Dynamic Adaptive Search Based Software Engineering

Mark Harman

Dynamic Adaptive SBSE

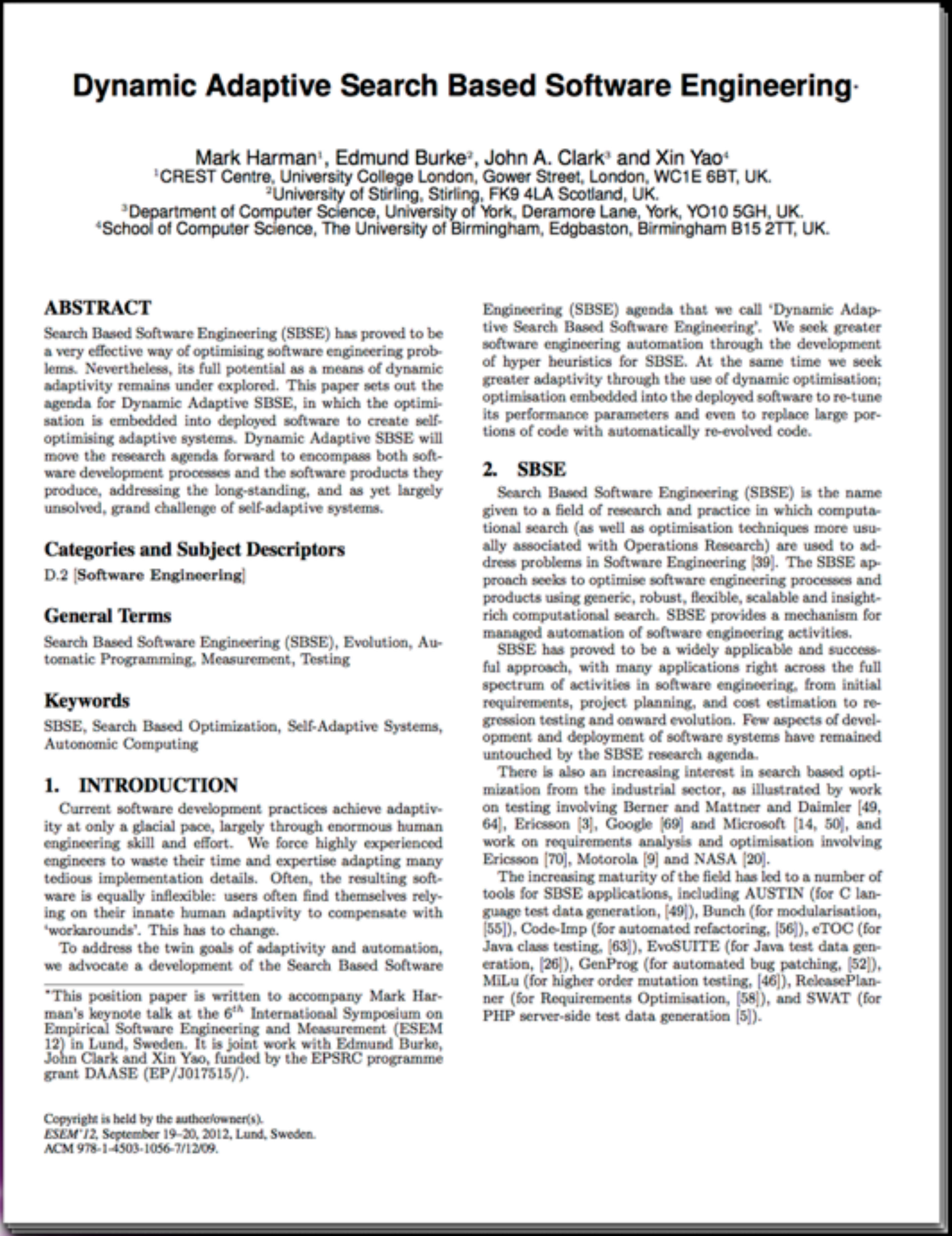
Compile SBSE into deployed Software

Dynamic Adaptive SBSE

Compile SBSE into deployed Software

What do you mean?

There is a paper that accompanied my keynote



Dynamic Adaptive Search Based Software Engineering[*]

Mark Harman[1], Edmund Burke[2], John A. Clark[3] and Xin Yao[4]
[1]CREST Centre, University College London, Gower Street, London, WC1E 6BT, UK.
[2]University of Stirling, Stirling, FK9 4LA Scotland, UK.
[3]Department of Computer Science, University of York, Deramore Lane, York, YO10 5GH, UK.
[4]School of Computer Science, The University of Birmingham, Edgbaston, Birmingham B15 2TT, UK.

ABSTRACT

Search Based Software Engineering (SBSE) has proved to be a very effective way of optimising software engineering problems. Nevertheless, its full potential as a means of dynamic adaptivity remains under explored. This paper sets out the agenda for Dynamic Adaptive SBSE, in which the optimisation is embedded into deployed software to create self-optimising adaptive systems. Dynamic Adaptive SBSE will move the research agenda forward to encompass both software development processes and the software products they produce, addressing the long-standing, and as yet largely unsolved, grand challenge of self-adaptive systems.

Categories and Subject Descriptors

D.2 [**Software Engineering**]

General Terms

Search Based Software Engineering (SBSE), Evolution, Automatic Programming, Measurement, Testing

Keywords

SBSE, Search Based Optimization, Self-Adaptive Systems, Autonomic Computing

1. INTRODUCTION

Current software development practices achieve adaptivity at only a glacial pace, largely through enormous human engineering skill and effort. We force highly experienced engineers to waste their time and expertise adapting many tedious implementation details. Often, the resulting software is equally inflexible: users often find themselves relying on their innate human adaptivity to compensate with 'workarounds'. This has to change.

To address the twin goals of adaptivity and automation, we advocate a development of the Search Based Software Engineering (SBSE) agenda that we call 'Dynamic Adaptive Search Based Software Engineering'. We seek greater software engineering automation through the development of hyper heuristics for SBSE. At the same time we seek greater adaptivity through the use of dynamic optimisation; optimisation embedded into the deployed software to re-tune its performance parameters and even to replace large portions of code with automatically re-evolved code.

2. SBSE

Search Based Software Engineering (SBSE) is the name given to a field of research and practice in which computational search (as well as optimisation techniques more usually associated with Operations Research) are used to address problems in Software Engineering [39]. The SBSE approach seeks to optimise software engineering processes and products using generic, robust, flexible, scalable and insight-rich computational search. SBSE provides a mechanism for managed automation of software engineering activities.

SBSE has proved to be a widely applicable and successful approach, with many applications right across the full spectrum of activities in software engineering, from initial requirements, project planning, and cost estimation to regression testing and onward evolution. Few aspects of development and deployment of software systems have remained untouched by the SBSE research agenda.

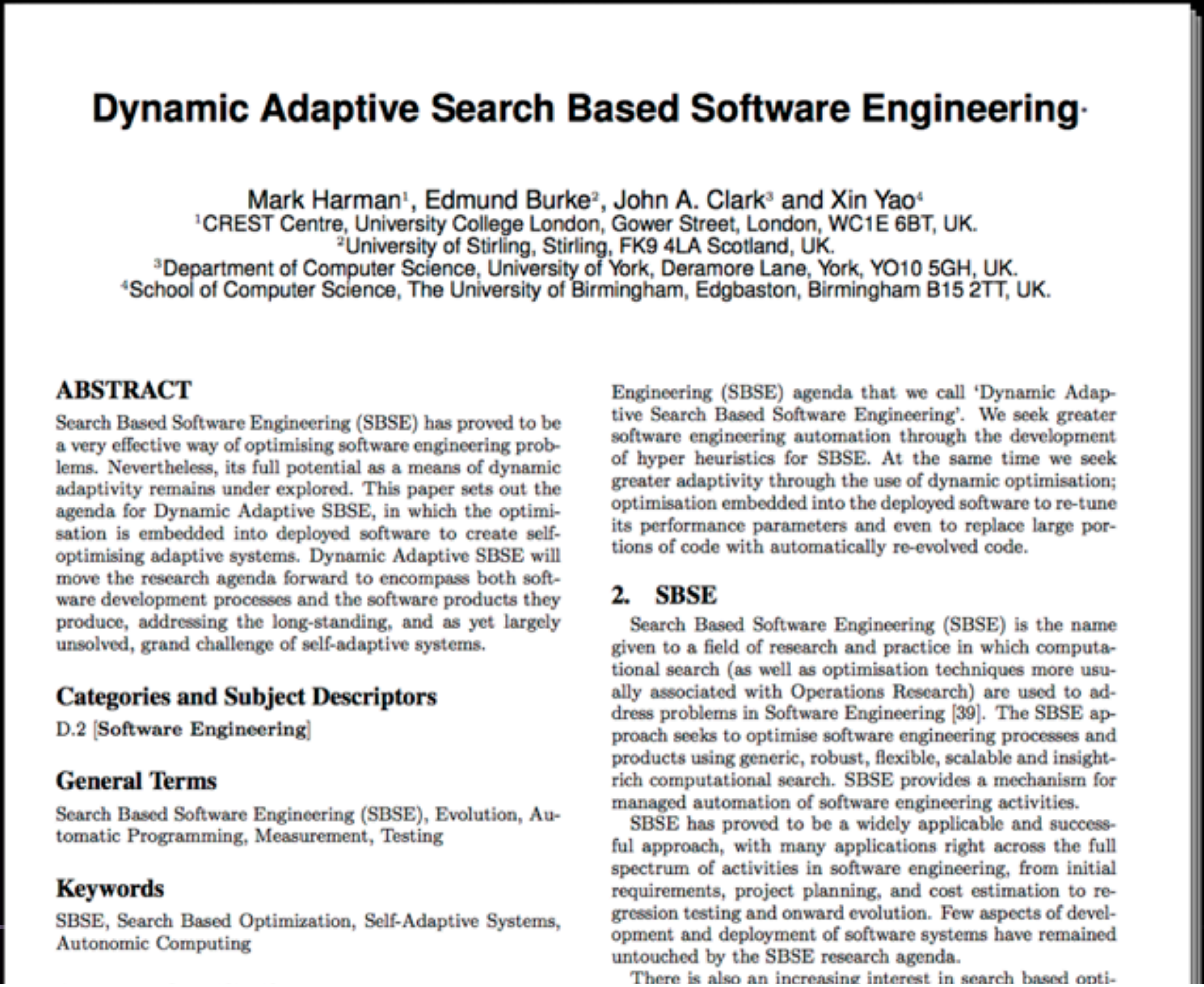
There is also an increasing interest in search based optimization from the industrial sector, as illustrated by work on testing involving Berner and Mattner and Daimler [49, 64], Ericsson [3], Google [69] and Microsoft [14, 50], and work on requirements analysis and optimisation involving Ericsson [70], Motorola [9] and NASA [20].

The increasing maturity of the field has led to a number of tools for SBSE applications, including AUSTIN (for C language test data generation, [49]), Bunch (for modularisation, [55]), Code-Imp (for automated refactoring, [56]), eTOC (for Java class testing, [63]), EvoSUITE (for Java test data generation, [26]), GenProg (for automated bug patching, [52]), MiLu (for higher order mutation testing, [46]), ReleasePlanner (for Requirements Optimisation, [58]), and SWAT (for PHP server-side test data generation [5]).

[*] This position paper is written to accompany Mark Harman's keynote talk at the 6th International Symposium on Empirical Software Engineering and Measurement (ESEM 12) in Lund, Sweden. It is joint work with Edmund Burke, John Clark and Xin Yao, funded by the EPSRC programme grant DAASE (EP/J017515/).

ESEM'12, September 19–20, 2012, Lund, Sweden.
ACM 978-1-4503-1056-7/12/09.

There is a paper that accompanied my keynote



Dynamic Adaptive Search Based Software Engineering

Mark Harman[1], Edmund Burke[2], John A. Clark[3] and Xin Yao[4]
[1]CREST Centre, University College London, Gower Street, London, WC1E 6BT, UK.
[2]University of Stirling, Stirling, FK9 4LA Scotland, UK.
[3]Department of Computer Science, University of York, Deramore Lane, York, YO10 5GH, UK.
[4]School of Computer Science, The University of Birmingham, Edgbaston, Birmingham B15 2TT, UK.

ABSTRACT

Search Based Software Engineering (SBSE) has proved to be a very effective way of optimising software engineering problems. Nevertheless, its full potential as a means of dynamic adaptivity remains under explored. This paper sets out the agenda for Dynamic Adaptive SBSE, in which the optimisation is embedded into deployed software to create self-optimising adaptive systems. Dynamic Adaptive SBSE will move the research agenda forward to encompass both software development processes and the software products they produce, addressing the long-standing, and as yet largely unsolved, grand challenge of self-adaptive systems.

Categories and Subject Descriptors

D.2 [**Software Engineering**]

General Terms

Search Based Software Engineering (SBSE), Evolution, Automatic Programming, Measurement, Testing

Keywords

SBSE, Search Based Optimization, Self-Adaptive Systems, Autonomic Computing

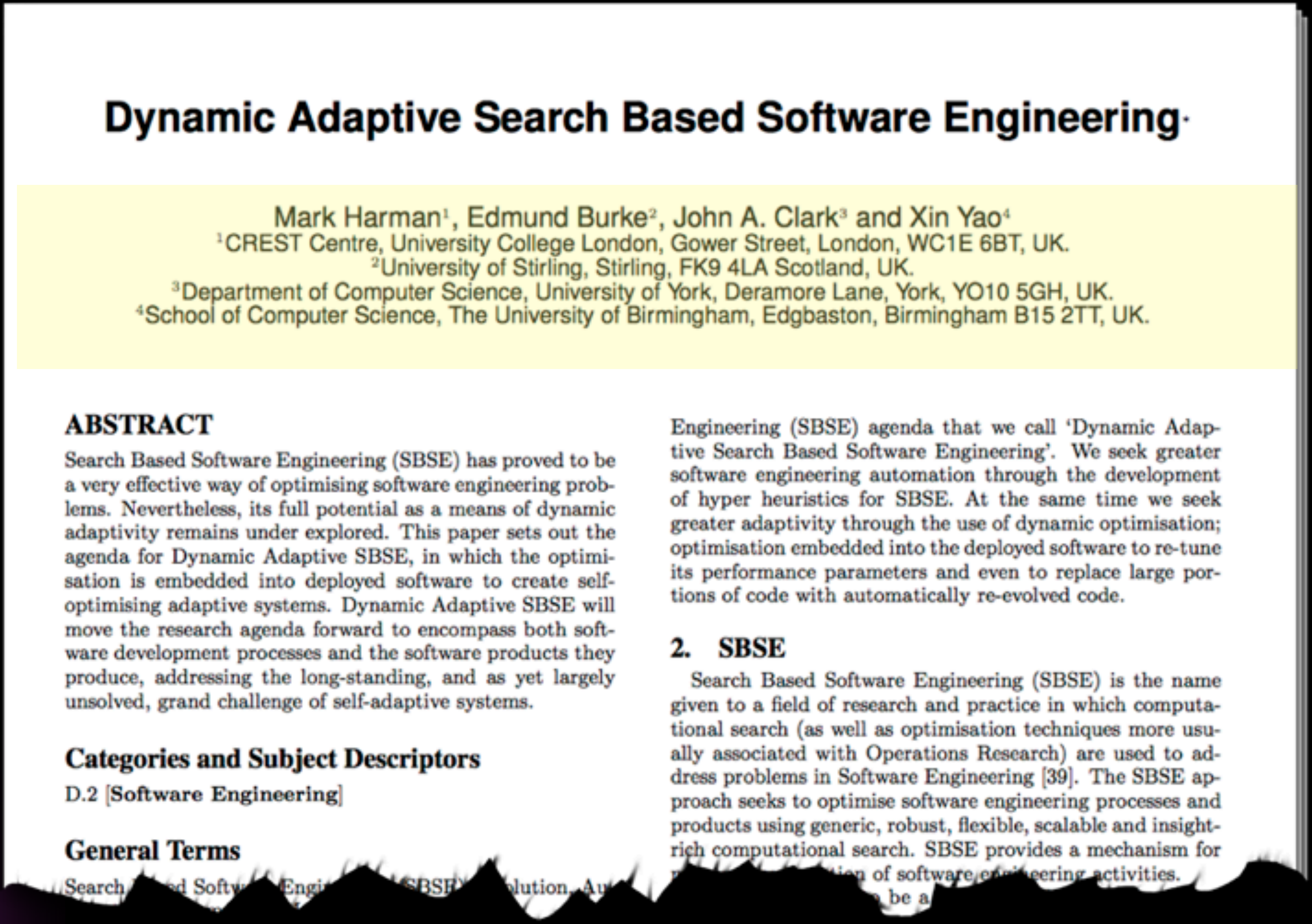
Engineering (SBSE) agenda that we call 'Dynamic Adaptive Search Based Software Engineering'. We seek greater software engineering automation through the development of hyper heuristics for SBSE. At the same time we seek greater adaptivity through the use of dynamic optimisation; optimisation embedded into the deployed software to re-tune its performance parameters and even to replace large portions of code with automatically re-evolved code.

2. SBSE

Search Based Software Engineering (SBSE) is the name given to a field of research and practice in which computational search (as well as optimisation techniques more usually associated with Operations Research) are used to address problems in Software Engineering [39]. The SBSE approach seeks to optimise software engineering processes and products using generic, robust, flexible, scalable and insight-rich computational search. SBSE provides a mechanism for managed automation of software engineering activities.

SBSE has proved to be a widely applicable and successful approach, with many applications right across the full spectrum of activities in software engineering, from initial requirements, project planning, and cost estimation to regression testing and onward evolution. Few aspects of development and deployment of software systems have remained untouched by the SBSE research agenda.

There is also an increasing interest in search based opti-



Dynamic Adaptive Search Based Software Engineering

Mark Harman[1], Edmund Burke[2], John A. Clark[3] and Xin Yao[4]
[1]CREST Centre, University College London, Gower Street, London, WC1E 6BT, UK.
[2]University of Stirling, Stirling, FK9 4LA Scotland, UK.
[3]Department of Computer Science, University of York, Deramore Lane, York, YO10 5GH, UK.
[4]School of Computer Science, The University of Birmingham, Edgbaston, Birmingham B15 2TT, UK.

ABSTRACT

Search Based Software Engineering (SBSE) has proved to be a very effective way of optimising software engineering problems. Nevertheless, its full potential as a means of dynamic adaptivity remains under explored. This paper sets out the agenda for Dynamic Adaptive SBSE, in which the optimisation is embedded into deployed software to create self-optimising adaptive systems. Dynamic Adaptive SBSE will move the research agenda forward to encompass both software development processes and the software products they produce, addressing the long-standing, and as yet largely unsolved, grand challenge of self-adaptive systems.

