Cool and Ripe for Exploitation: Search-Based Software Engineering

Chris Simons

Department of Computer Science and Creative Technologies

University of the West of England

Bristol BS16 1QY United Kingdom

chris.simons@uwe.ac.uk

http://www.cems.uwe.ac.uk/~clsimons/

ACCU 2014 Conference

8 – 12 April 2014

Agenda

- Motivation
- The very good idea - search
- Drawing inspiration from nature
- Examples – genetic programming, and more
- The developer and interactive search
- Some resources available


MORDAC, THE PREVENTER OF INFORMATION SERVICES.
SECURITY IS MORE IMPORTANT THAN USABILITY.
www.dilbert.com
scottadams@aol.com
IN A PERFECT WORLD, NO ONE WOULD BE ABLE TO USE ANYTHING.
©2007 Scott Adams, Inc./Dist. by UFS, Inc.
To complete the log-in procedure, stare directly at the sun.

Motivation

- Programming solutions and trade-offs are difficult
  - Where do solutions come from??
  - And then when we've got some,
    - Robust & rock solid or ship to customer now?
    - Hard-code a kludge or fix the error ?
    - Architect a solution or patch-the-patch?
    - Etc. etc.

Any number of trade-offs during development?

Why is this difficult?

- Complex development problems
  - Dependencies
- Measurements
  - Quantitative, objective
- Value judgements
  - Qualitative, subjective
- Difficult to predict full consequences of candidate solutions?

Finding candidate trade-off solutions

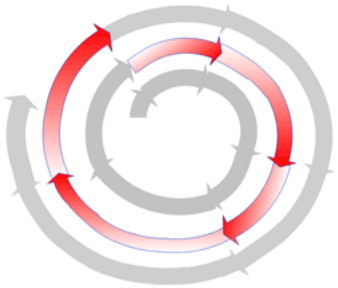
- ***Deduction***
  - Reasoning from one or more general statements (premises) to reach a logically certain conclusion(s)
  - E.g. reuse of patterns, idioms, libraries etc.
- ***Induction***
  - Conclusion is inherently uncertain and probably the truth
  - E.g. trial & error, generate & test in complex, novel problems
- ***Abduction***
  - From observation, to inference, to hypothesis
  - E.g. designing experiments, comparing this to that

Focussing on inductive reasoning….

WHAT COULD POSSIBLY WORK?

formulate a candidate solution

i. Construct(s)?
ii. Assembly?
iii. Configuration?

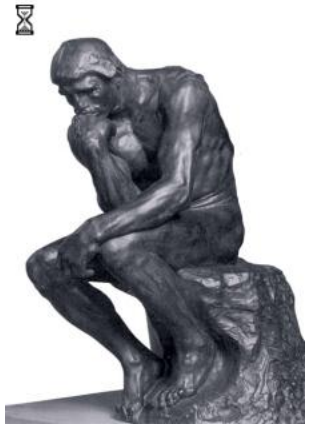


ANSWER
the requirement!

IS IT ANY GOOD?

evaluate the candidate solution

i. Fitness?
ii. Measure / Metrics?
iii. TEST SPECIFICATION!

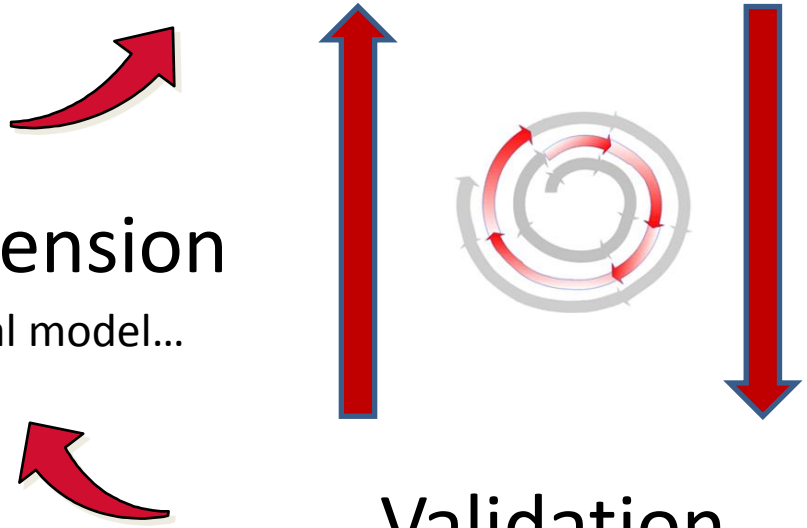


Evidence from Psychology

"Mental Models"

Description

Reasoners attempt to formulate a model…



Comprehension

An internal model…

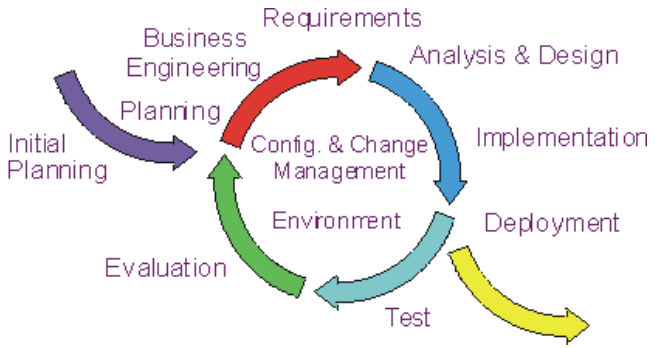
Validation

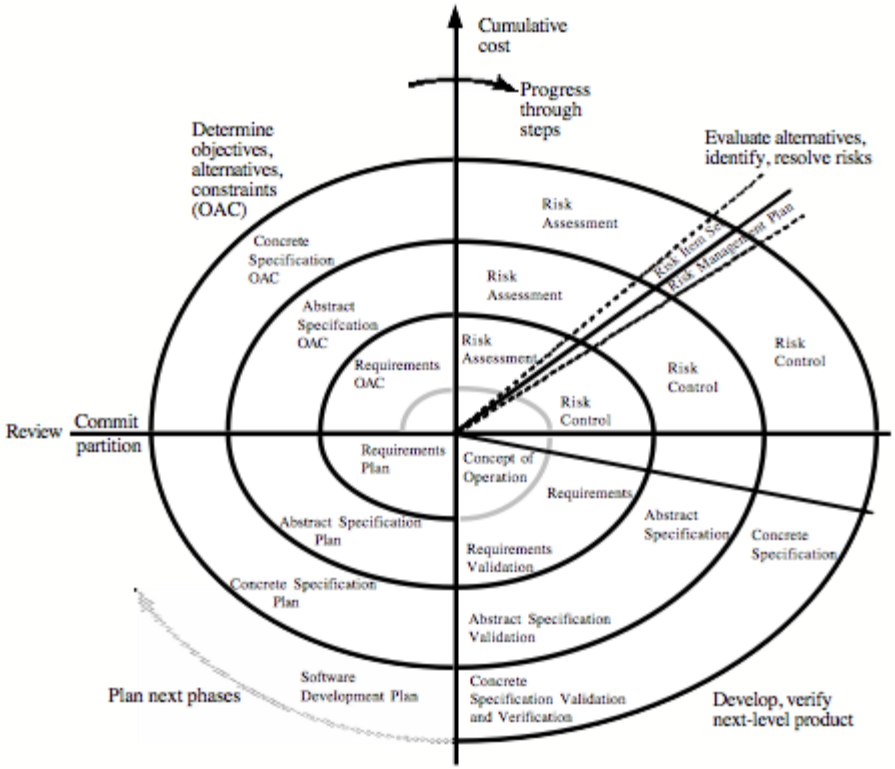
Is there an alternative model?
If yes, better? Keep, else discard

