: *EvoSuite* (2010-2016)

- Automated test generation for **Java unit testing**, with a single mouse click
- International effort (eg Germany and UK)
- Used to find hundreds of bugs in hundreds of systems
- More than 40 publications, for more than 2000 citations
- Free, **open-source** tool that can already be **used in industry**
- *Academic research can and should have impact on industry practices*



Current project: *EvoMaster* (2016-now)

- Next step, from single units, to test whole systems
- Test generation for enterprise systems:
 - Web services
 - Databases
 - GUIs
 - Mobiles
- Much more complicated than unit testing
 - Eg, several processes, distributed systems, networking, much larger scale
- *IMPORTANT: **open-source** tool that practitioners can use in their daily work*



Industry Collaborations

- Apply research tools to open-source projects is good, but not enough
- Need direct contact with engineers, to understand their problems and try out research solutions in real contexts
- Example: current collaborations with ***Universitetsforlaget*** and ***Skatteetaten*** (just started) to apply our research results on their systems
- *Academic research can and should have impact on practice*

Demo