Categories and Subject Descriptors

D.2 [Software Engineering]

General Terms

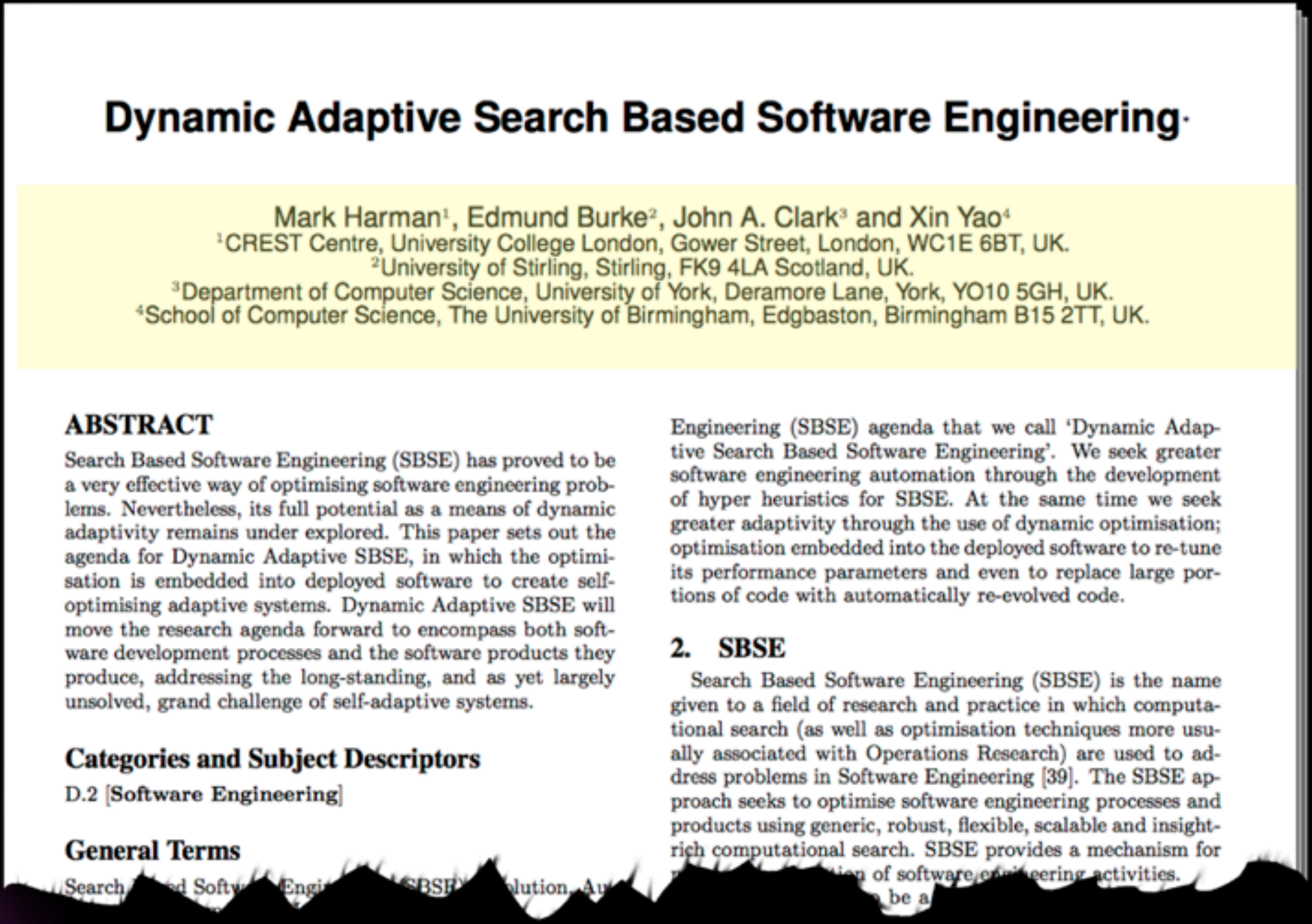
Search Softw Engi SBSE lution, Au

Engineering (SBSE) agenda that we call 'Dynamic Adaptive Search Based Software Engineering'. We seek greater software engineering automation through the development of hyper heuristics for SBSE. At the same time we seek greater adaptivity through the use of dynamic optimisation; optimisation embedded into the deployed software to re-tune its performance parameters and even to replace large portions of code with automatically re-evolved code.

2. SBSE

Search Based Software Engineering (SBSE) is the name given to a field of research and practice in which computational search (as well as optimisation techniques more usually associated with Operations Research) are used to address problems in Software Engineering [39]. The SBSE approach seeks to optimise software engineering processes and products using generic, robust, flexible, scalable and insight-rich computational search. SBSE provides a mechanism for of software engineering activities.

be a



Dynamic Adaptive Search Based Software Engineering

Mark Harman[1], Edmund Burke[2], John A. Clark[3] and Xin Yao[4]
[1]CREST Centre, University College London, Gower Street, London, WC1E 6BT, UK.
[2]University of Stirling, Stirling, FK9 4LA Scotland, UK.
[3]Department of Computer Science, University of York, Deramore Lane, York, YO10 5GH, UK.
[4]School of Computer Science, The University of Birmingham, Edgbaston, Birmingham B15 2TT, UK.

ABSTRACT

Search Based Software Engineering (SBSE) has proved to be a very effective way of optimising software engineering problems. Nevertheless, its full potential as a means of dynamic adaptivity remains under explored. This paper sets out the agenda for Dynamic Adaptive SBSE, in which the optimisation is embedded into deployed software to create self-optimising adaptive systems. Dynamic Adaptive SBSE will move the research agenda forward to encompass both software development processes and the software products they produce, addressing the long-standing, and as yet largely unsolved, grand challenge of self-adaptive systems.

Categories and Subject Descriptors

D.2 [**Software Engineering**]

General Terms

Search d Softw Engi SBSE lution, Au

Engineering (SBSE) agenda that we call 'Dynamic Adaptive Search Based Software Engineering'. We seek greater software engineering automation through the development of hyper heuristics for SBSE. At the same time we seek greater adaptivity through the use of dynamic optimisation; optimisation embedded into the deployed software to re-tune its performance parameters and even to replace large portions of code with automatically re-evolved code.

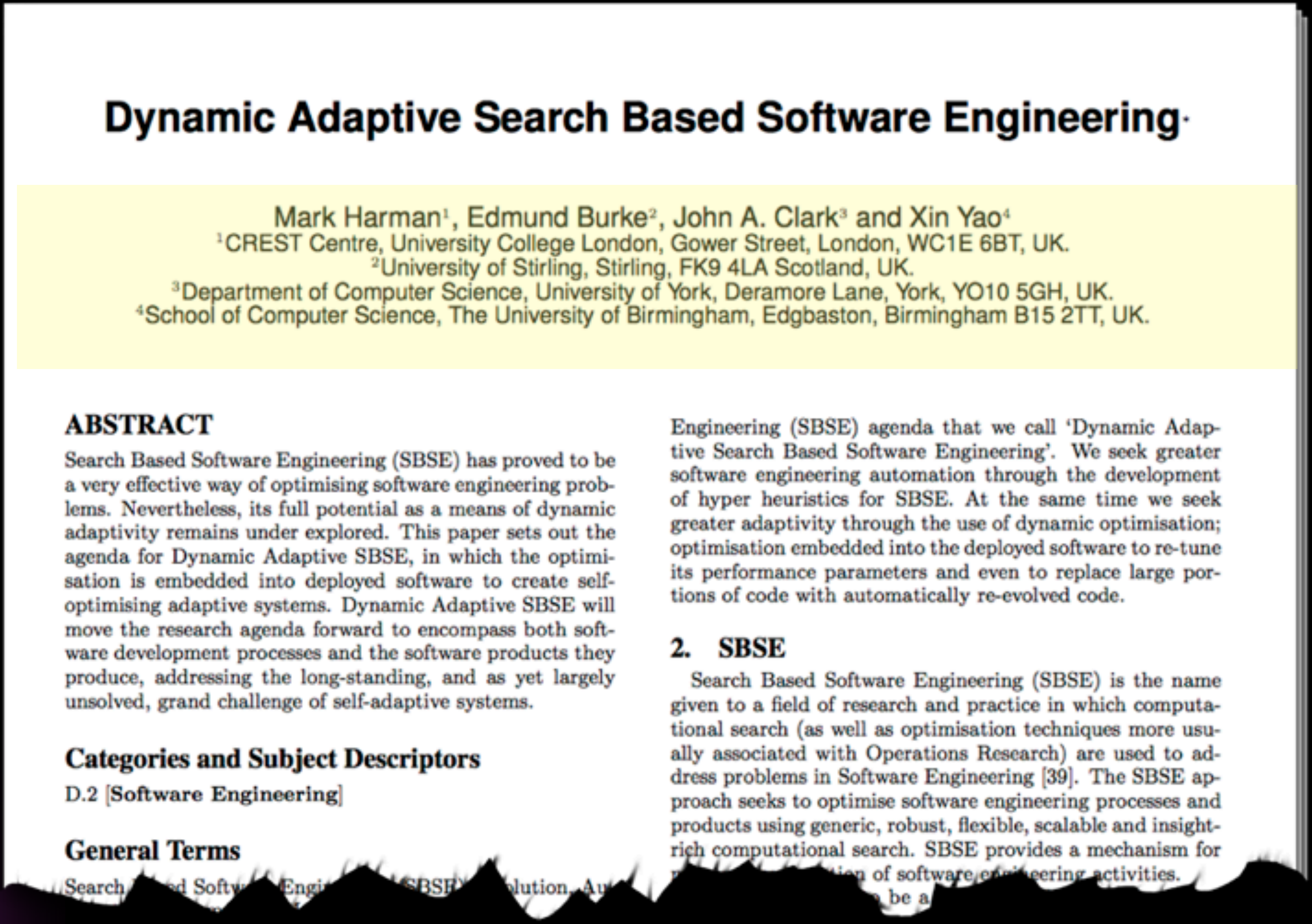
2. SBSE

Search Based Software Engineering (SBSE) is the name given to a field of research and practice in which computational search (as well as optimisation techniques more usually associated with Operations Research) are used to address problems in Software Engineering [39]. The SBSE approach seeks to optimise software engineering processes and products using generic, robust, flexible, scalable and insight-rich computational search. SBSE provides a mechanism for

n of software engineering activities.

be a

DAASE



Dynamic Adaptive Search Based Software Engineering*

Mark Harman[1], Edmund Burke[2], John A. Clark[3] and Xin Yao[4]
[1]CREST Centre, University College London, Gower Street, London, WC1E 6BT, UK.
[2]University of Stirling, Stirling, FK9 4LA Scotland, UK.
[3]Department of Computer Science, University of York, Deramore Lane, York, YO10 5GH, UK.
[4]School of Computer Science, The University of Birmingham, Edgbaston, Birmingham B15 2TT, UK.

ABSTRACT

Search Based Software Engineering (SBSE) has proved to be a very effective way of optimising software engineering problems. Nevertheless, its full potential as a means of dynamic adaptivity remains under explored. This paper sets out the agenda for Dynamic Adaptive SBSE, in which the optimisation is embedded into deployed software to create self-optimising adaptive systems. Dynamic Adaptive SBSE will move the research agenda forward to encompass both software development processes and the software products they produce, addressing the long-standing, and as yet largely unsolved, grand challenge of self-adaptive systems.

Categories and Subject Descriptors

D.2 [**Software Engineering**]

General Terms

Search d Softw Engi SBSE lution, Au

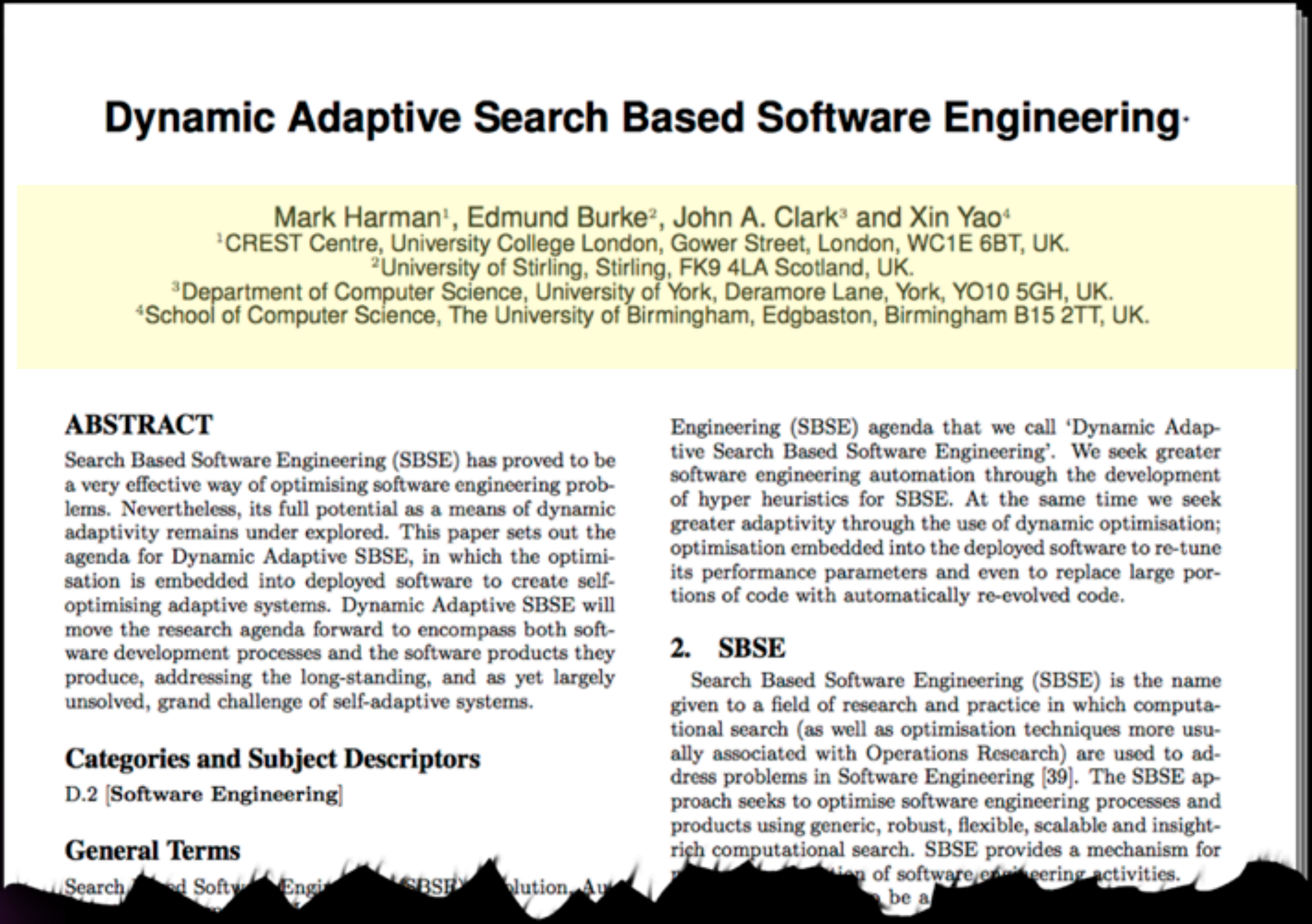
Engineering (SBSE) agenda that we call 'Dynamic Adaptive Search Based Software Engineering'. We seek greater software engineering automation through the development of hyper heuristics for SBSE. At the same time we seek greater adaptivity through the use of dynamic optimisation; optimisation embedded into the deployed software to re-tune its performance parameters and even to replace large portions of code with automatically re-evolved code.

2. SBSE

Search Based Software Engineering (SBSE) is the name given to a field of research and practice in which computational search (as well as optimisation techniques more usually associated with Operations Research) are used to address problems in Software Engineering [39]. The SBSE approach seeks to optimise software engineering processes and products using generic, robust, flexible, scalable and insight-rich computational search. SBSE provides a mechanism for

of software engineering activities.

be a



Dynamic Adaptive Search Based Software Engineering

Mark Harman[1], Edmund Burke[2], John A. Clark[3] and Xin Yao[4]
[1]CREST Centre, University College London, Gower Street, London, WC1E 6BT, UK.
[2]University of Stirling, Stirling, FK9 4LA Scotland, UK.
[3]Department of Computer Science, University of York, Deramore Lane, York, YO10 5GH, UK.
[4]School of Computer Science, The University of Birmingham, Edgbaston, Birmingham B15 2TT, UK.

ABSTRACT

Search Based Software Engineering (SBSE) has proved to be a very effective way of optimising software engineering problems. Nevertheless, its full potential as a means of dynamic adaptivity remains under explored. This paper sets out the agenda for Dynamic Adaptive SBSE, in which the optimisation is embedded into deployed software to create self-optimising adaptive systems. Dynamic Adaptive SBSE will move the research agenda forward to encompass both software development processes and the software products they produce, addressing the long-standing, and as yet largely unsolved, grand challenge of self-adaptive systems.

Categories and Subject Descriptors

D.2 [Software Engineering]

General Terms

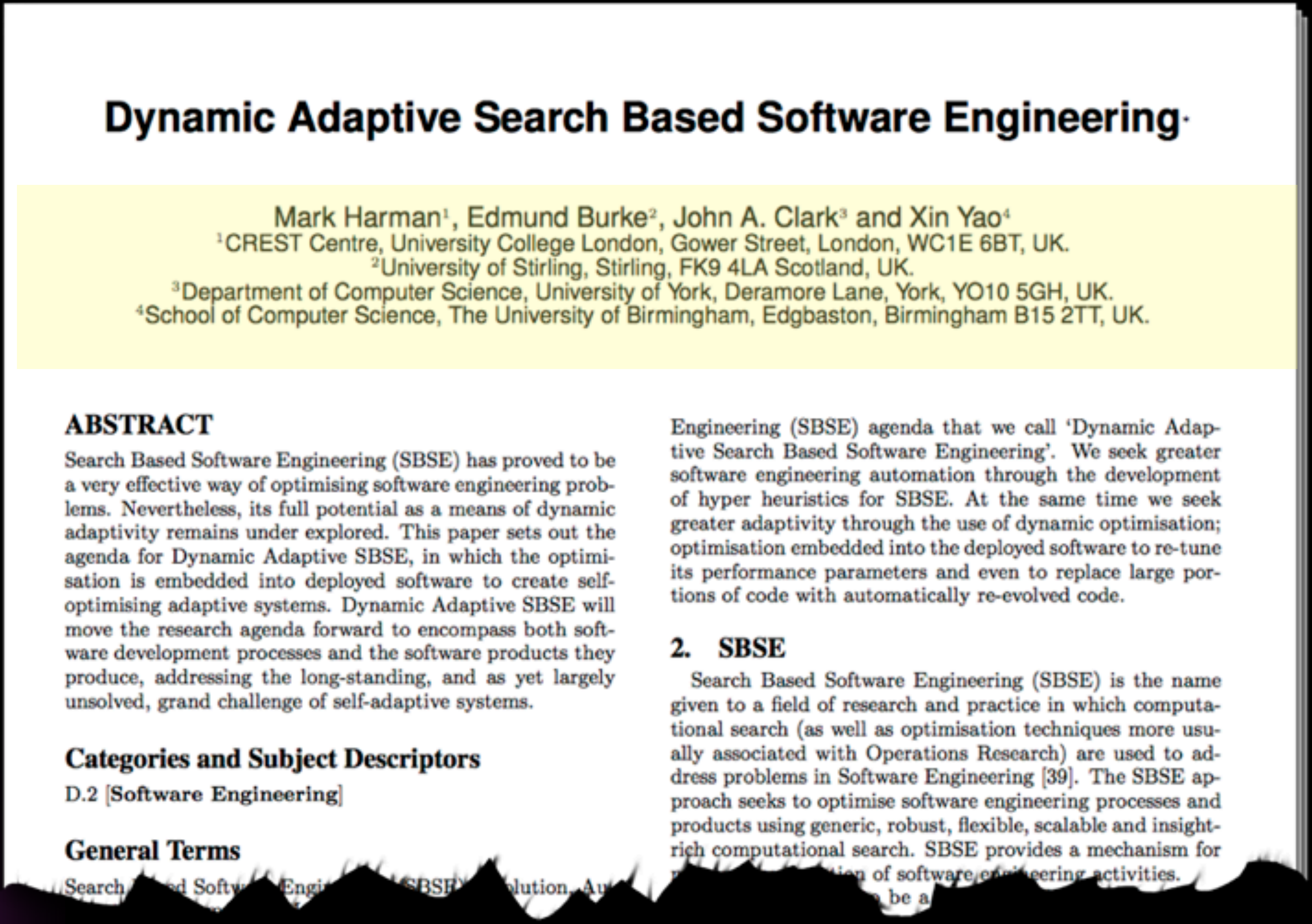
Search ed Softw Engi SBSE lution, Au

Engineering (SBSE) agenda that we call 'Dynamic Adaptive Search Based Software Engineering'. We seek greater software engineering automation through the development of hyper heuristics for SBSE. At the same time we seek greater adaptivity through the use of dynamic optimisation; optimisation embedded into the deployed software to re-tune its performance parameters and even to replace large portions of code with automatically re-evolved code.