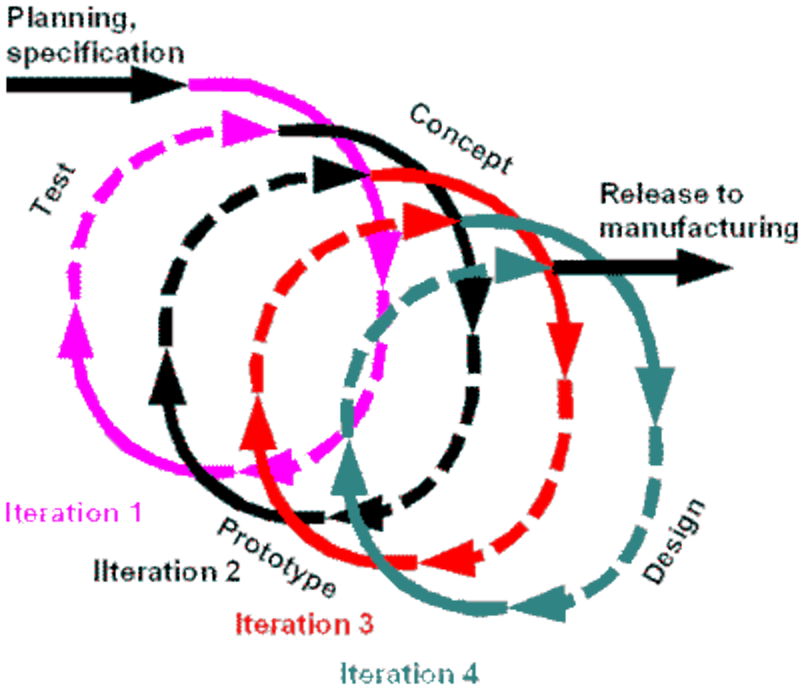
"Because the number of mental models is finite[...], the search can in principle be exhaustive. If it is uncertain whether there is an alternative model to the premises, then the conclusion can be drawn in a tentative and probabilistic way"

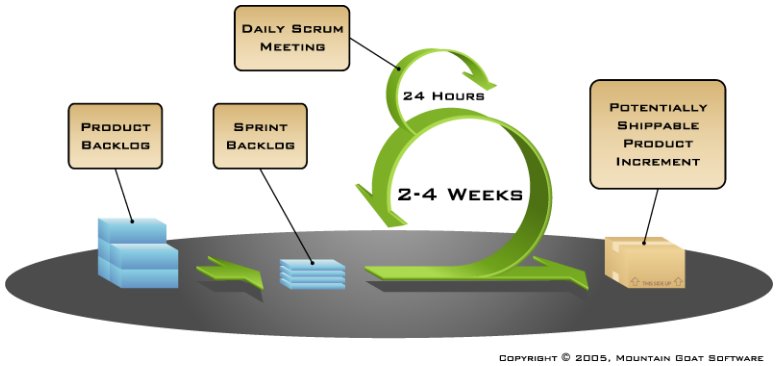
Johnson-Laird, P., Byrne, R. (1991) *Deduction*, Laurence Erlbaum Associates.

Evidence: Iterative & Incremental Process









Some forerunner hints (1)…

Development as 'Search' strategies

"Make educated guesses about the kinds of inventions that you'll need based on the nature of your application. We guide our search from the perspectives of:

- *Decision making, control and coordination activities,*
- *Structures and groups of objects,*
- *Etc….*

Trade-off?

The best way to evaluate potential candidates […] is to shift perspective.

Stop brainstorming candidates when you run out of energy."

Wirfs-Brock, R., and McKean, A. (2003) *Object Design: Roles, Responsibilities, and Collaborations.* Addison-Wesley, Boston, MA, USA, p 84.

Some forerunner hints (2)…

Trade-off?

"Fact 27: There is seldom one best design solution to a software problem."

"Fact 28: Design is a complex, iterative process. The initial design solution will likely be wrong and certainly not optimal."

55 facts and 10 fallacies in:

Glass, R. (2003) *Fact and Fallacies of Software Engineering.* Addison-Wesley, Pearson Education, pp. 79-83.

Some forerunner hints(3)….

As we iteratively engineer software, we must ***adapt*** to (changing) requirements and technical debt…

"If there is one thing of which we can be sure, it is that a system of any substantial size is going to evolve. It is going to evolve even as it is undergoing development. Later, when in use, the changing environment will call for further evolution. … the system itself should be resilient to change, or change tolerant. Another way of putting this goal is to say that the system should be capable of evolving gracefully."

Jacobson, I., Booch, G., Rumbaugh, J. (1999) *The Unified Software Development Process*, Addison Wesley.

So what's the very good idea?

Rather than attempting to formulate a solution…

Formulate the ***space*** of all solutions, then…

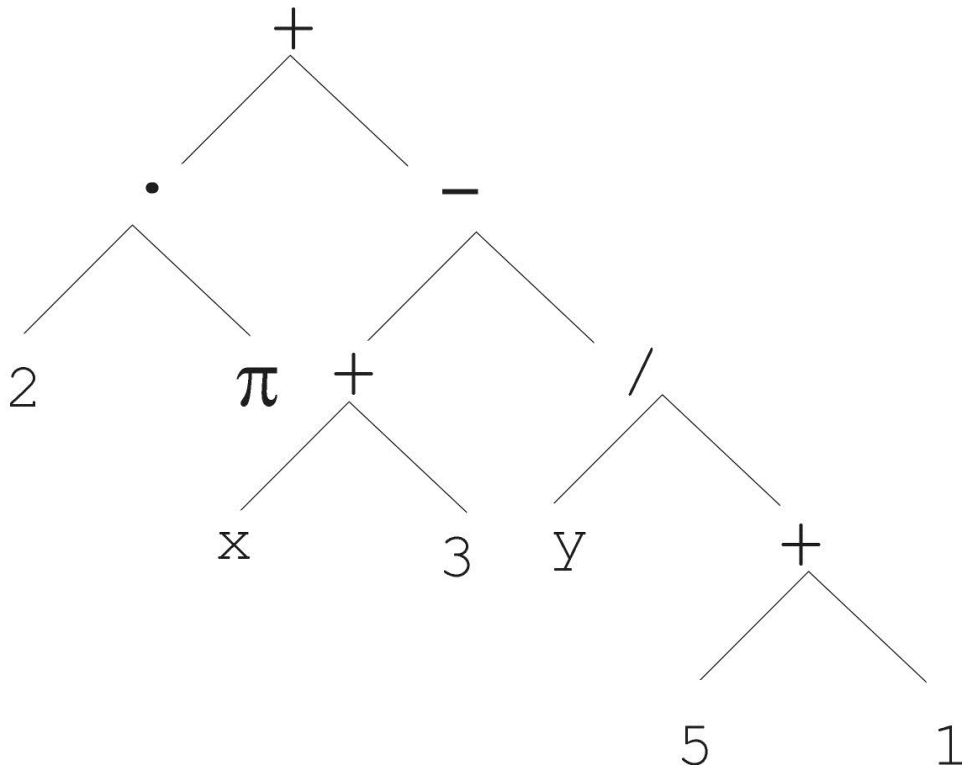
iterate (i.e.) ***search*** over the space