2. SBSE

Search Based Software Engineering (SBSE) is the name given to a field of research and practice in which computational search (as well as optimisation techniques more usually associated with Operations Research) are used to address problems in Software Engineering [39]. The SBSE approach seeks to optimise software engineering processes and products using generic, robust, flexible, scalable and insight-rich computational search. SBSE provides a mechanism for of software eering activities. be a

DAASE



Dynamic Adaptive Search Based Software Engineering·

Mark Harman[1], Edmund Burke[2], John A. Clark[3] and Xin Yao[4]
[1]CREST Centre, University College London, Gower Street, London, WC1E 6BT, UK.
[2]University of Stirling, Stirling, FK9 4LA Scotland, UK.
[3]Department of Computer Science, University of York, Deramore Lane, York, YO10 5GH, UK.
[4]School of Computer Science, The University of Birmingham, Edgbaston, Birmingham B15 2TT, UK.

ABSTRACT

Search Based Software Engineering (SBSE) has proved to be a very effective way of optimising software engineering problems. Nevertheless, its full potential as a means of dynamic adaptivity remains under explored. This paper sets out the agenda for Dynamic Adaptive SBSE, in which the optimisation is embedded into deployed software to create self-optimising adaptive systems. Dynamic Adaptive SBSE will move the research agenda forward to encompass both software development processes and the software products they produce, addressing the long-standing, and as yet largely unsolved, grand challenge of self-adaptive systems.

Categories and Subject Descriptors

D.2 [**Software Engineering**]

General Terms

Search [illegible]d Softw[illegible] Engi[illegible] (SBSE[illegible]olution, Au

Engineering (SBSE) agenda that we call 'Dynamic Adaptive Search Based Software Engineering'. We seek greater software engineering automation through the development of hyper heuristics for SBSE. At the same time we seek greater adaptivity through the use of dynamic optimisation; optimisation embedded into the deployed software to re-tune its performance parameters and even to replace large portions of code with automatically re-evolved code.

2. SBSE

Search Based Software Engineering (SBSE) is the name given to a field of research and practice in which computational search (as well as optimisation techniques more usually associated with Operations Research) are used to address problems in Software Engineering [39]. The SBSE approach seeks to optimise software engineering processes and products using generic, robust, flexible, scalable and insight-rich computational search. SBSE provides a mechanism for [illegible]ion of software engineering activities. [illegible] be a

DAASE

Experimental

Empirical

DAASE

Experimental vs. Empirical

discussed in the paper

Experimental vs. Empirical

discussed in the paper

... but no time to discuss this today ...

Dynamic Adaptive SBSE

Compile SBSE into deployed Software

The project

DAASE:

Dynamic Adaptive Automated Software Engineering

£12m project (2012-2018)

PhD studentships

RA positions

The project

DAASE:

Dynamic Adaptive Automated Software Engineering

£6.8m project (2012-2018)

PhD studentships

RA positions

The project

DAASE:

Dynamic Adaptive Automated Software Engineering

£6.8m project (2012-2018)

PhD studentships

RA positions

EPSRC Grant

DTC

Programme

UCL

UNIVERSITY OF STIRLING

UNIVERSITY OF BIRMINGHAM

THE UNIVERSITY of York

EPSRC Grant

DTC

Programme

UCL UNIVERSITY OF STIRLING UNIVERSITY OF BIRMINGHAM THE UNIVERSITY of York

EPSRC Grant

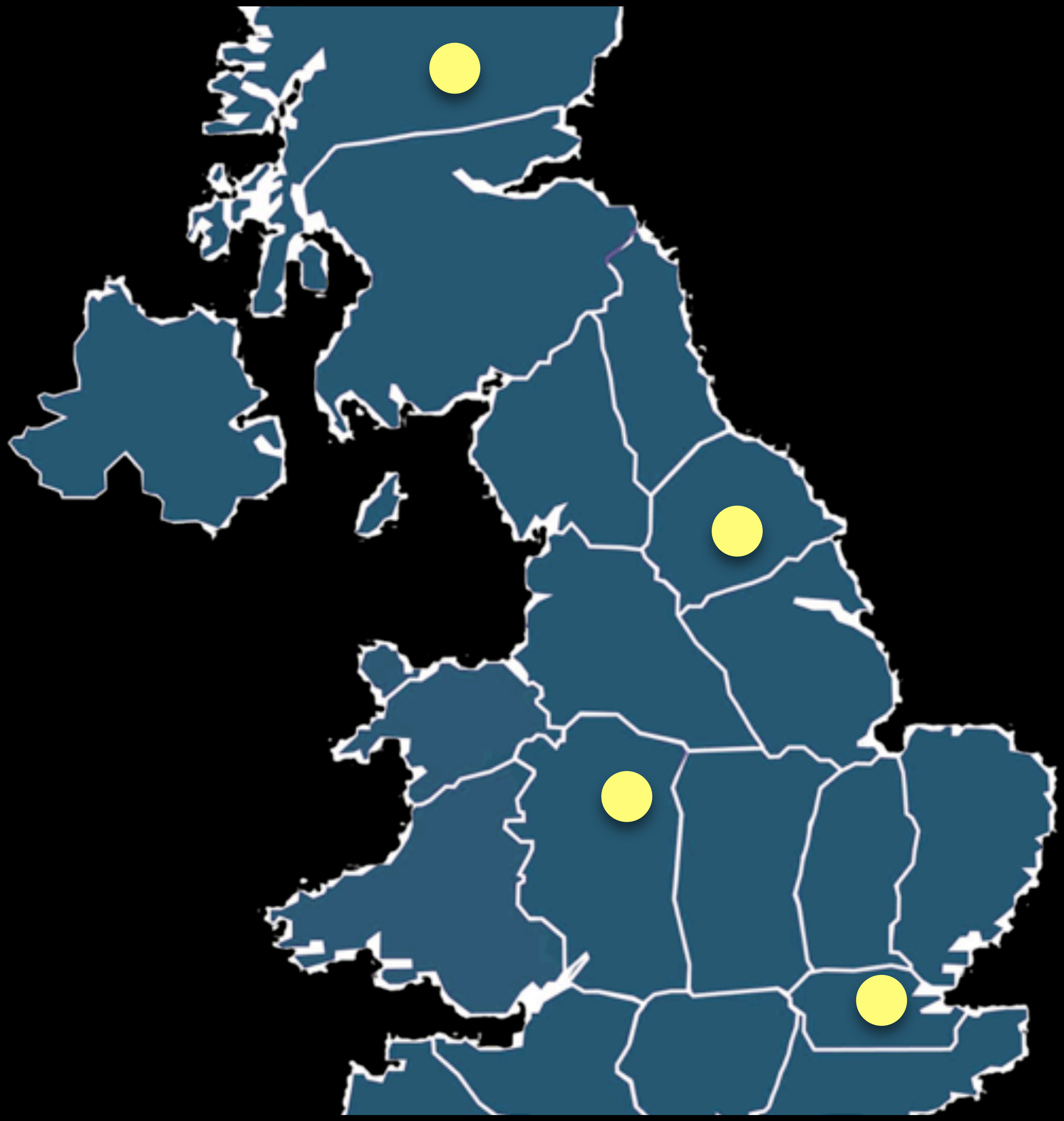
DTC

Programme

EPSRC Grant

DTC

Programme



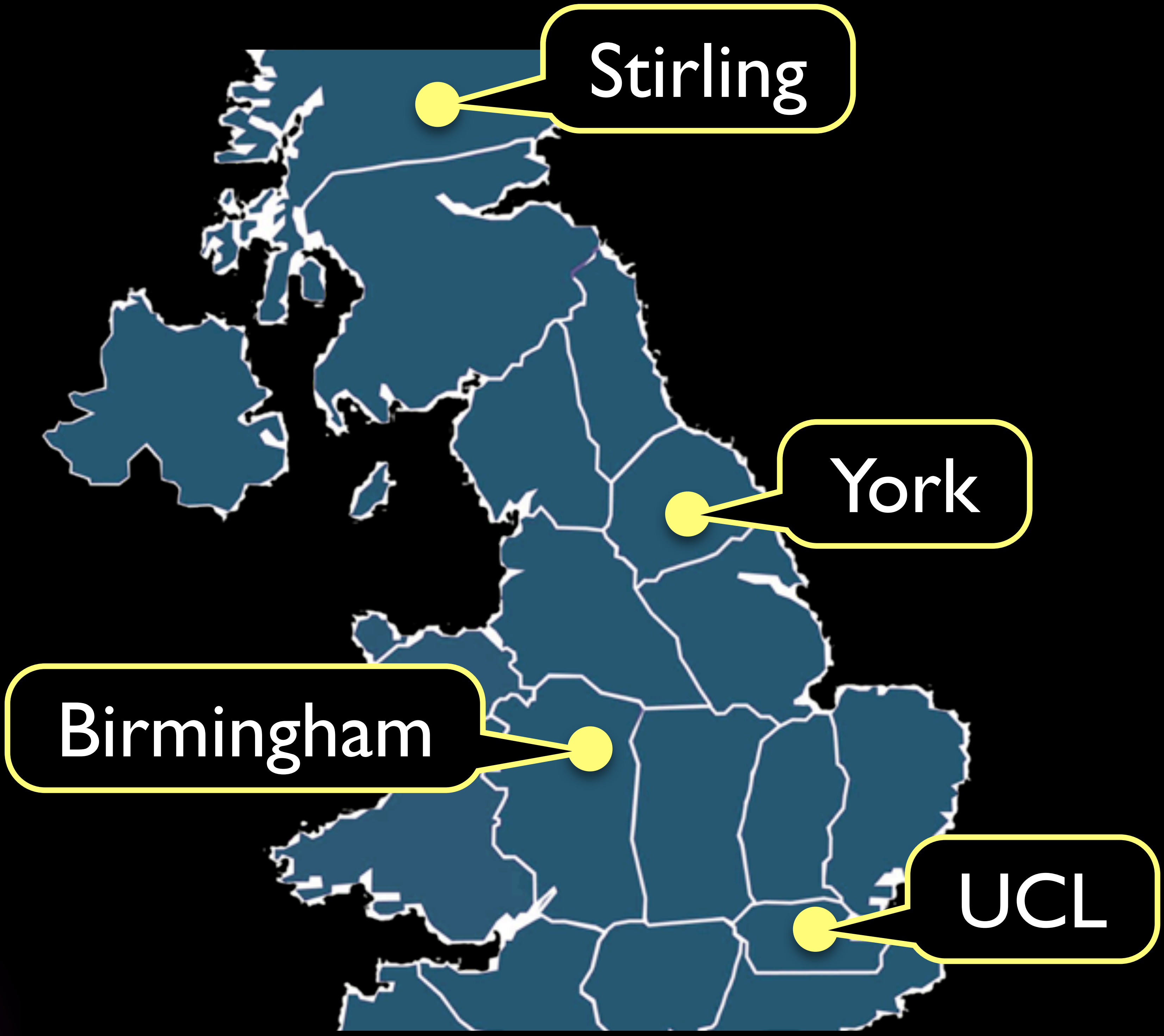
DAASE

Mark Harman, CREST

EPSRC Grant

DTC

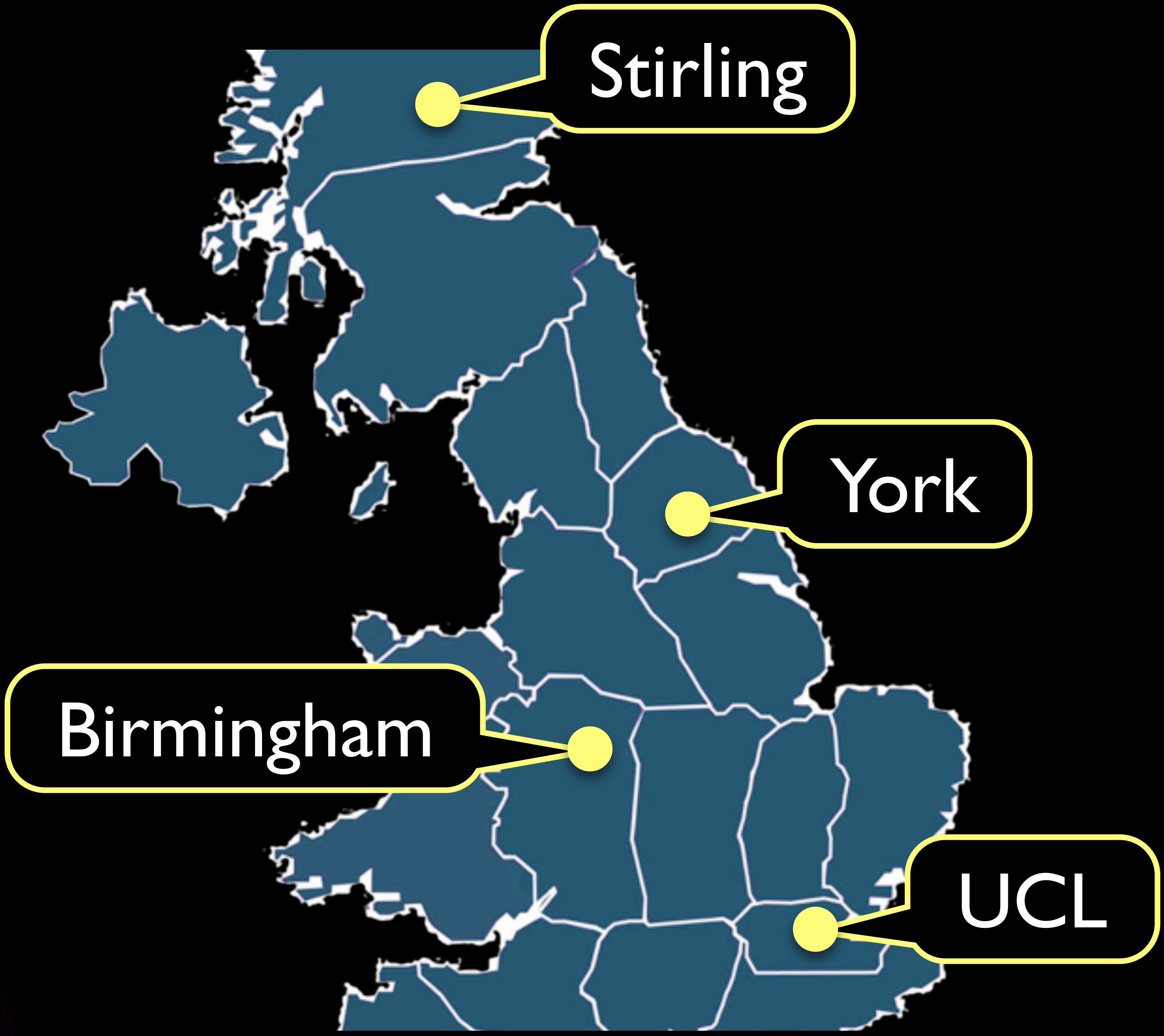
Programme



DAASE

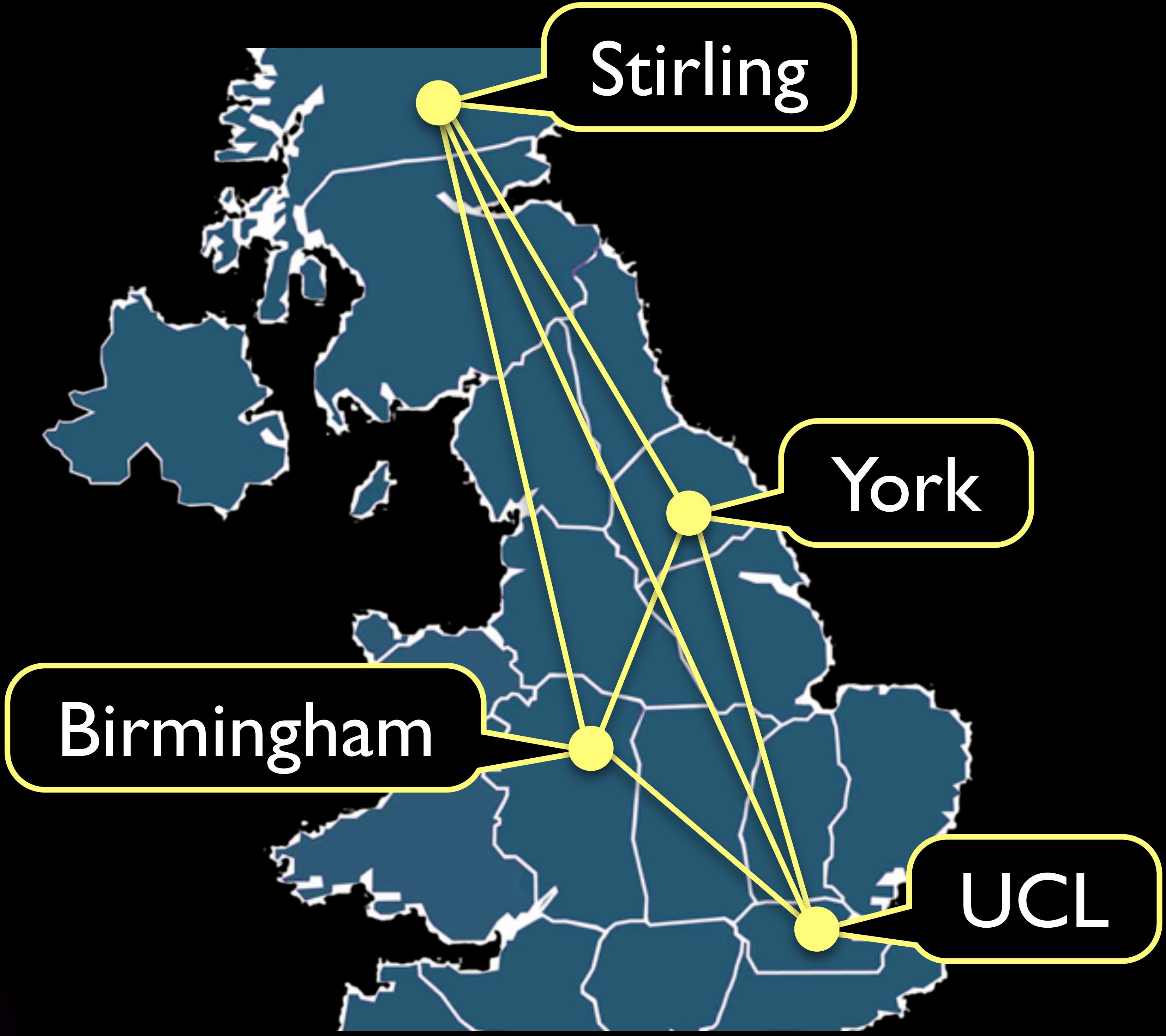
EPSRC Grant

DTC Programme



EPSRC
Grant

DTC Programme



EPSRC Grant

DTC

Programme

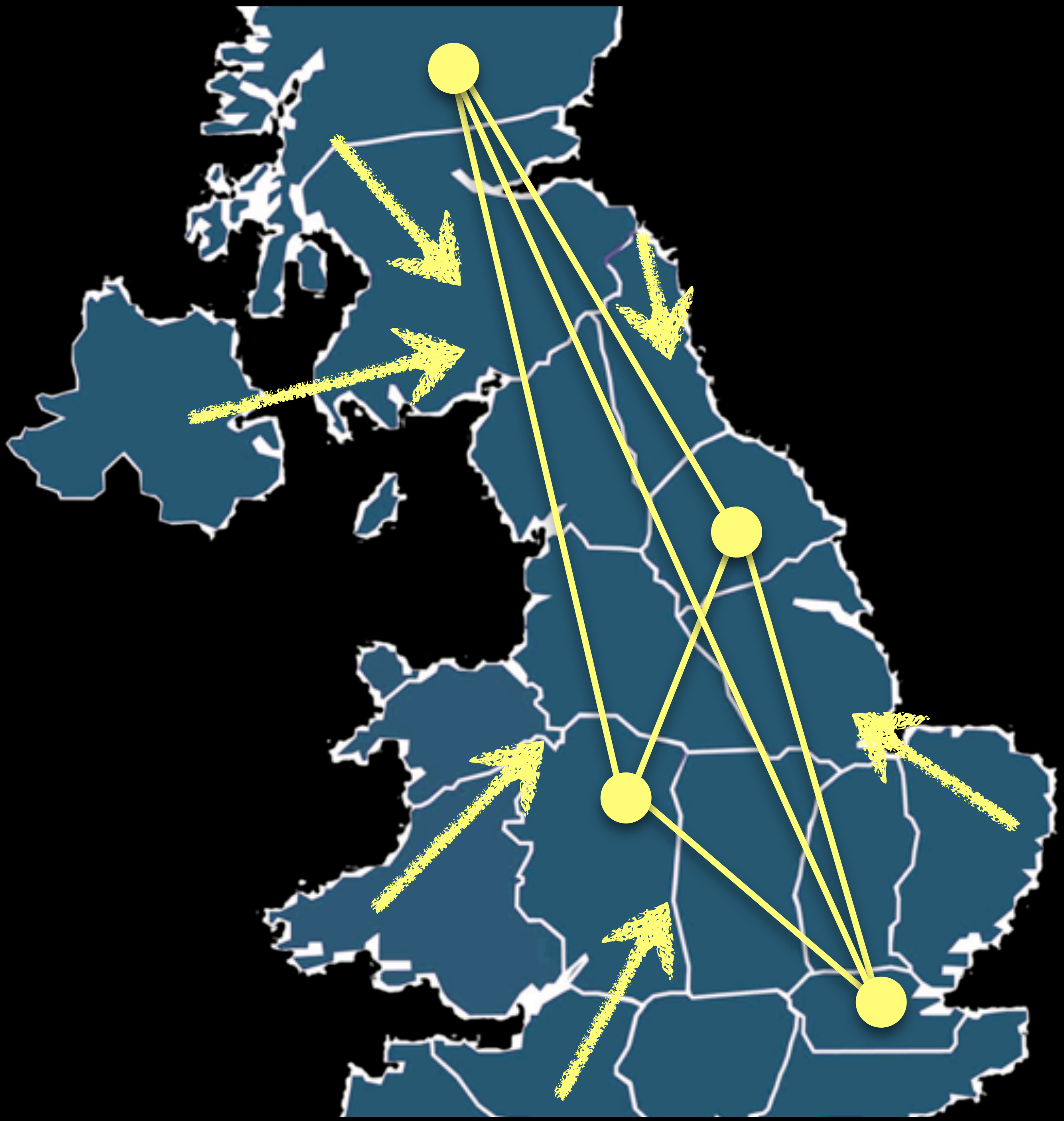
DAASE

Mark Harman, CREST

EPSRC Grant

DTC

Programme



DAASE

Mark Harman, CREST

EPSRC Grant

DTC

Programme

DAASE

Mark Harman, CREST

EPSRC Grant

DTC

Programme

DAASE

Mark Harman, CREST

Saturday, 16 February 13

EPSRC Grant

DTC

Programme

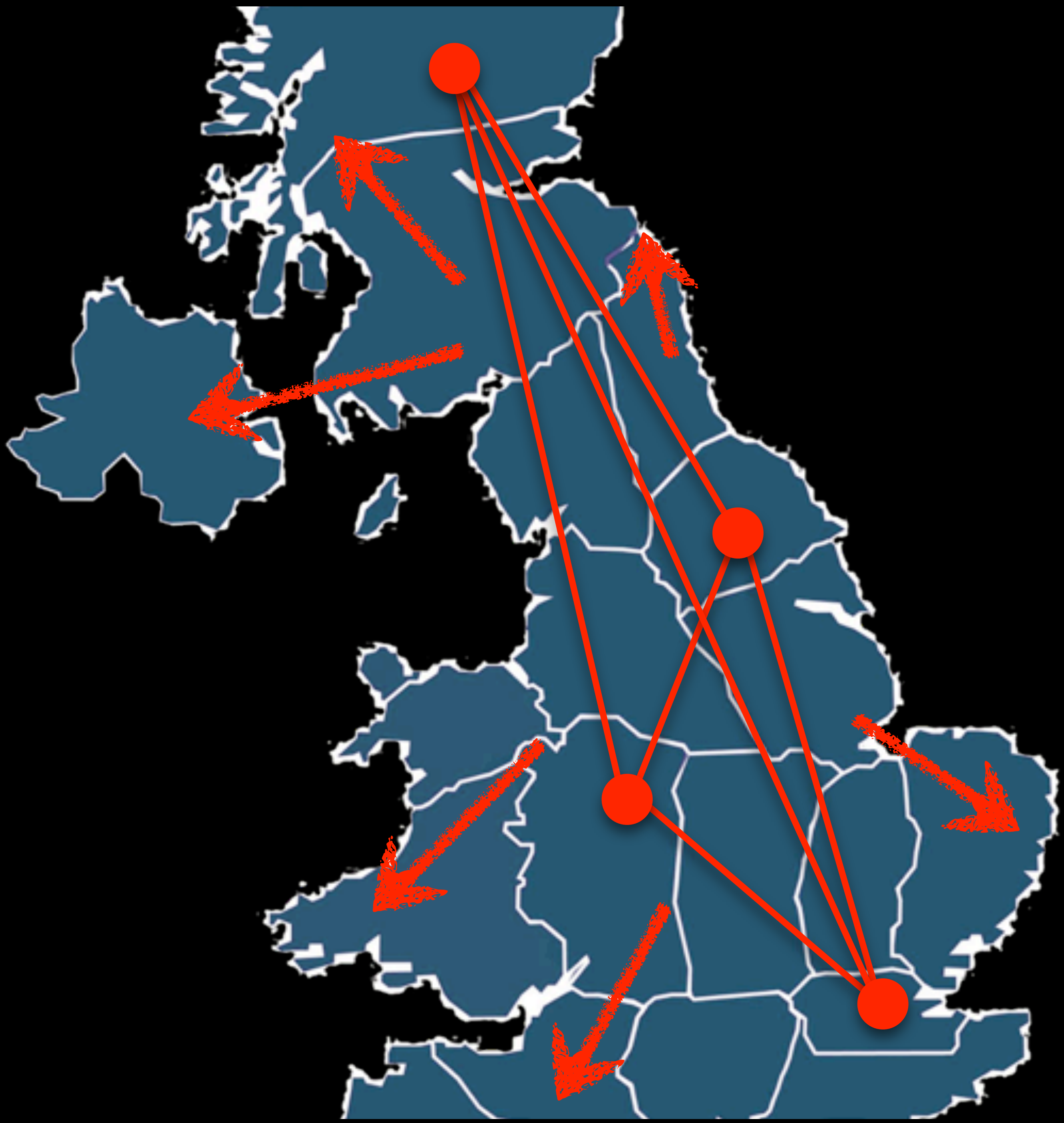
DAASE

Mark Harman, CREST

EPSRC Grant

DTC

Programme



DAASE

Mark Harman, CREST

EPSRC Grant

DTC

Programme

DAASE

Mark Harman, CREST

Saturday, 16 February 13

EPSRC Grant

DTC

Programme

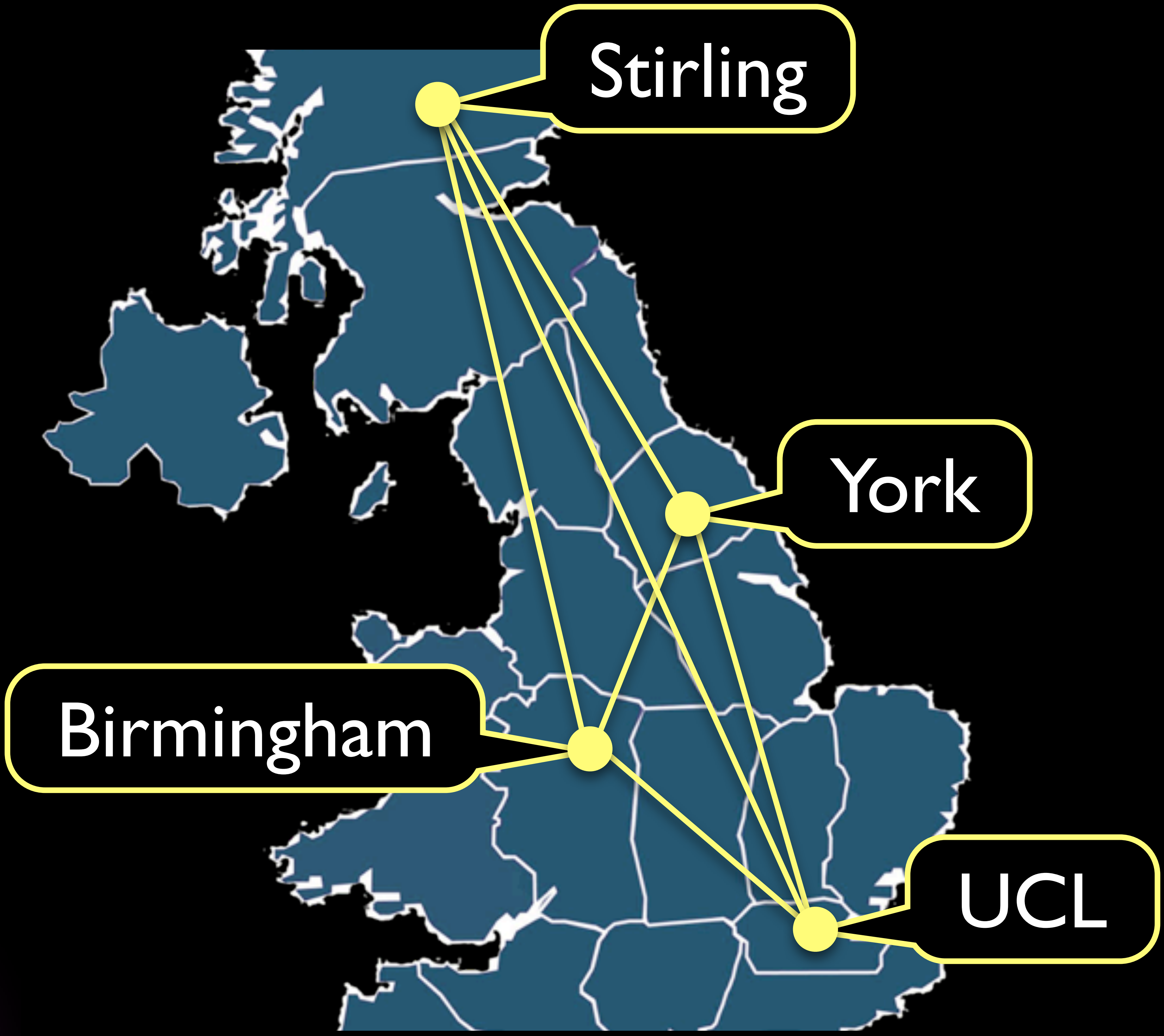
DAASE

Mark Harman, CREST

EPSRC
Grant

DTC

Programme



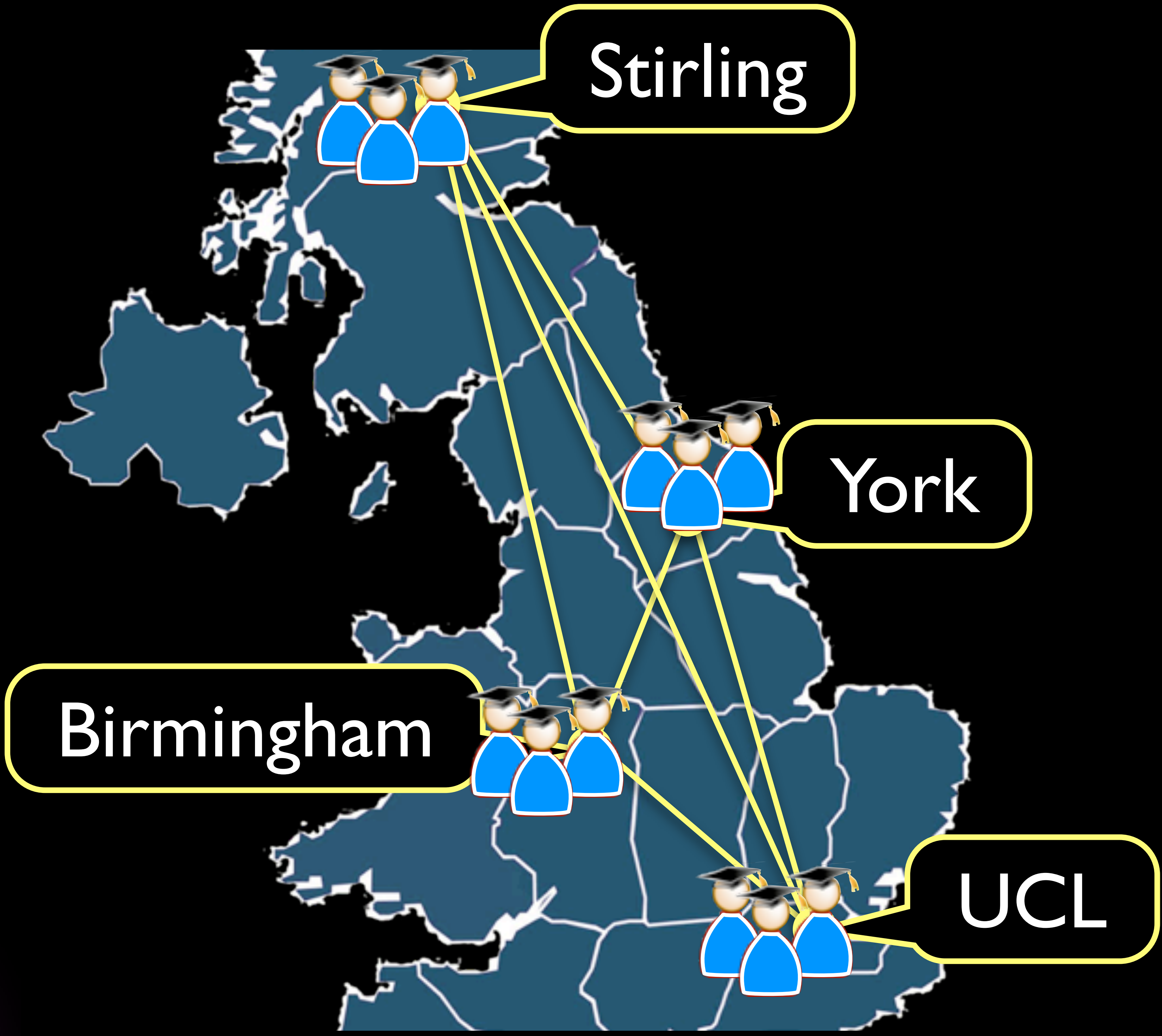
DAASE

Mark Harman, CREST

EPSRC Grant

DTC

Programme



DAASE

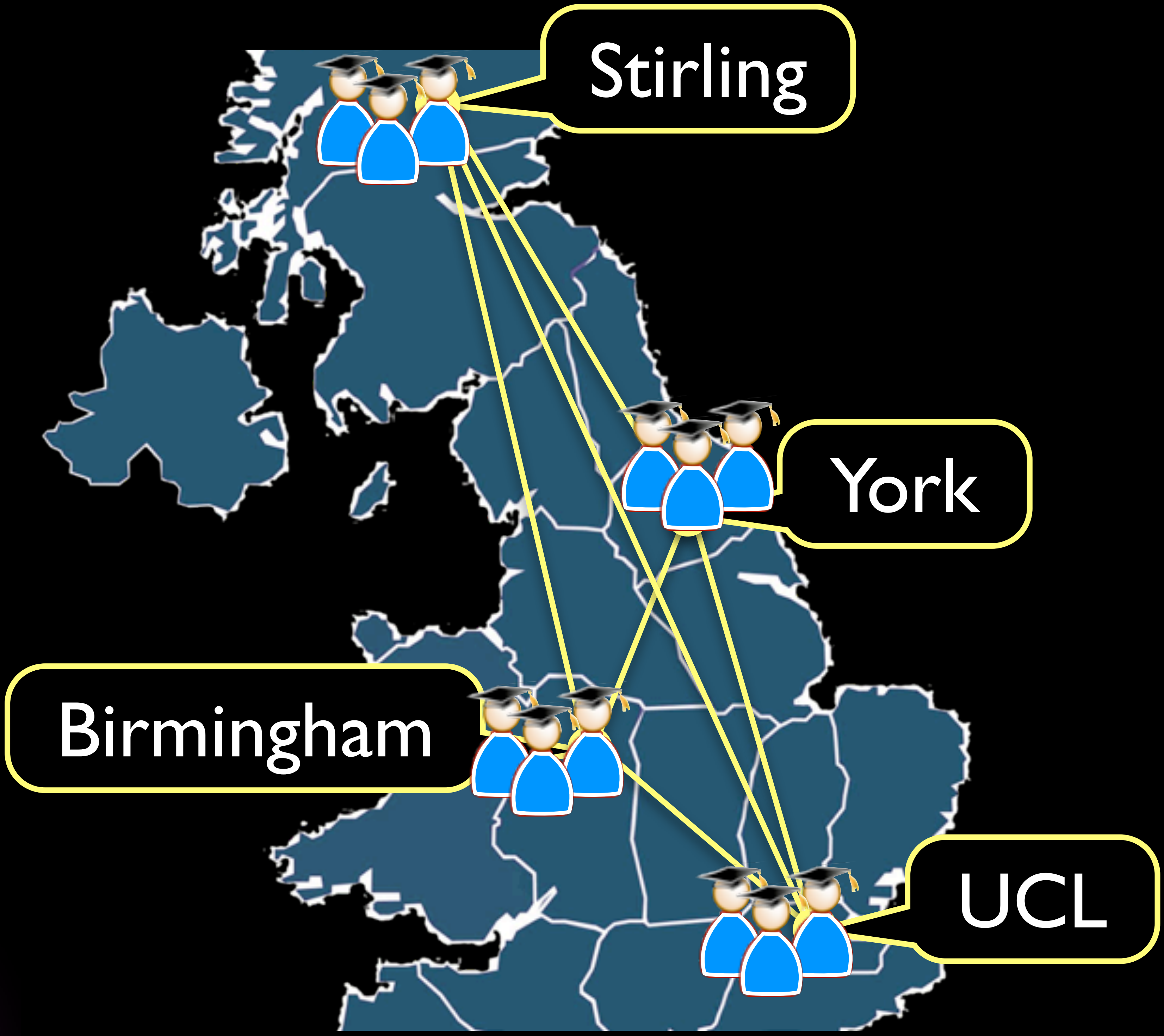
Mark Harman, CREST

Saturday, 16 February 13

EPSRC Grant

DTC

Programme