We need three things for search

1. *Representation*
   - To encode all the possibilities of the search space
2. *Fitness measures*
   - To evaluate possibilities in the search space
3. *Search Approach*
   - E.g. brute-force exhaustive enumeration is straightforward

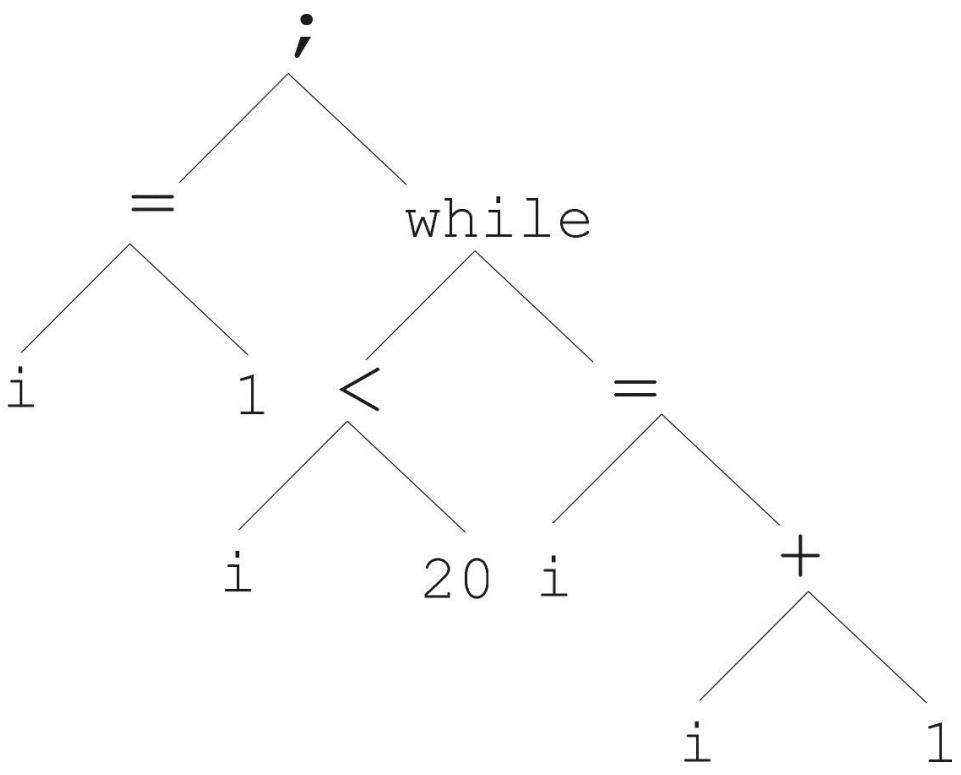
Representation – tree example

$$2 \cdot \pi + \left((x+3) - \frac{y}{5+1} \right)$$



Representation – tree example

```
i = 1;
while( i < 20 )
{
      i = i + 1
}
```



- Other representations are available e.g.
  - Binary, integer, float, character
  - Structures, as sequences, sets, maps etc.

Fitness measures - examples

- *Quantitative, e.g.*
  - Coupling, cohesions, complexity etc.
  - Security, performance, robustness etc.
- *Qualitative, e.g.*
  - Elegance, understandability, etc.

more on this later…

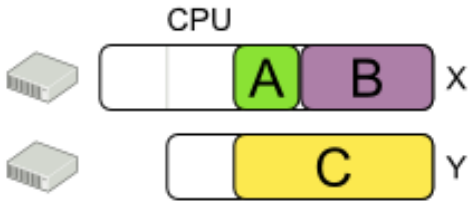
Search Approach?

- Can exhaustively enumerate over each solution, but
  - Spaces get big, very quickly…

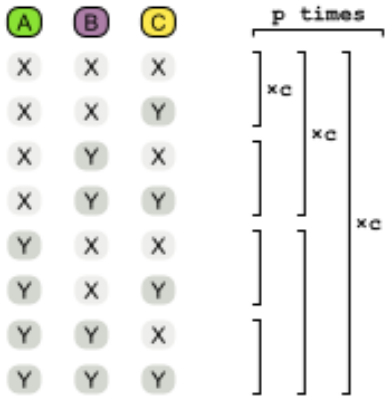
Calculate the size of the search space

Given a Solution model, how many different combinations can it represent?

Cloud balancing



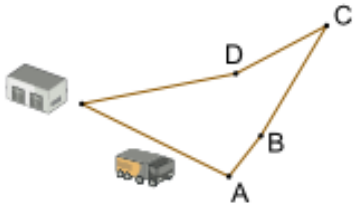
Model:



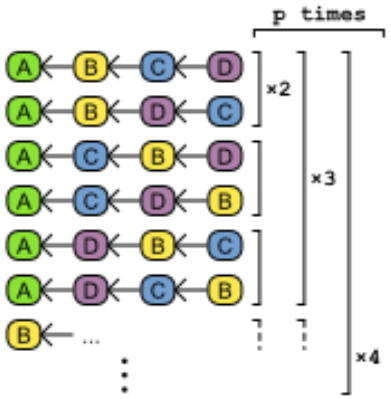
Search space: c^p

# computers	# processes	search space
2	3	8
100	300	10^{600}
200	600	10^{1380}
400	1200	10^{6967}

Traveling salesman (TSP)



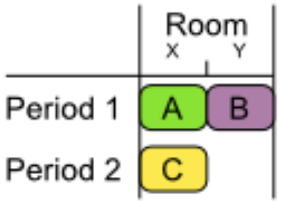
Model: linked list



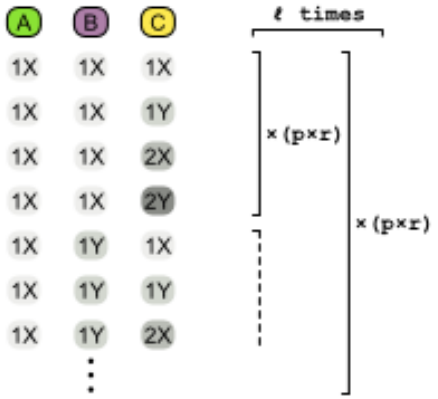
Search space: $n!$

# customers	search space
4	24
100	10^{157}
1000	10^{2567}
10000	10^{35659}

Course scheduling



Model:



Search space: $(p \times r)^{\ell}$

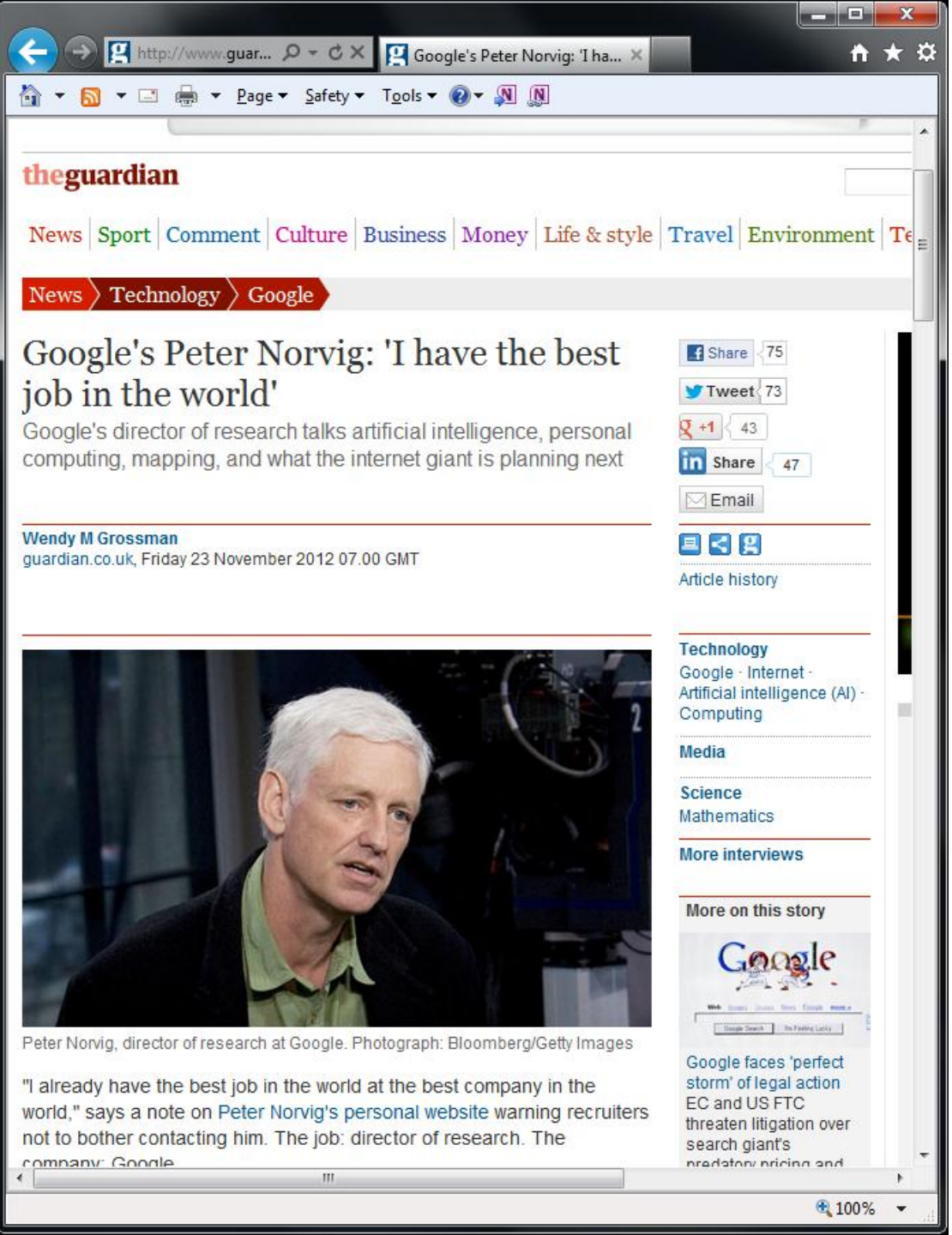
# periods	# rooms	# lectures	space
2	2	3	64
36	6	100	10^{233}
36	18	400	10^{1124}
36	36	800	10^{2490}

http://www.optaplanner.org/blog/2014/03/27/searchSpaceSizeCalculation.png

Observation & Inference:

Exhaustive enumeration quickly becomes intractable

Could the size of search space in software development be one of the causes of its difficulty?

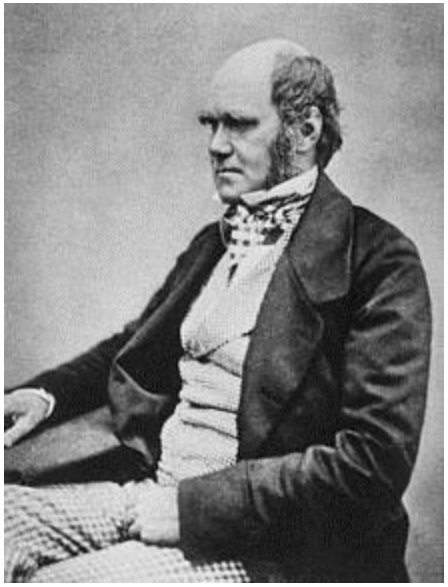


"This idea that intelligence is the one thing that amplifies itself indefinitely, I guess is what I'm resistant to. Intelligence can let you solve harder problems, but some problems are just resistant, and you get to a point that being smarter isn't going to help you at all, and I think a lot of our problems are like that. Like in politics - it's not like we're saying that if only we had a politician who was slightly smarter all our problems would go away."

So what to do?

Take inspiration from nature…

Evolution in nature

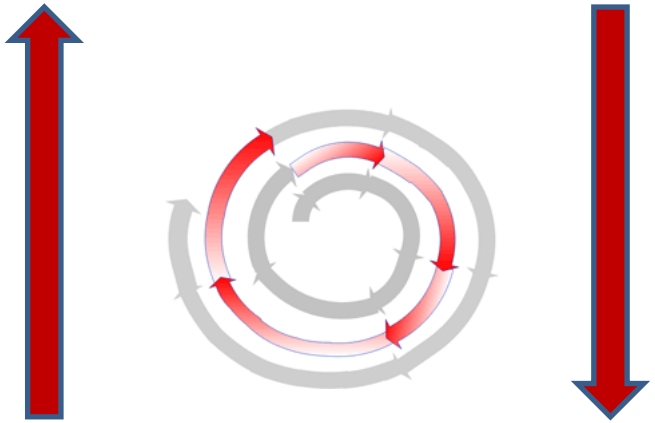


Darwin, aged 45 in 1854,
by then working towards
On the Origin of Species

Evolution is the change in the inherited characteristics of biological populations over successive generations.