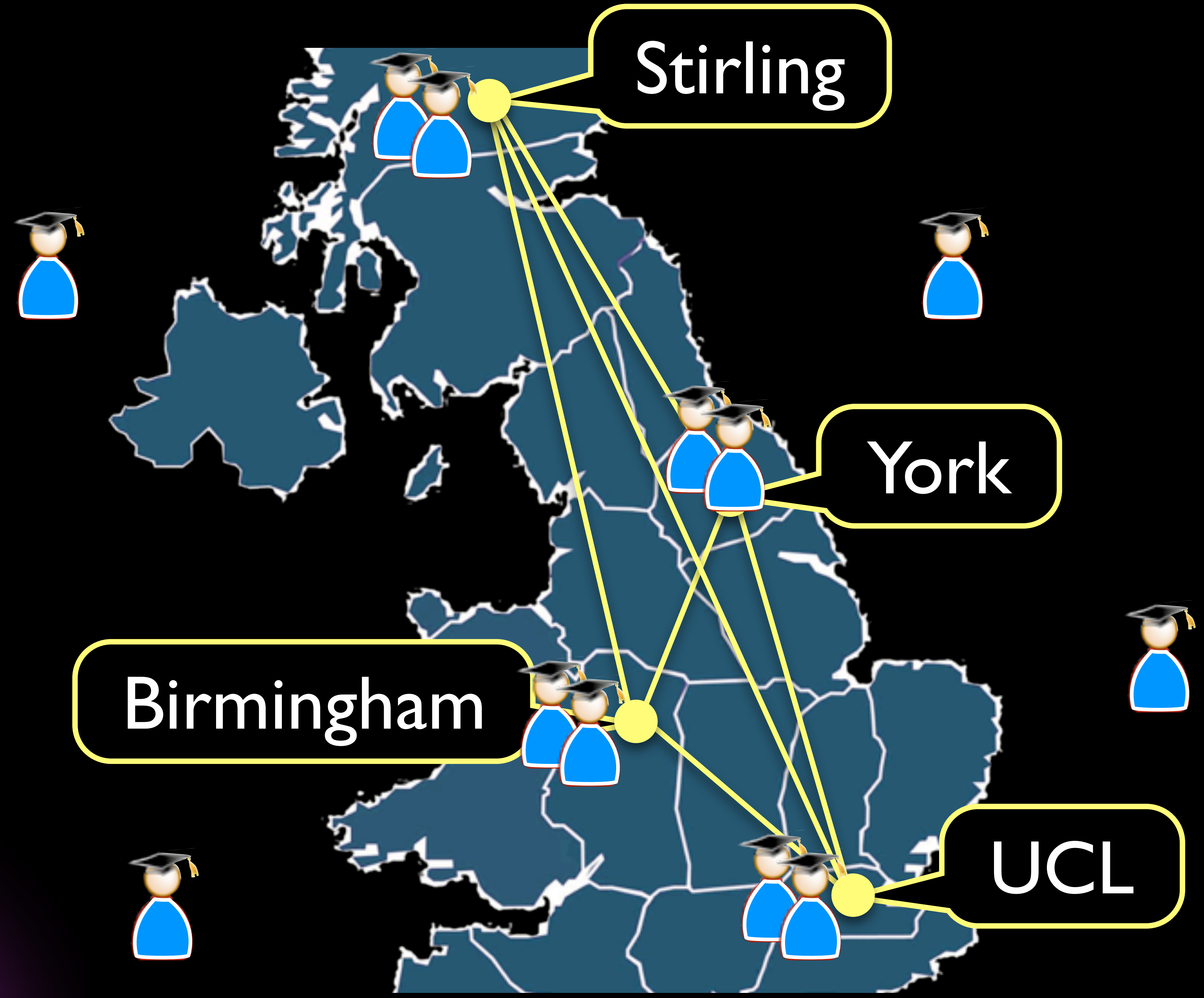


DAASE

EPSRC Grant

DTC

Programme



DAASE

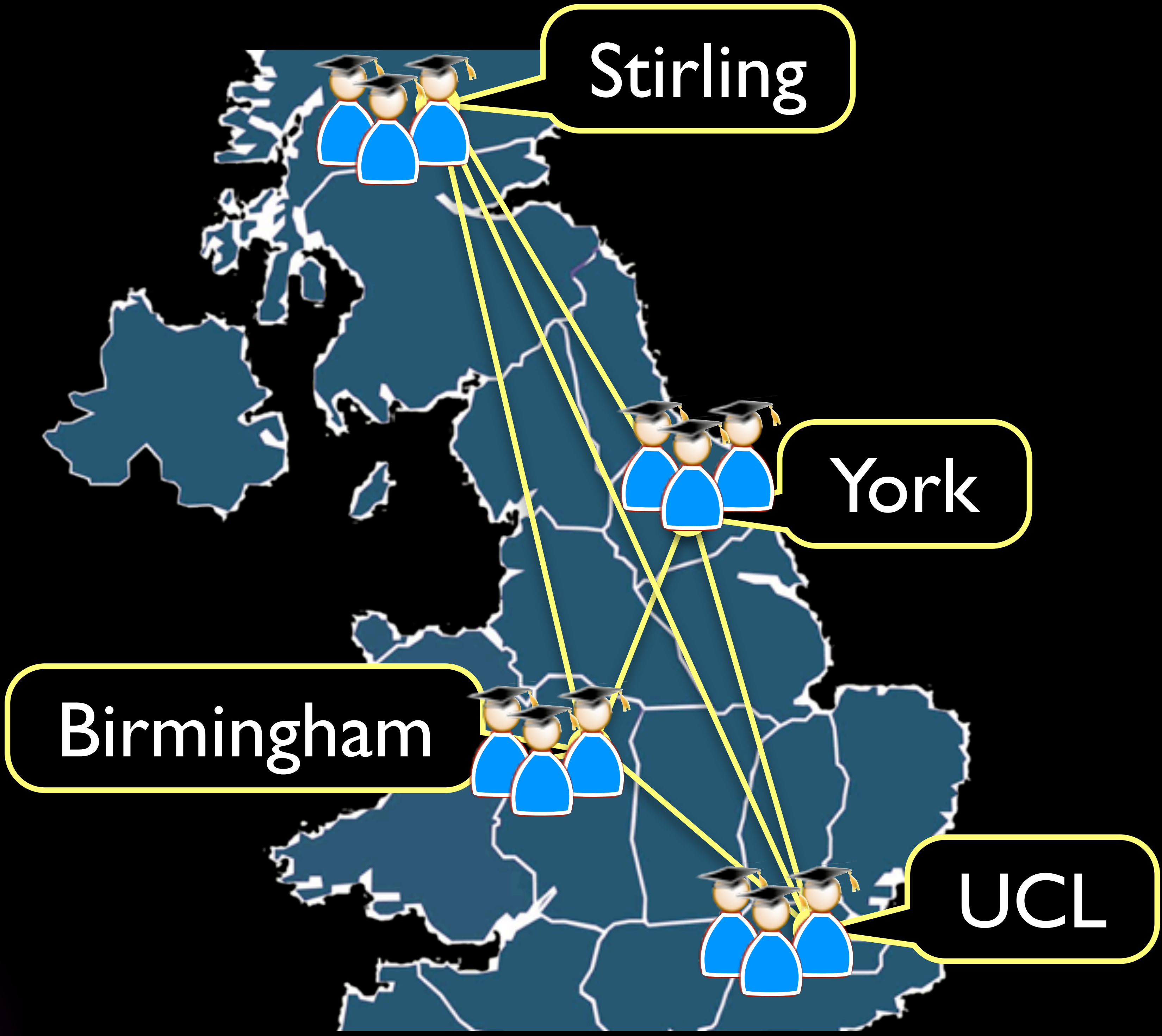
Mark Harman, CREST

Saturday, 16 February 13

EPSRC
Grant

DTC

Programme



Stirling

York

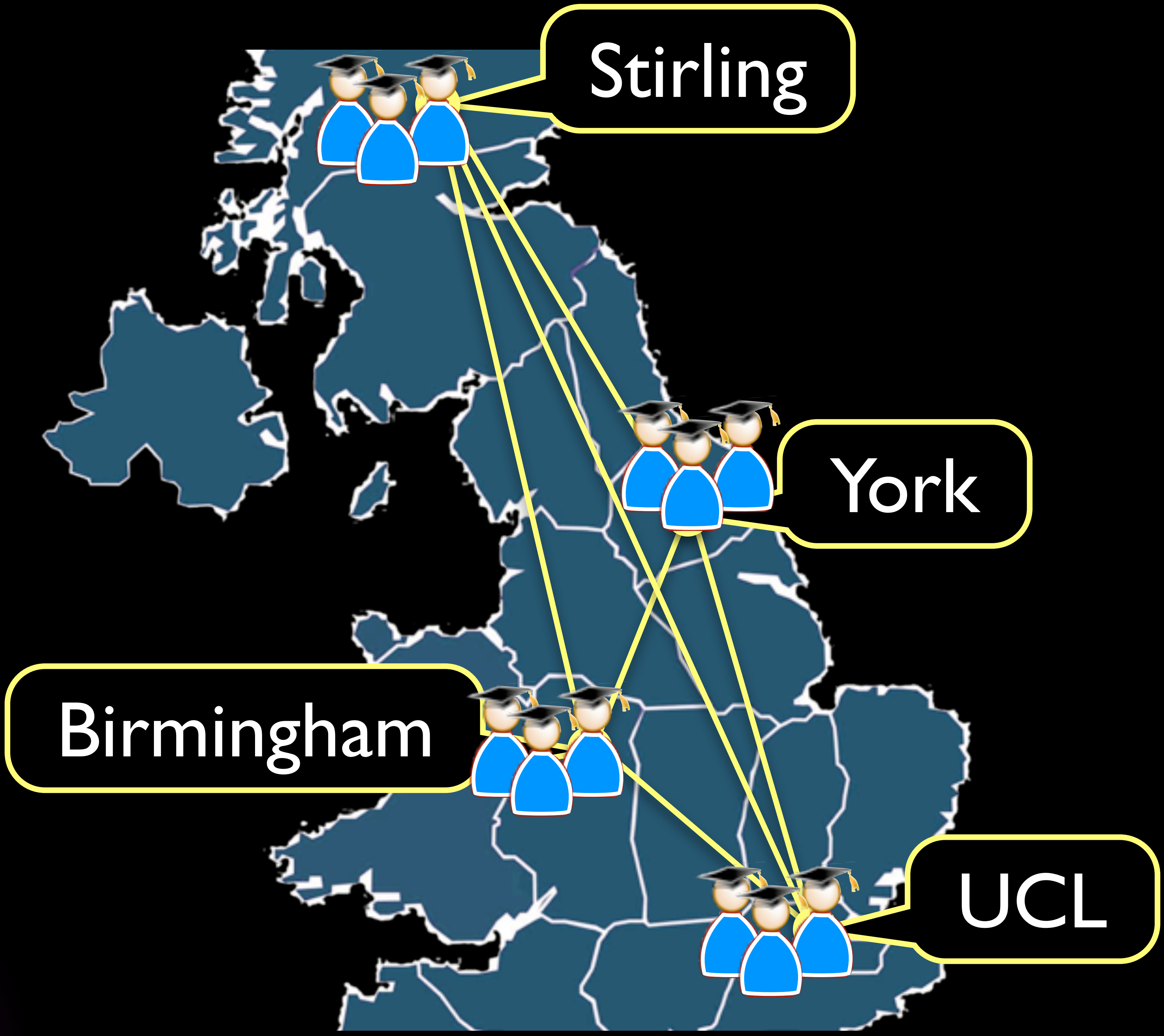
Birmingham

UCL

EPSRC
Grant

DTC

Programme



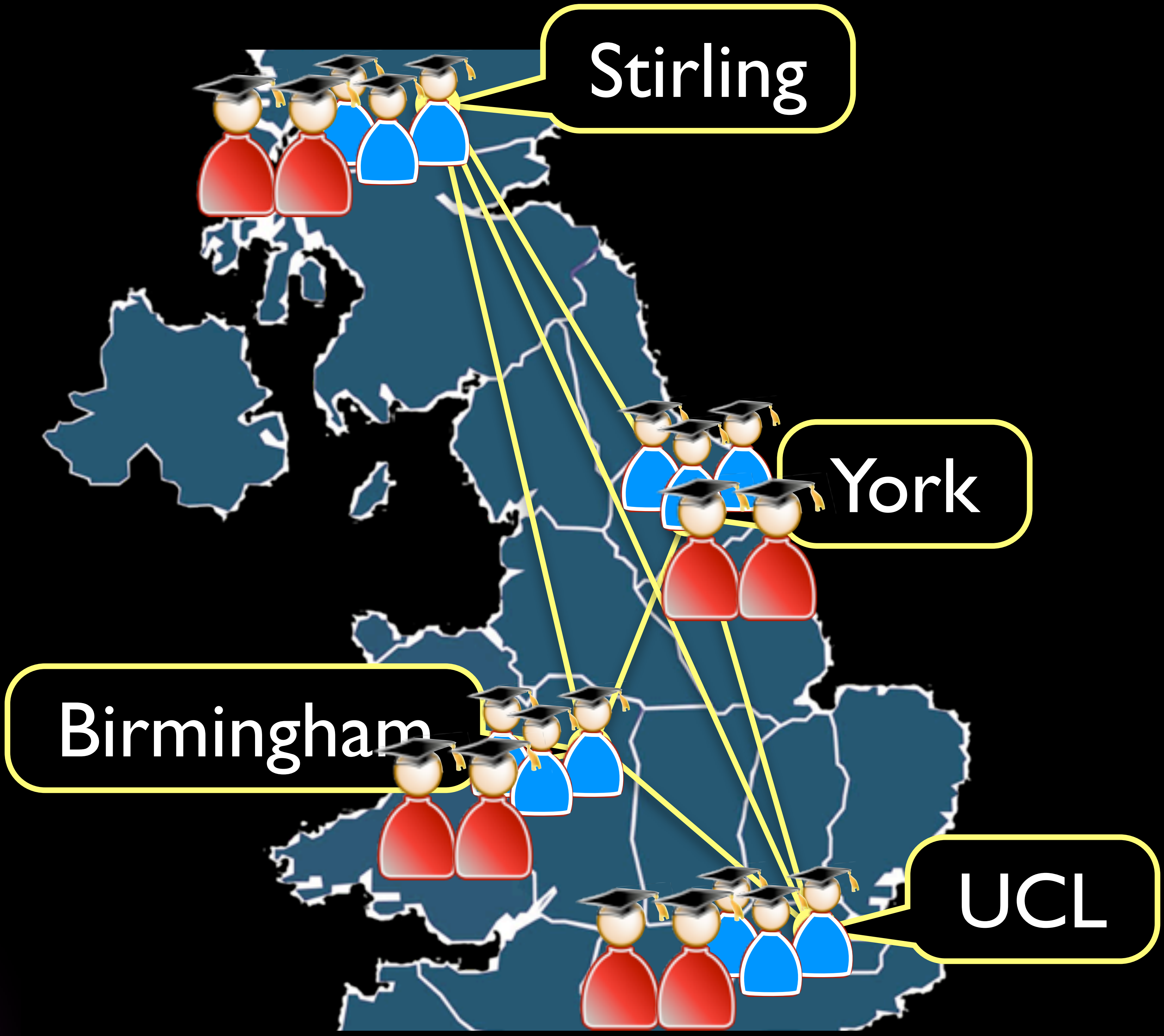
DAASE

Mark Harman, CREST

EPSRC
Grant

DTC

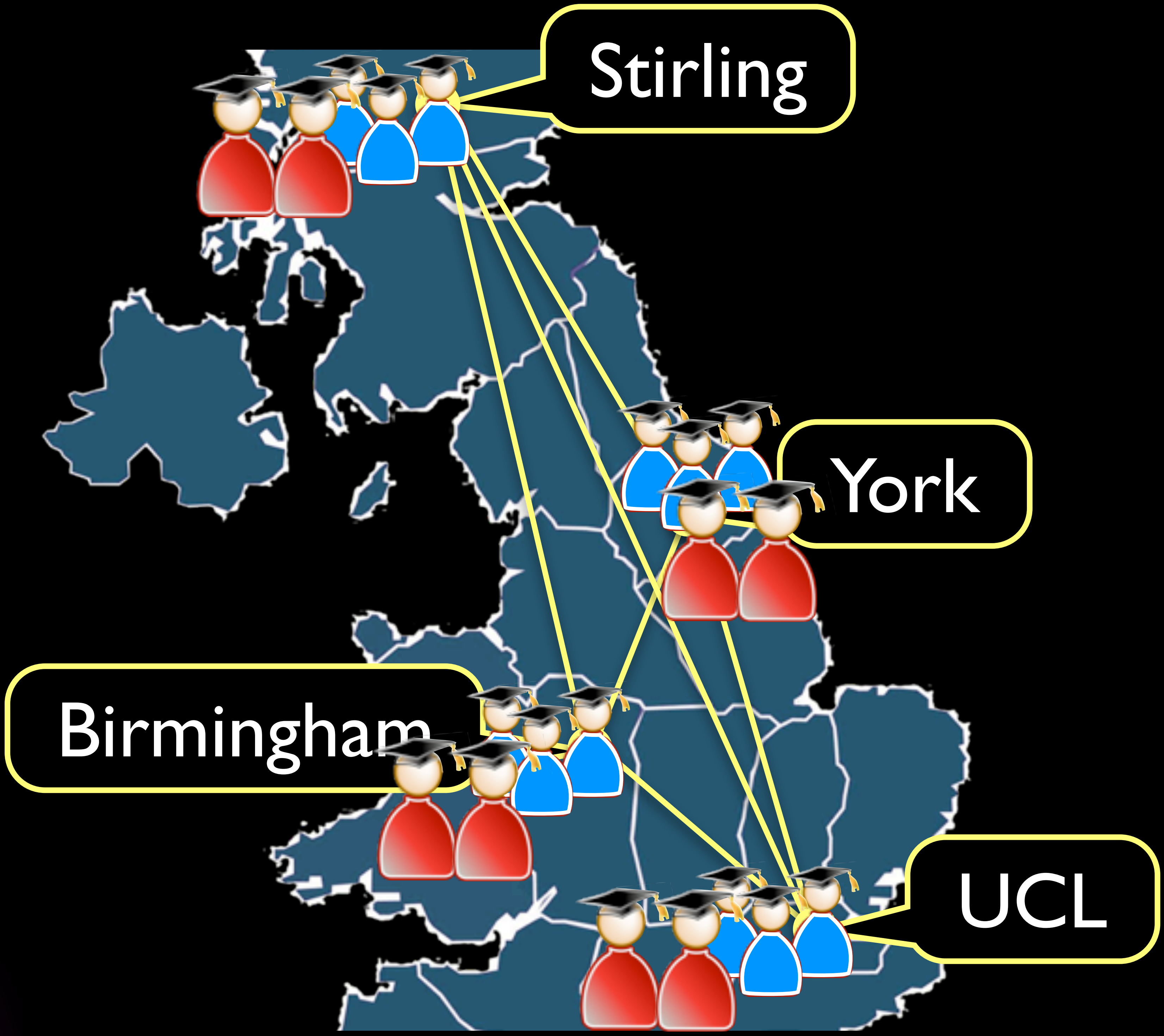
Programme



EPSRC
Grant

DTC

Programme

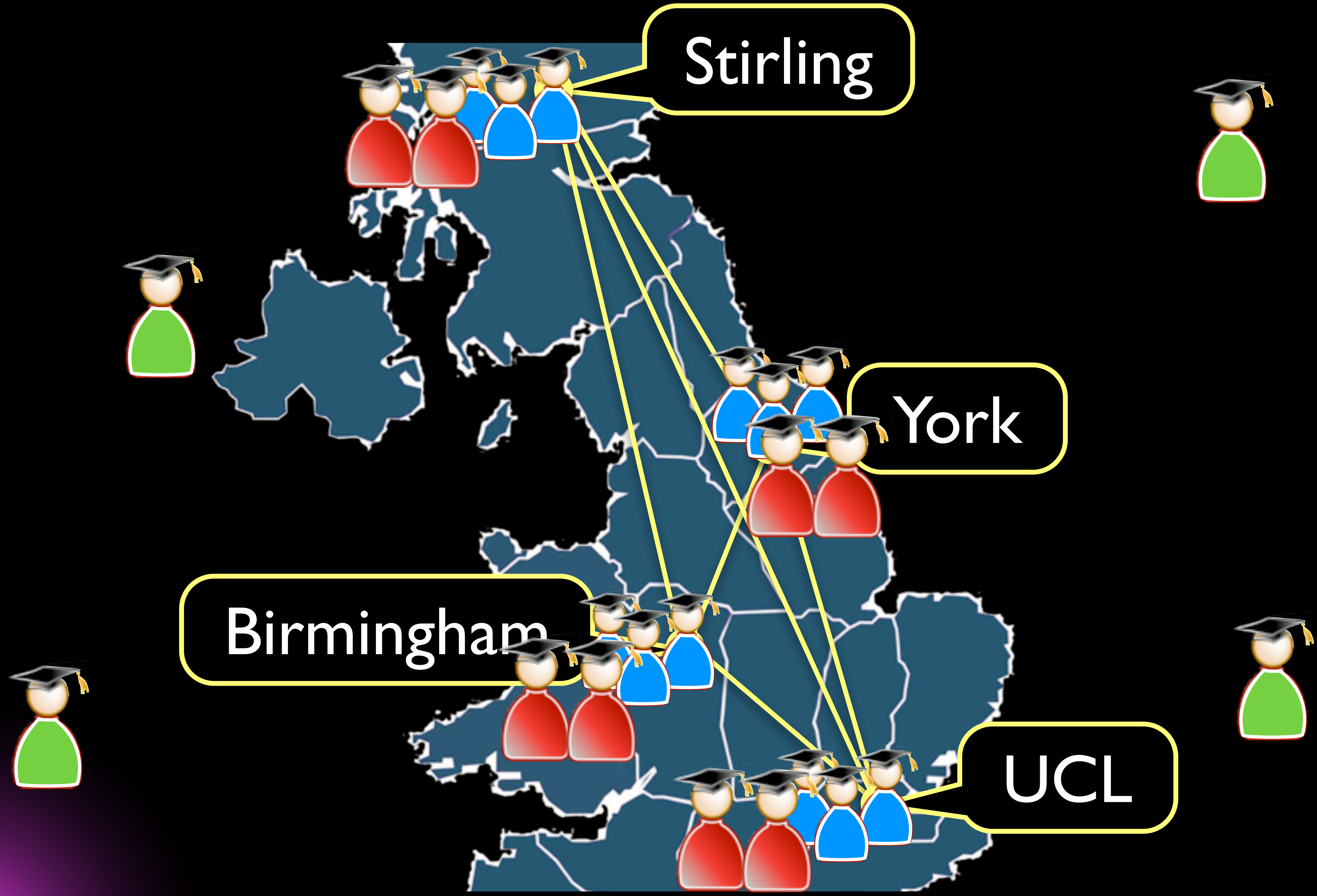


DAASE

EPSRC
Grant

DTC

Programme



DAASE

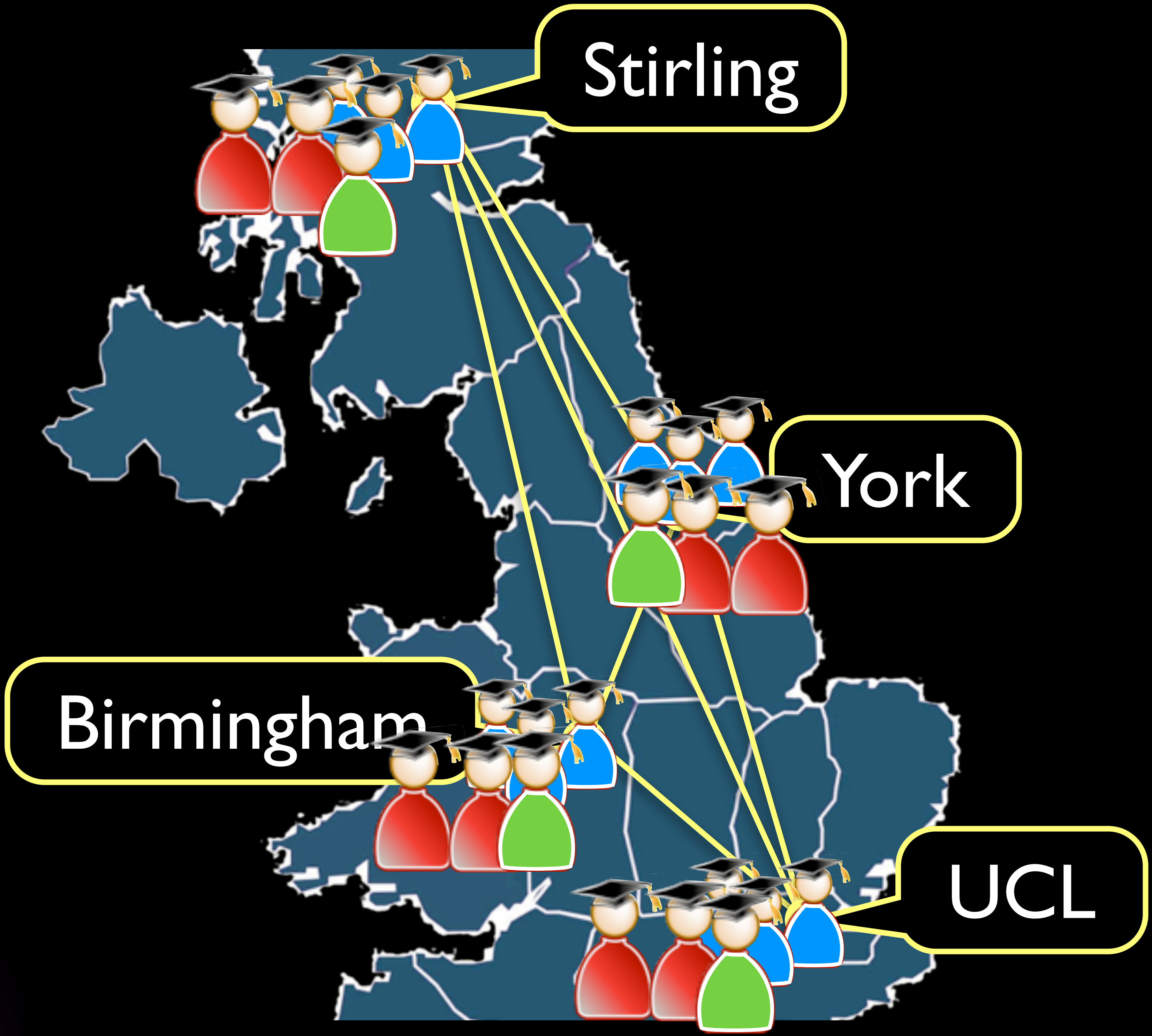
Mark Harman, CREST

Saturday, 16 February 13

EPSRC Grant

DTC

Programme



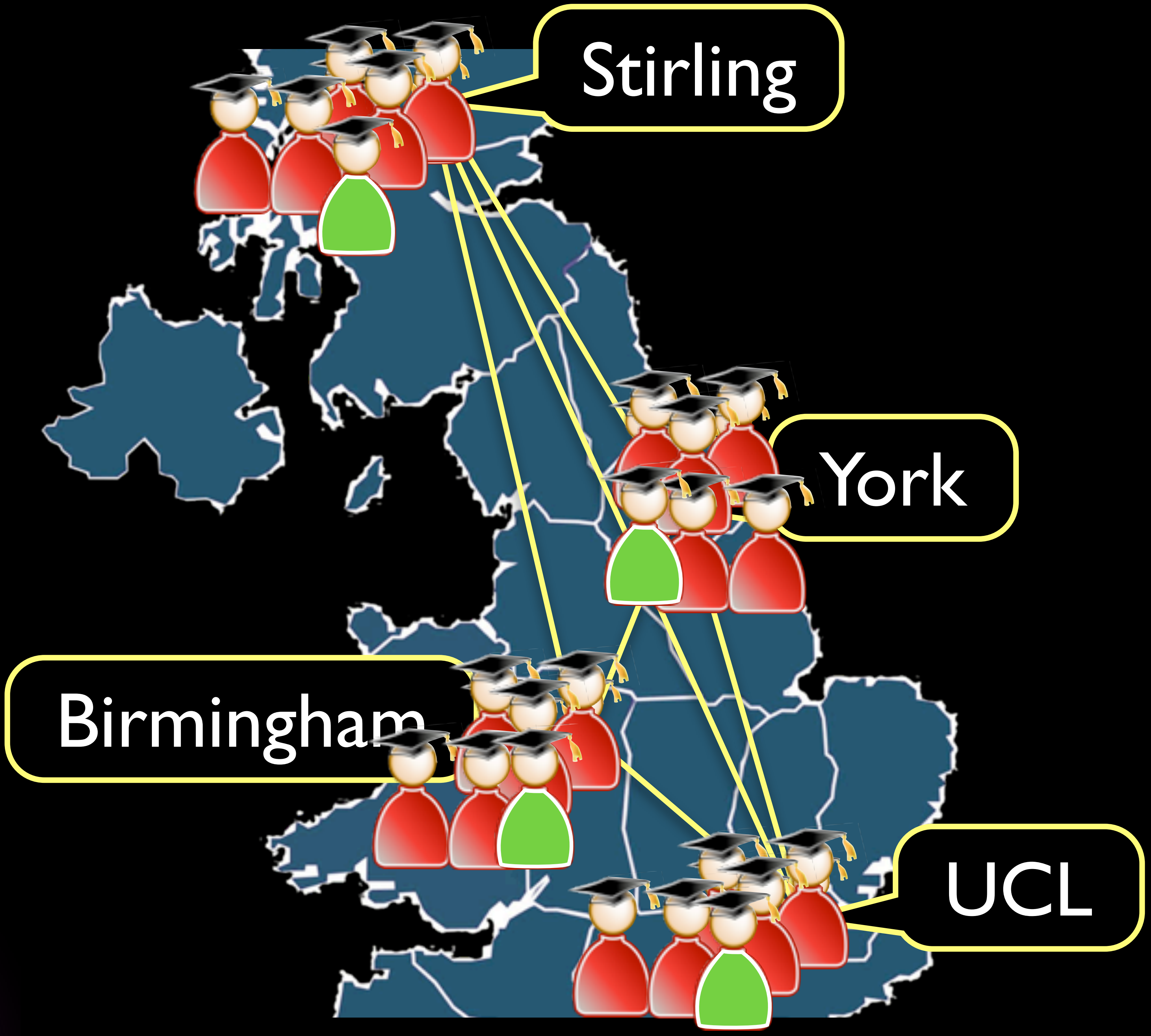
DAASE

Mark Harman, CREST

EPSRC
Grant

DTC

Programme



DAASE

Mark Harman, CREST

EPSRC
Grant

DTC

Programme

EPSRC Grant

DTC

Programme

DAASE

Mark Harman, CREST

EPSRC Grant

DTC

Programme

DAASE

Mark Harman, CREST

Dynamic Adaptive SBSE

Compile SBSE into deployed Software

Dynamic Adaptive SBSE

Compile SBSE into deployed Software

What ***is*** SBSE?

What is SBSE

What is SBSE

In SBSE we apply search techniques to search large search spaces, guided by a fitness function that captures properties of the acceptable software artefacts we seek.

What is SBSE

In SBSE we apply search techniques to search large search spaces, guided by a fitness function that captures properties of the acceptable software artefacts we seek.

like google search?