environment

Selection of fittest individuals



sexual reproduction for
diversity and population change

(Evolutionary) Search

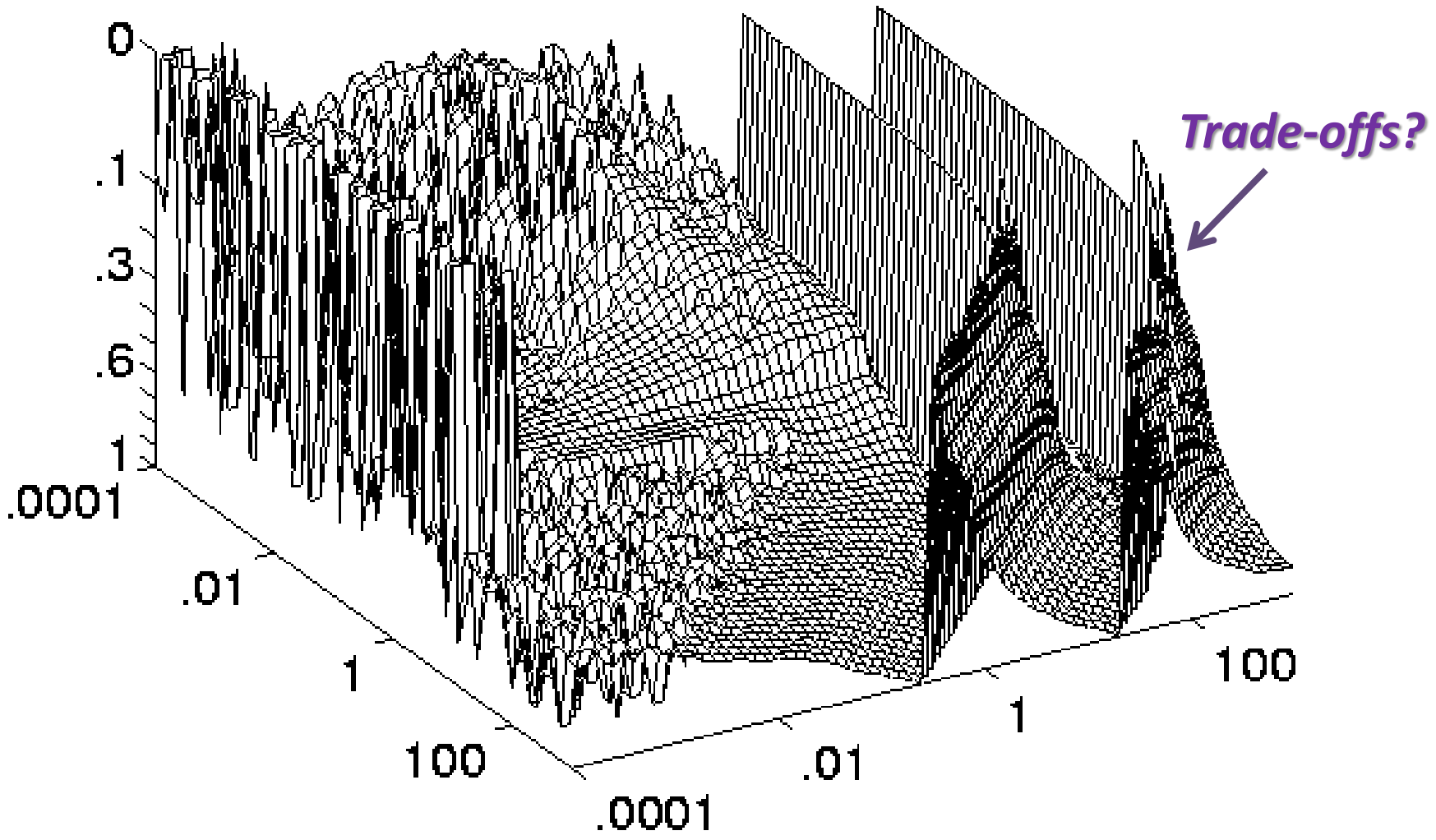
Representation of an "individual" solution

e.g. models, trees, arrays etc. etc.

```
initialise population at random
while( not done )
  evaluate each individual
  select parents
  recombine pairs of parents
  mutate new candidate individuals
  select candidates for next generation
end while
```

Eiben, A.E., Smith, J.E. (2003) *Introduction to Evolutionary Computing*, Springer.

Software Fitness Landscape



Evolutionary Computing

Not new…

- Alan Turin (1952)
  - "Computing Machinery and Intelligence" in *Mind*
  - hints at a *"…genetical programming..."*
- Alex Fraser (1957)
  - Computational simulation of natural evolution
- Fogel *et al.* (1966)
  - Evolutionary programming (finite state machines)
- Rechenburg (1973)
  - Evolutionary Strategies
- Holland (1975)
  - Genetic Algorithms
- Kosa (1992)
  - Genetic Programming

many, many more!!

Search-Based Software Engineering

The term “Search-Based Software Engineering” (SBSE) coined in 2001.

“… a new field of software engineering research and practice is emerging: search-based software engineering. The paper argues that software engineering is ideal for the application of […] search techniques.

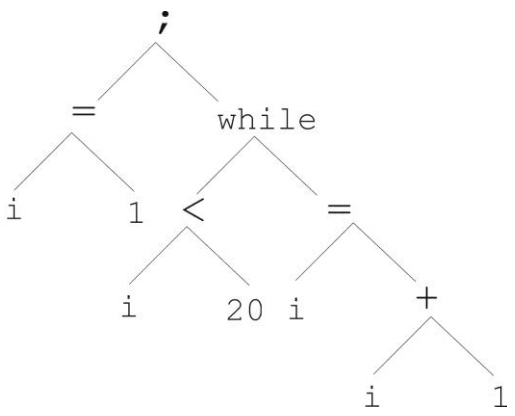
Such search-based techniques could provide solutions to the difficult problems of balancing competing (and some times inconsistent) constraints and may suggest ways of finding acceptable solutions in situations where perfect solutions are either theoretically impossible or practically infeasible”.