like code search?
like breadth first search?

What is SBSE

In SBSE we apply search techniques to search large search spaces, guided by a fitness function that captures properties of the acceptable software artefacts we seek.

like google search?
like code search?
like breadth first search?

DAASE

What is SBSE

In SBSE we apply search techniques to search large search spaces, guided by a fitness function that captures properties of the acceptable software artefacts we seek.

like google search?
like code search?
like breadth first search?

potentially
exhaustive

What is SBSE

In SBSE we apply search techniques to search large search spaces, guided by a fitness function that captures properties of the acceptable software artefacts we seek.

like google search?
like code search?
like breadth first search?

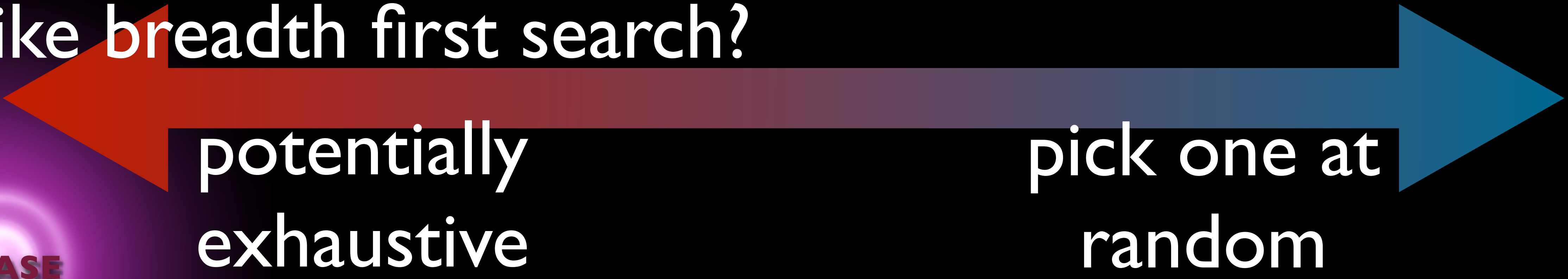
potentially exhaustive

pick one at random

What is SBSE

In SBSE we apply search techniques to search large search spaces, guided by a fitness function that captures properties of the acceptable software artefacts we seek.

like google search?
like code search?
like breadth first search?



potentially exhaustive

pick one at random

What is SBSE

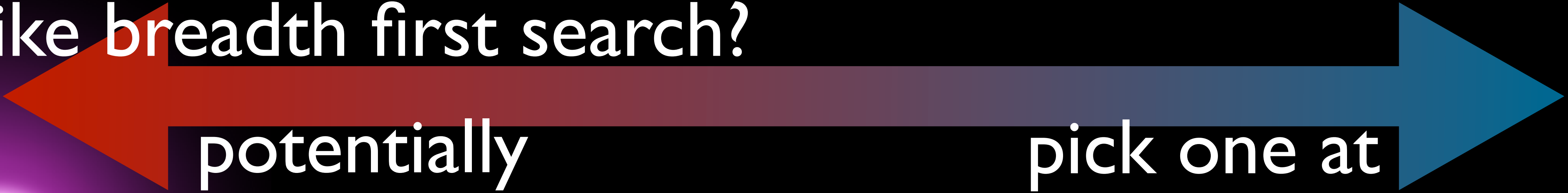
In SBSE we apply search techniques to search large search spaces, guided by a fitness function that captures properties of the acceptable software artefacts we seek.

sweet spot

like google search?

like code search?