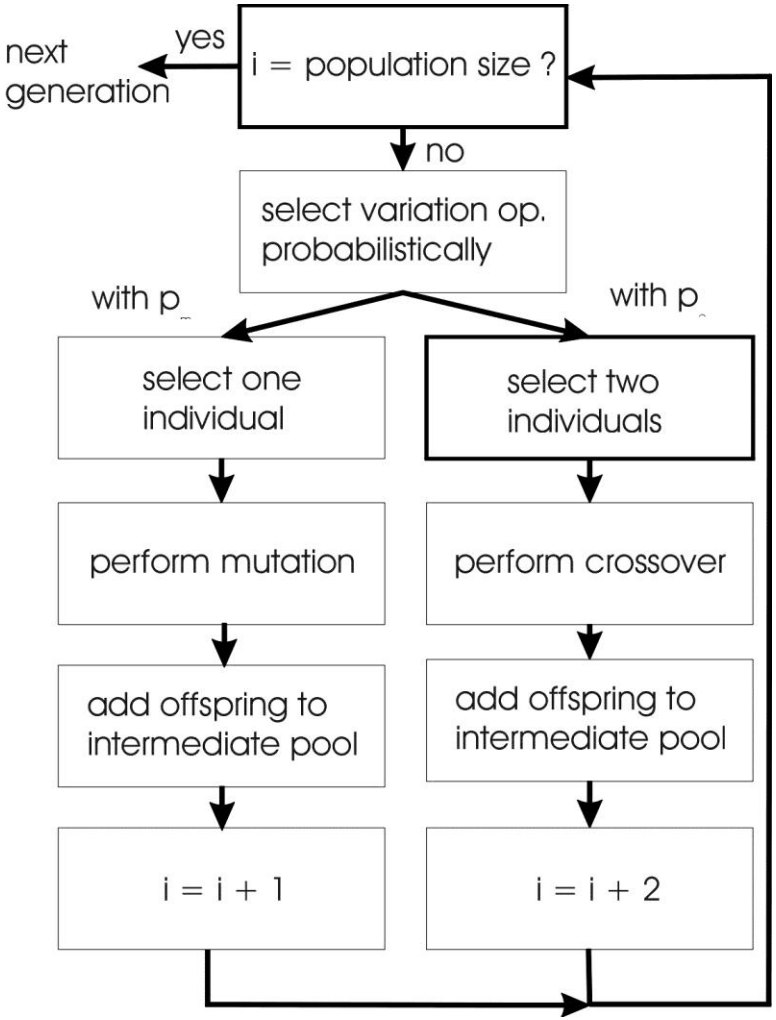
Harman, M., Jones, B.J. (2001) Search-Based Software Engineering. *Information and Software Technology,* vol. 43, no.14, pp. 833-839.

Harman, M. (2011) Software Engineering meets Evolutionary Computation. *Computer,* vol. 44, no.10, pp. 31-39.

Search in code - Genetic Programming

```
i = 1;
while( i < 20 )
{
        i = i + 1
}
```





Parse tree symbolic expressions defined by:

- Set of *terminals*
- Set of *functions*

Genetic Programming Example

Simple line fitting using regression

http://alphard.ethz.ch/gerber/approx/default.html

files

Genetic Programming Example

"We evolved a widely-used and highly complex 50000 line system, seeking improved versions that are faster than the original, yet as least as good semantically. Our approach found a version that is 70 times faster (on average) and is also a small Semantic improvement on the original".

Note the multiple objective trade-off between performance and semantics

Langdon, W.B., Harman, M. (2012) Genetically Improving 50000 Lines of C++, Research Note, University College London.

See also:

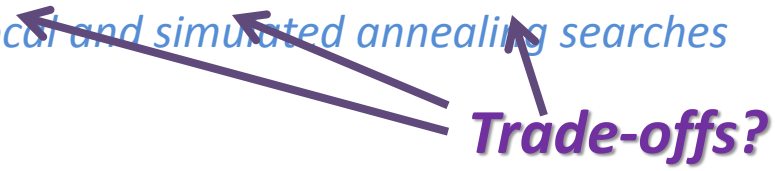
Poli, R., Langdon, W.B., McPhee, N.F. (2008) *A Field Guide to Genetic Programming,* LuLu Publishing.

Other examples of Search (1)

- ***Automatic Refactoring***
  - Representation
    - Parse trees of object-oriented source code
  - Fitness
    - Cohesion (maximise), coupling (minimise), dependency etc.
  - Diversity preservation
    - Crossover and mutation using Fowler refactoring patterns
      - E.g. push down, pull up, collapse hierarchy etc.
  - Search approach
    - Evolutionary algorithm (and other), 'population' of parse trees

Other examples of Search (1)

*"The high cost of software maintenance could be reduced by automatically improving the design of object-oriented programs without altering their behaviour. We have constructed a software tool capable of refactoring object-oriented programs to conform more closely to a given design quality model, by formulating the task as a search problem in the space of alternative designs. This novel approach is validated by two case studies, where programs are automatically refactored to increase **flexibility**, **reusability** and **understandability** as defined by a contemporary quality model. Both local and simulated annealing searches were found to be effective in this task."*



O'Keefe, M., O Cinneide, M. (2008) Search-based Refactoring for Software Maintenance. Journal of Systems and Software, vol. 81, Iss. 4, pp. 502-516.

Limitation: If the population size of search is, say, 100, which one is best? For Large scale software systems, can the programmer compare candidates?

Other examples of Search (2)

- ***Automatic Test Case Generation***
  - Representation
    - Inputs of a test case (sequence of ints, floats, strings etc.)
  - Fitness
    - Branch coverage (maximise), execution time (minimise), etc.
  - Diversity preservation
    - Crossover and mutation at random
  - Search approach
    - Evolutionary algorithm (and other), 'population' of test cases representing a test suite.

Other examples of Search (2)

"Search-Based Software Testing is the use of a […] search technique, such as a Genetic Algorithm, to automate or partially automate a testing task, for example the automatic generation of test data. Key to the optimization process is a problem-specific fitness function. The role of the fitness function is to guide the search to good solutions from a potentially infinite search space, within a practical time limit. Work on Search-Based Software Testing dates back to 1976, with interest in the area beginning to gather pace in the 1990s. More recently there has been an explosion of the amount of work. This paper reviews past work and the current state of the art, and discusses potential future research areas and open problems that remain in the field."

McMinn, P. (2011) Search-Based Software Testing: Past, Present and Future. 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), IEEE Press, pp. 153-163.

Observation: Especially beneficial in regression testing

Other examples of Search (3)

- ***Automatic Bug Fixing***
  - Representation
    - Parse trees of source code
  - Fitness
    - %age tests passing in existing test suites
  - Diversity preservation
    - Crossover and mutation at random
  - Search approach
    - Genetic programming, population of source code variants

Other examples of Search (3)

"This paper describes GenProg, an automated method for repairing defects in off-the-shelf, legacy programs without formal specifications, program annotations, or special coding practices. GenProg uses an extended form of genetic programming to evolve a program variant that retains required functionality but is not susceptible to a given defect, using existing test suites to encode both the defect and required functionality. Structural differencing algorithms and delta debugging reduce the difference between this variant and the original program to a minimal repair. We describe the algorithm and report experimental results of its success on 16 programs totaling 1.25 M lines of C code and 120K lines of module code, spanning eight classes of defects, in 357 seconds, on average. We analyze the generated repairs qualitatively and quantitatively to demonstrate that the process efficiently produces evolved programs that repair the defect, are not fragile input memorizations, and do not lead to serious degradation in functionality."

Le Goues, C., Nguyen, T., Forrest, S., Weimer, W. (2012) GenProg: A Generic Method For Automatic Program Repair. IEEE Transactions on Software Engineering, vol. 38, no. 1, pp. 54-72.

SBSE repository

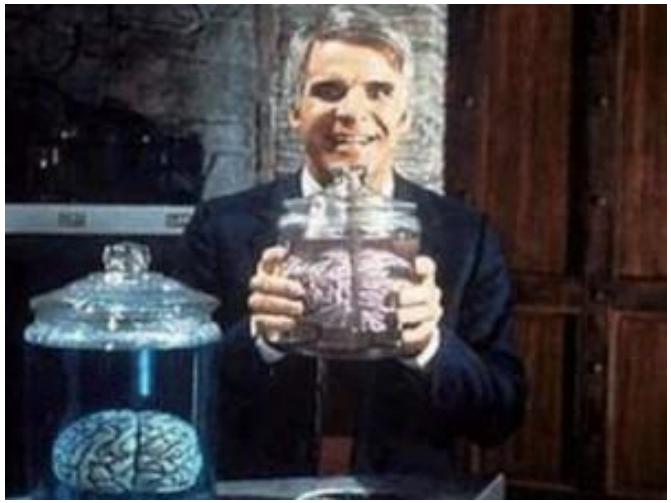
Zhang, Y. (2014) Repository of Publications on Search-based Software Engineering. http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/.

So given the successful applications of SBSE, is programming now just a case of (i) representation, (ii) fitness measures, (iii) search algorithm?

Where is the programmer?

Simons, C.L. (2013) "Whither (Away) Software Engineers in SBSE?", in First International Workshop on Combining Modelling with Search-Based Software Engineering (CMSBSE 2013), in conjunction with the International Conference on Software Engineering, (ICSE '13), IEEE Press, pp.49-50.

- We tried to ***replace*** people to fully automate
  - Didn't really work…?



- Better as a human-machine ***partnership***
- But partnership requires agreement
  - mutually predictable actions
  - maintain common ground

Klein, G., *et al.*, (2004) "Ten Challenges for Making Automation a 'Team Player' in Joint Human-Agent Activity", *IEEE Intelligent Systems*, vol. 19, no. 6., pp. 91-95.

What is “evaluation”?

A value judgement…

What is “evaluation”?

A value judgement…

Objective?

Software Metrics e.g. lines of code, coupling, cohesion….

Subjective?

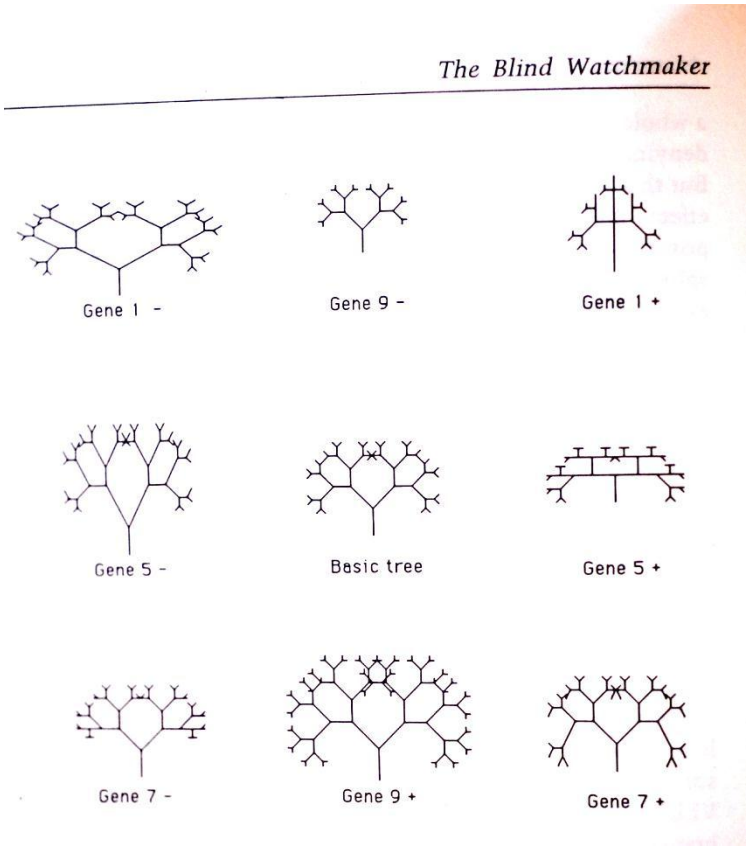
Value judgement e.g. elegance, symmetry, patterns….

A combination of both i.e. “multi-obsubjective”

Interactive Evolutionary Computing

- Human in the fitness function -- e.g. Dawkins 'biomorphs'

  Dawkins, R. (1986) *The Blind Watchmaker,* Penguin Books.



- Since then:
  - Art
  - Music
  - Image processing
  - Games
  - Industrial product design
  - Fashion Design
  - Control and robotics
  - Etc. etc. etc.

Tagaki, H. (2001) Interactive Evolutionary Computation: A Fusion of the Capabilities of EC Optimisation and Human Evaluation. *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1275-1296.

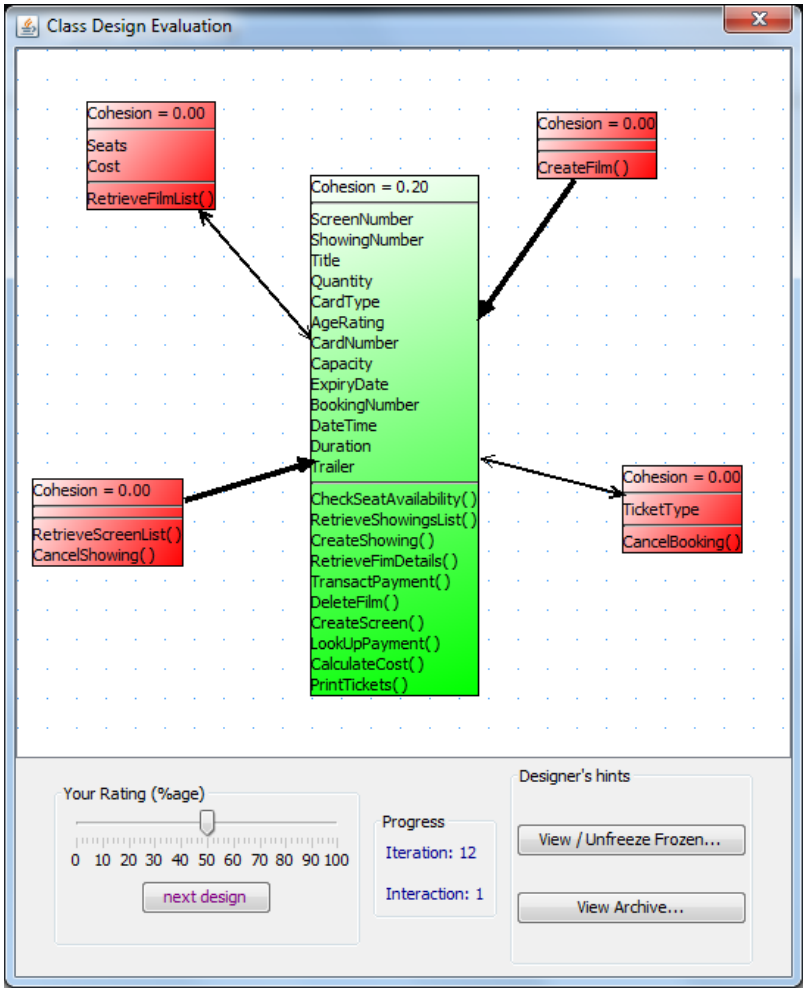
Evaluation: Aesthetics, Elegance?