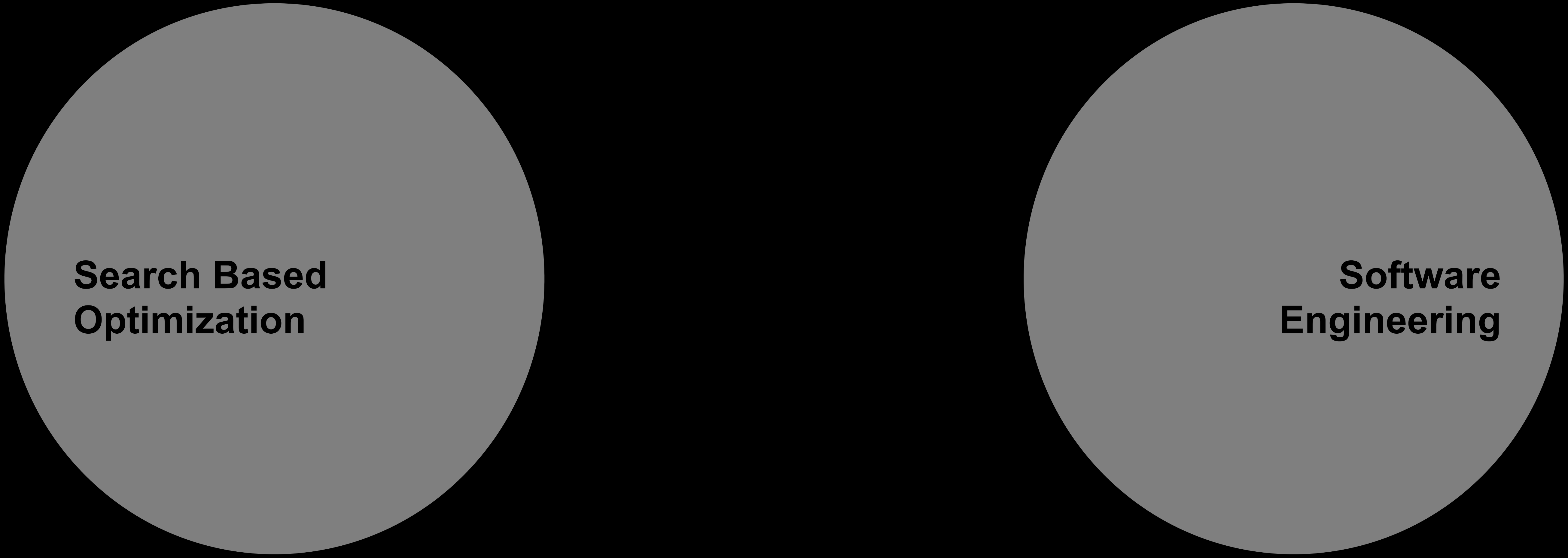
like breadth first search?



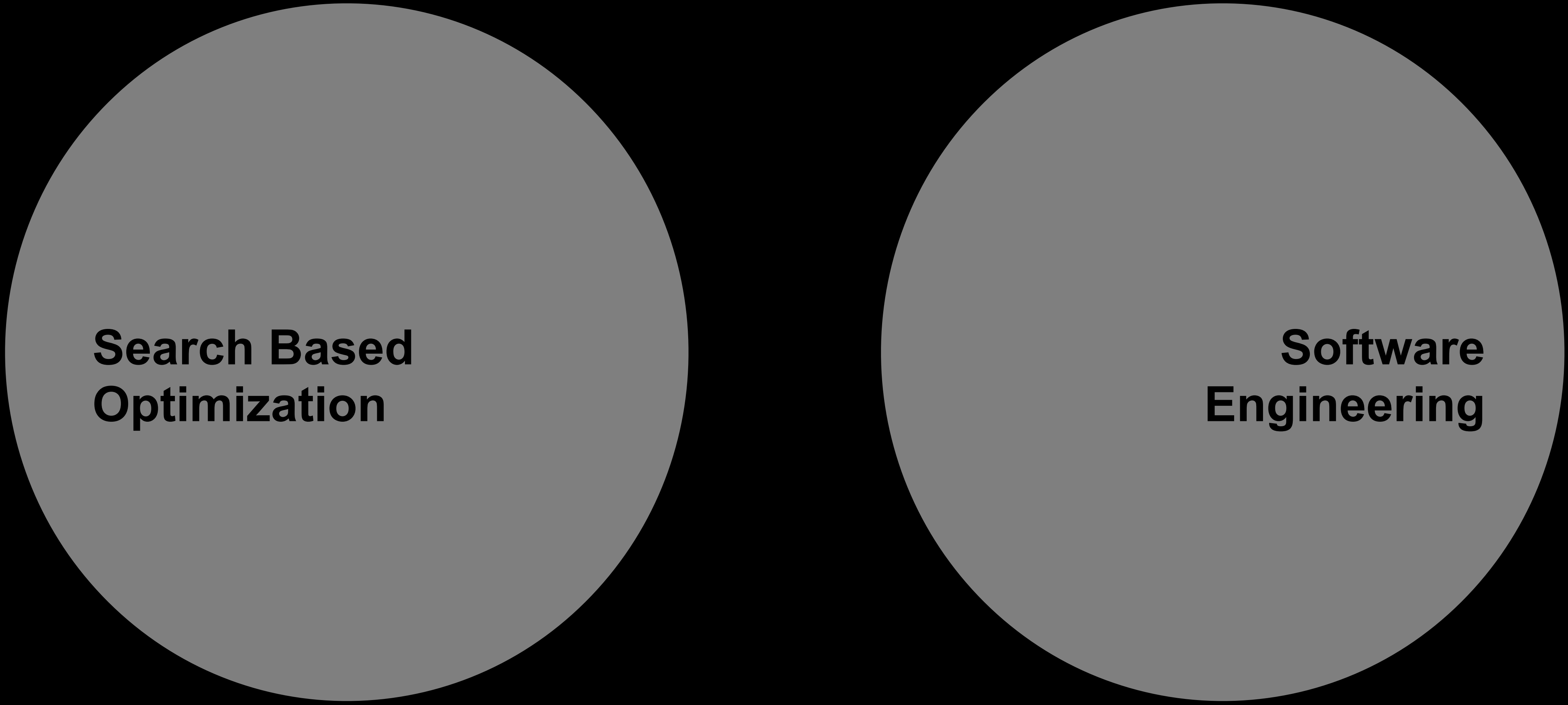
potentially exhaustive

pick one at random

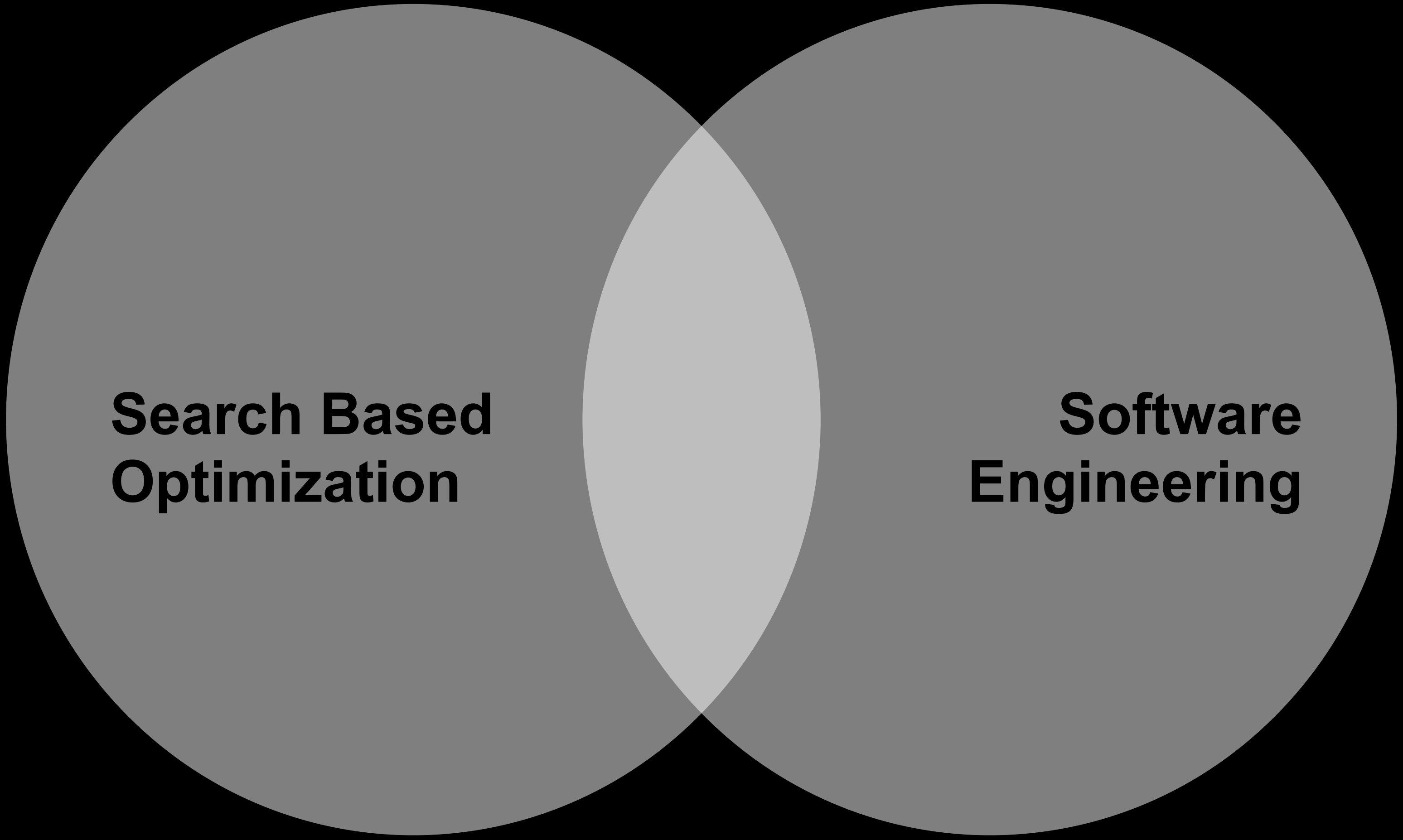
What is SBSE



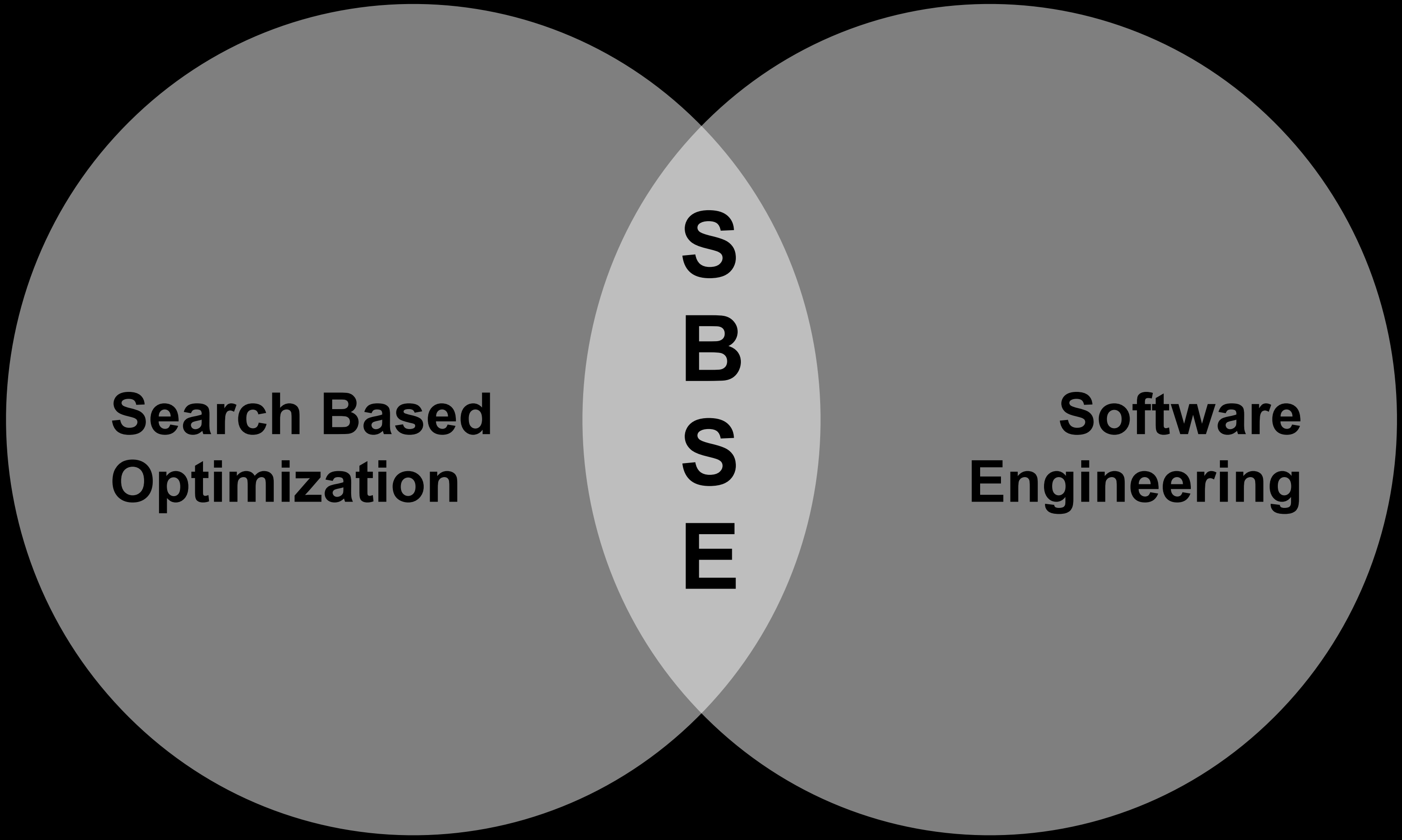
What is SBSE



What is SBSE



What is SBSE



What is SBSE

In SBSE we apply search techniques to search large search spaces, guided by a fitness function that captures properties of the acceptable software artefacts we seek.

What is SBSE

In SBSE we apply search techniques to search large search spaces, guided by a fitness function that captures properties of the acceptable software artefacts we seek.

Tabu Search

Particle Swarm Optimization

Ant Colonies

Genetic Algorithms

Hill Climbing

Genetic Programming

Simulated Annealing

Random

Greedy

LP

Estimation of Distribution Algorithms

What is SBSE

In SBSE we apply search techniques to search large search spaces, guided by a fitness function that captures properties of the acceptable software artefacts we seek.

Tabu Search

Particle Swarm Optimization

Ant Colonies

Genetic Algorithms

Hill Climbing

Genetic Programming

Simulated Annealing

Random

Greedy

LP

Estimation of Distribution Algorithms

Origins

Origins

Origins

1999 - 2003

Origins

1999 - 2003

Origins

SEMINAL 1999 - 2003

SEBASE 2006 - 2011

DAASE

Origins

SEMINAL 1999 - 2003

SEBASE 2006 - 2011

1998: Tracy, Clark and Mander

Origins

SEMINAL 1999 - 2003

SEBASE 2006 - 2011

1998: Tracy, Clark and Mander Feldt

Origins

SEMINAL 1999 - 2003

SEBASE 2006 - 2011

1998: Tracy, Clark and Mander   Feldt

1996: Roper

Origins

SEMINAL 1999 - 2003

SEBASE 2006 - 2011

1998: Tracy, Clark and Mander Feldt

1996: Roper

1995: Korel, Jones, Sthamer, Watkins

Origins

SEMINAL 1999 - 2003

SEBASE 2006 - 2011

1998: Tracy, Clark and Mander   Feldt

1996: Roper

1995: Korel, Jones, Sthamer, Watkins

1992: Xanthakis et al.

Origins

SEMINAL 1999 - 2003

SEBASE 2006 - 2011

1998: Tracy, Clark and Mander Feldt

1996: Roper

1995: Korel, Jones, Sthamer, Watkins

1992: Xanthakis et al.

1976: Miller and Spooner

What is SBSE

let’s listen to software engineers ...

... what sort of things do they say?