"the significance of symmetry was only made clear with the discovery that stress and disease make it harder for an individual to develop a perfectly symmetric body. Small differences on either side of an imaginary mid-plane therefore betray genetic quality, and potential mates use this to gauge each other's desirability. Put simply, symmetry is sexy".

Schilthuizen, M., "Lopsided Love", *New Scientist,* 18 June 2011, pp. 42-45.

M.C.Escher

Coupling vs Elegance in Software Design

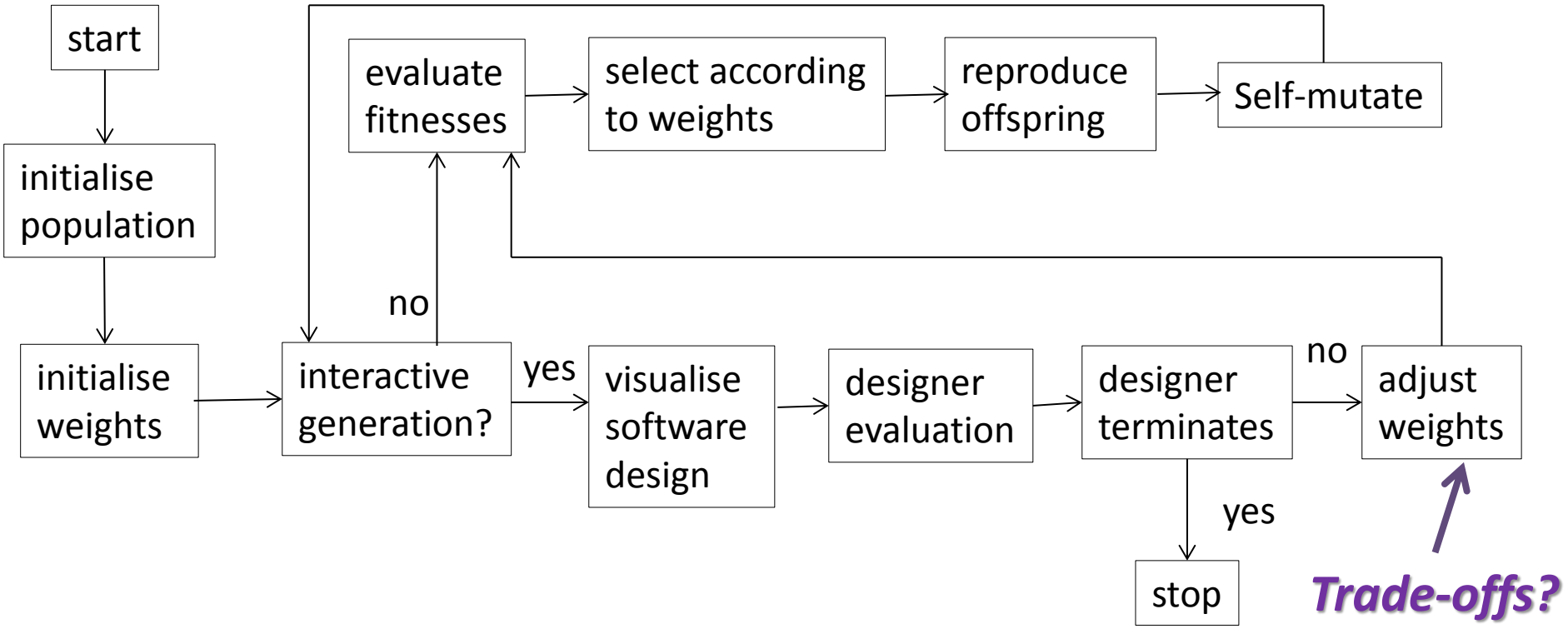


versus

Brown, W.J., *et al.*, (1998) *Anti-Patterns: Refactoring Software Architectures and Projects in Crisis*, Wiley.

Example (1) Interactive Software Design

- Novel elegance metrics
  - Numbers among classes, Ratio of Attributes to Methods

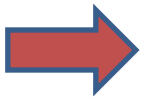


Simons, C.L., Smith, J.E. (2012) Elegant Object-oriented Software Design via Interactive Evolutionary Computation, *IEEE Transactions on Systems, Man and Cybernetics – Part C*, vol. 42, iss. 6, pp. 1797-1805.

Elegance

- Significant role in searching (evolving) effective software designs
- Wide potential for other design fields via interactive search

Example (2) Interactive Software Testing



"Search-based testing could be a key aspect in creating an automated tool for supporting testing activities. However, domain specific quality criteria and trade-offs make it difficult to develop a general fitness function a priori, so interaction between domain specialists and such a tool would be critical to its success.

In this paper, we present a system for interactive search-based software testing and investigate a way for domain specialists to guide the search by dynamically re-weighting quality goals. Our empirical investigation shows that objective re-weighting can help a human domain specialist interactively guide the search, without requiring specialised knowledge of the system and without sacrificing population diversity".

Observation: beneficial to emphasise quality goals over mechanistic test generation

Marculescu, B., Feldt, R., Torkar, R. (2013) Objective Re-Weighting to Guide an Interactive Search Based Software Testing System. In Proceedings of the 12th International Conference on Machine Learning and Applications (ICMLA'13)

Challenges ahead?

- Searching for strategies rather than instances
- Exploiting many-core computing
- Giving insight to software developers
- Optimising compilation and deployment
- Balancing Computation and Human Interaction

Harman, M. (2012) “The Role of Artificial Intelligence in Software Engineering”, *Proceedings of the First International Workshop on Realising Artificial Intelligence in Software Engineering (RAISE)* , pp. 1-6.

Some resources available

- Evolving Objects (EO): an Evolutionary Computation Framework (C++)
  - http://eodev.sourceforge.net/
- Open BEAGLE (C++)
  - https://code.google.com/p/beagle/
- ECJ 21 (Java Evolutionary Computation)
  - http://cs.gmu.edu/~eclab/projects/ecj/
- ECF (Evolutionary Computational Framework) (C++)
  - http://gp.zemris.fer.hr/ecf/
- JCLEC – Java Class Library for Evolutionary Computation
  - http://jclec.sourceforge.net/
- Etc. etc.

And finally…
there's even an article on a GA in Overload!

Buontempo, F. (2013) How to Program Your Way Out of a Paper Bag Using Genetic Algorithms. *Overload*, Iss. 118 (December 2013).

http://www.accu.org/index.php/journals/1825

Conclusions

- Search-based Software Engineering is coming of age
- Search helps developers to be more productive
- Ripe for exploitation

I'm happy to collaborate on SBSE tools and their application

Thank you

- Any questions?