Software Engineers Say

Software Engineers Say

Software Engineers Say

We need to satisfy business and technical concerns

We need to reduce risk while maintaining completion time

We need increased cohesion and decreased coupling

We need fewer tests that find more nasty bugs

We need to optimise for all metrics M1,..., Mn

Software Engineers Say

We need to satisfy business and technical concerns

We need to reduce risk while maintaining completion time

We need increased cohesion and decreased coupling

We need fewer tests that find more nasty bugs

We need to optimise for all metrics M1,..., Mn

Software Engineers Say

We need to satisfy business and technical concerns

We need to reduce risk while maintaining completion time

We need increased cohesion and decreased coupling

We need fewer tests that find more nasty bugs

We need to optimise for all metrics M1,..., Mn

DAASE

Software Engineers Say

We need to satisfy business and technical concerns

We need to reduce risk while maintaining completion time

We need increased cohesion and decreased coupling

We need fewer tests that find more nasty bugs

We need to optimise for all metrics M1,..., Mn

Software Engineers Say

We need to satisfy business and technical concerns

We need to reduce risk while maintaining completion time

We need increased cohesion and decreased coupling

We need fewer tests that find more nasty bugs

We need to optimise for all metrics M1,..., Mn

Software Engineers Say

We need to satisfy business and technical concerns

We need to reduce risk while maintaining completion time

We need increased cohesion and decreased coupling

We need fewer tests that find more nasty bugs

We need to optimise for all metrics M1,..., Mn

Software Engineers Say

We need to satisfy business and technical concerns

We need to reduce risk while maintaining completion time

We need increased cohesion and decreased coupling

We need fewer tests that find more nasty bugs

We need to optimise for all metrics M1,..., Mn

Software Engineers Say

We need to satisfy business and technical concerns

We need to reduce risk while maintaining completion time

We need increased cohesion and decreased coupling

We need fewer tests that find more nasty bugs

We need to optimise for all metrics M1,..., Mn

Software Engineers Say

Requirements: We need to satisfy business and technical concerns

Management: We need to reduce risk while maintaining completion time

Design: We need increased cohesion and decreased coupling

Testing: We need fewer tests that find more nasty bugs

Refactoring: We need to optimise for all metrics M1,..., Mn

Software Engineers Say

Requirements: We need to satisfy business and technical concerns

Management: We need to reduce risk while maintaining completion time

Design: We need increased cohesion and decreased coupling

Testing: We need fewer tests that find more nasty bugs

Refactoring: We need to optimise for all metrics M1,..., Mn

All have been addressed in the SBSE literature

Engineering words

Engineering words

with acceptable bounds

tolerance

improve performance

optimise

reduce cost

optimize

fit for purpose

within constraints

Engineering words

with acceptable bounds

tolerance

improve performance

optimise

reduce cost

optimize

fit for purpose

within constraints

Engineering words

with acceptable bounds

tolerance

improve performance

optimise

reduce cost

optimize

fit for purpose

within constraints

DAASE

Engineering words

with acceptable bounds

tolerance

improve performance

optimise

reduce cost

optimize

fit for purpose

within constraints

DAASE

Engineering words

with acceptable bounds

tolerance

improve performance

optimise

reduce cost

optimize

fit for purpose

within constraints

Engineering words

with acceptable bounds

tolerance

improve performance

optimise

reduce cost

optimize

fit for purpose

within constraints

DAASE

Engineering words

with acceptable bounds

tolerance

improve performance

optimise

reduce cost

optimize

fit for purpose

within constraints

DAASE

Engineering words

with acceptable bounds

tolerance

improve performance

optimise

reduce cost

optimize

fit for purpose

within constraints

The advantages of SBSE

The advantages of SBSE

The advantages of SBSE

Insight-rich

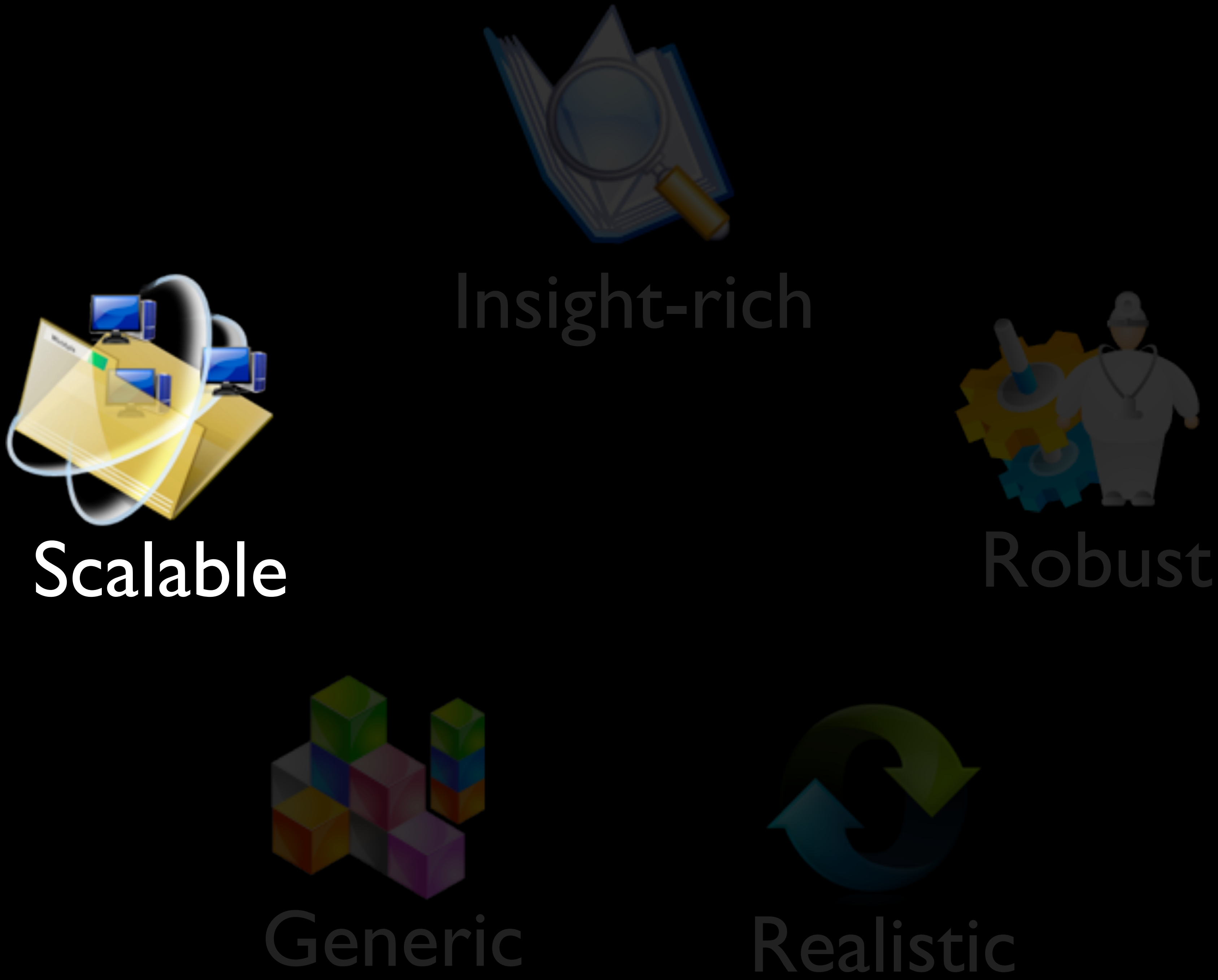
Scalable

Robust

Generic

Realistic

The advantages of SBSE



Scalable

Insight-rich

Robust

Generic

Realistic

The advantages of SBSE

Insight-rich

Scalable

Robust

Generic

Realistic

The advantages of SBSE

Insight-rich

Scalable

Robust

Generic

Realistic

The advantages of SBSE

Insight-rich

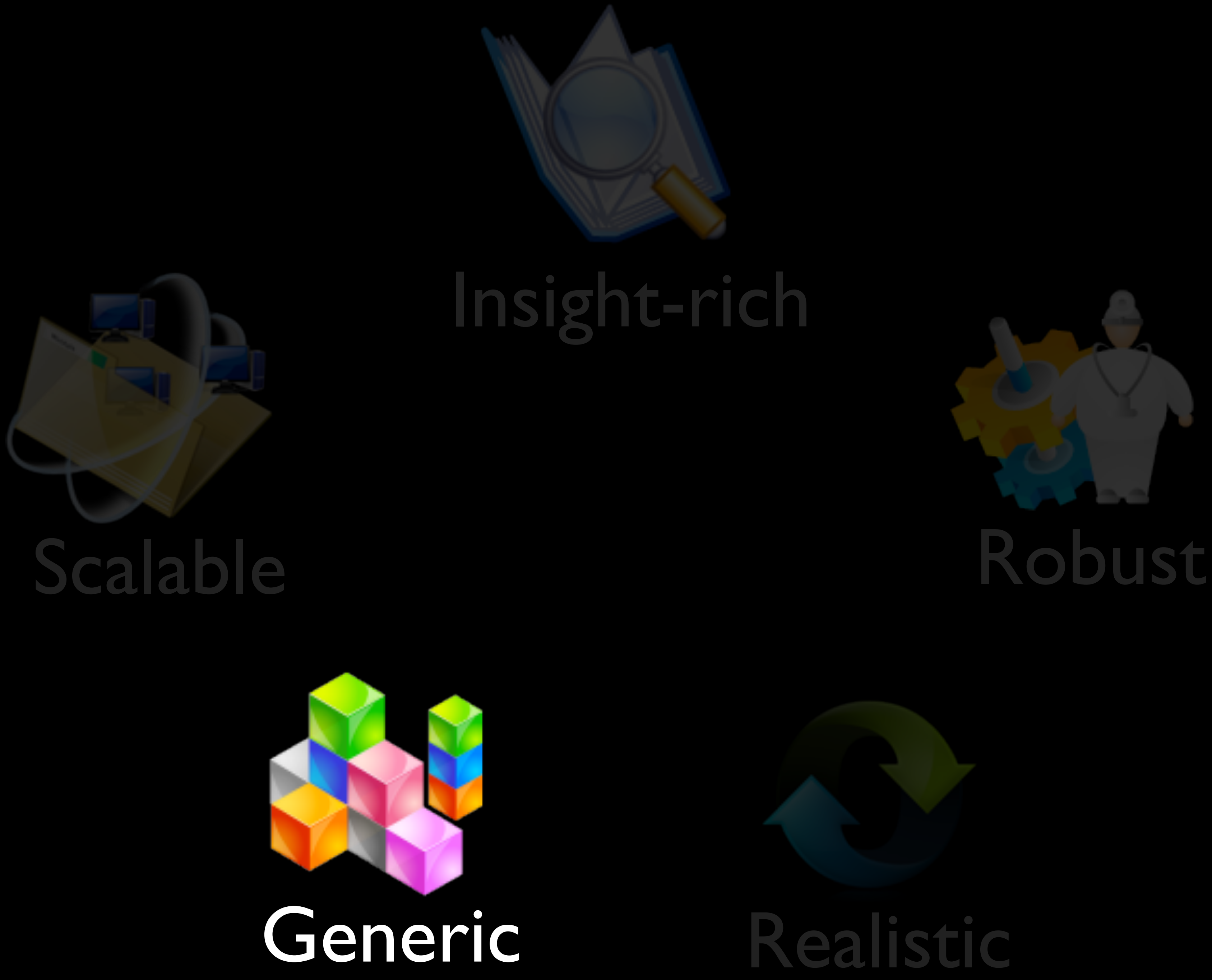
Scalable

Robust

Generic

Realistic

The advantages of SBSE



Insight-rich

Scalable

Robust

Generic

Realistic

... but ...
why is
Software Engineering
different?

in situ fitness test

in situ fitness test

Physical Engineering

in situ fitness test

Physical Engineering

in situ fitness test

Physical Engineering

cost: $20,000.00

in situ fitness test

Physical Engineering

Virtual Engineering

cost: $20,000.00

DAASE

in situ fitness test

Physical Engineering

cost: $20,000.00

Virtual Engineering

in situ fitness test

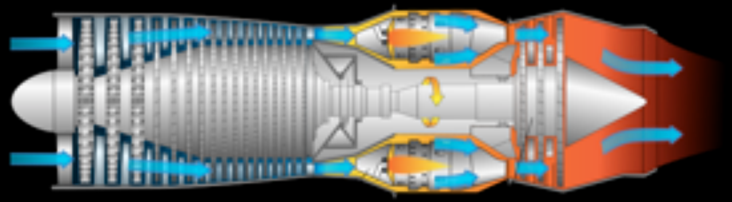
Physical Engineering

cost: $20,000.00

Virtual Engineering

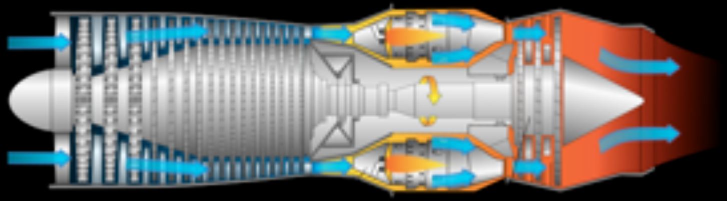
cost: $0.00.0000000002

spot the difference



spot the difference

Traditional Engineering Artifact



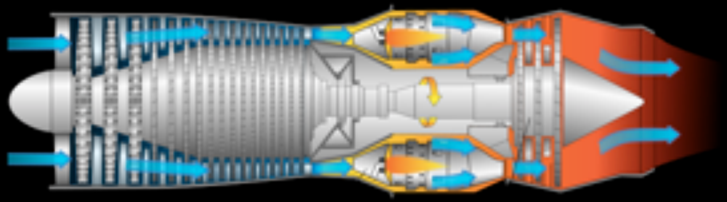
DAASE

Mark Harman, CREST

spot the difference

Traditional Engineering Artifact

Optimization goal



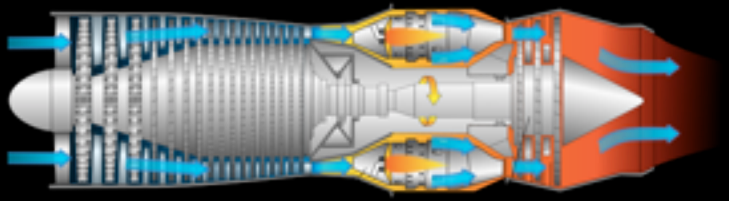
DAASE

spot the difference

Traditional Engineering Artifact

Optimization goal

Maximize compression



DAASE

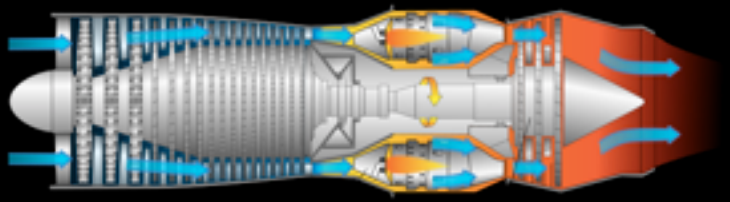
spot the difference

Traditional
Engineering Artifact

Optimization
goal

Maximize compression

Minimize fuel consumption



DAASE

Mark Harman, CREST

spot the difference

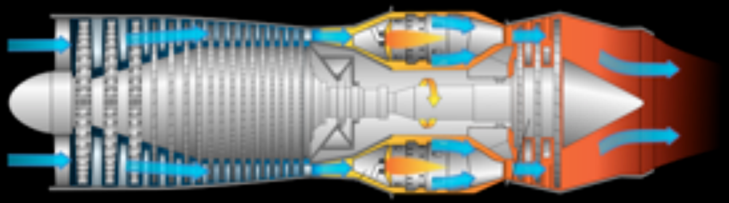
Traditional Engineering Artifact

Optimization goal

Maximize compression

Minimize fuel consumption

Fitness computed on a representation



spot the difference

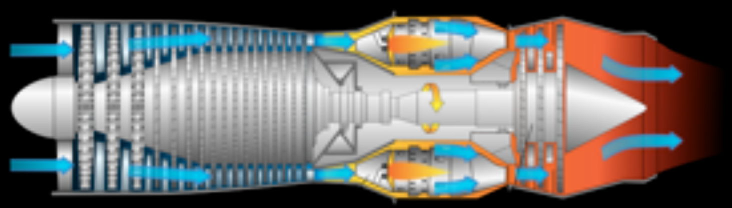
Traditional Engineering Artifact

Optimization goal

Maximize compression

Minimize fuel consumption

Fitness computed on a representation



Software Engineering Artifact

spot the difference

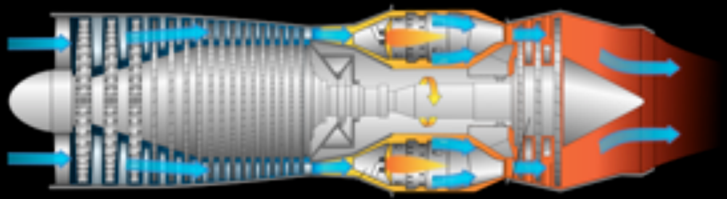
Traditional Engineering Artifact

Optimization goal

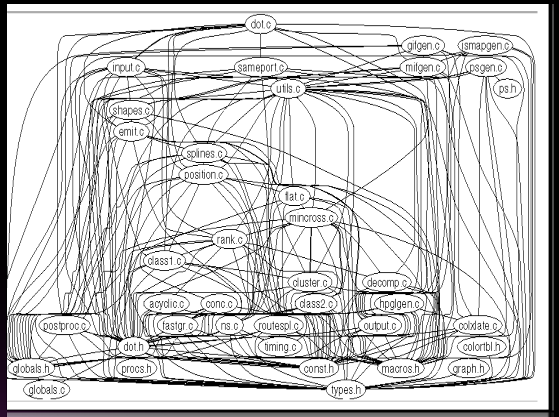
Fitness computed on a representation

Maximize compression

Minimize fuel consumption



Software Engineering Artifact



spot the difference

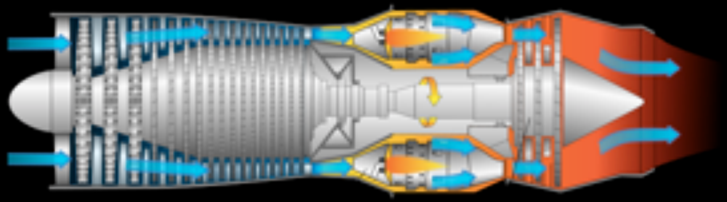
Traditional Engineering Artifact

Optimization goal

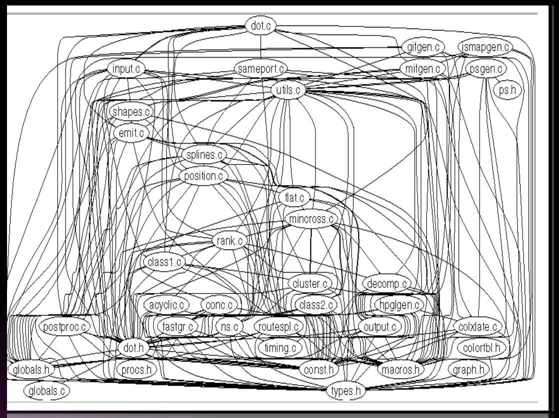
Maximize compression

Minimize fuel consumption

Fitness computed on a representation



Software Engineering Artifact



Optimization goal

spot the difference

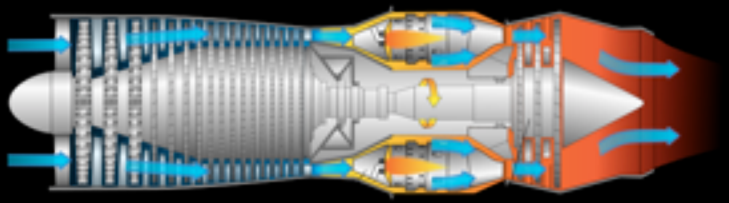
Traditional
Engineering Artifact

Optimization
goal

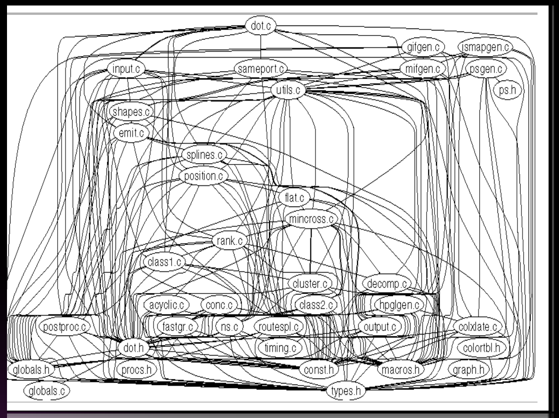
Maximize compression

Minimize fuel consumption

Fitness computed
on a representation



Software
Engineering Artifact



Optimization
goal

Maximize cohesion

DAASE

Mark Harman, CREST

Saturday, 16 February 13

spot the difference

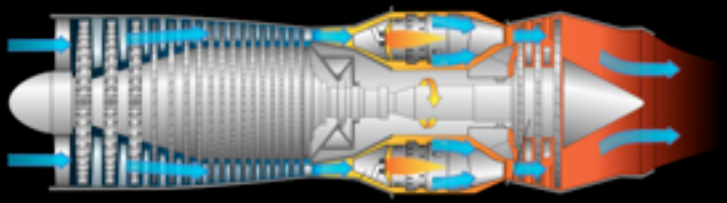
Traditional Engineering Artifact

Optimization goal

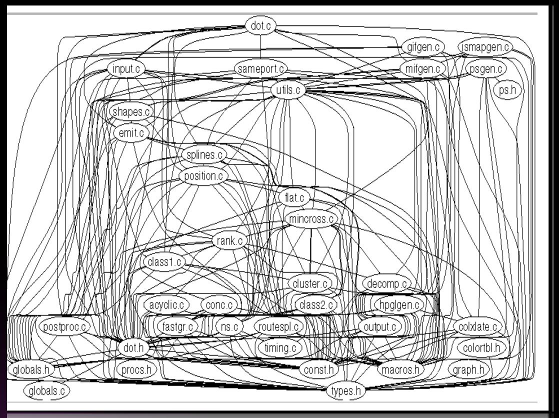
Maximize compression

Minimize fuel consumption

Fitness computed on a representation



Software Engineering Artifact



Optimization goal

Maximize cohesion

Minimize coupling

spot the difference

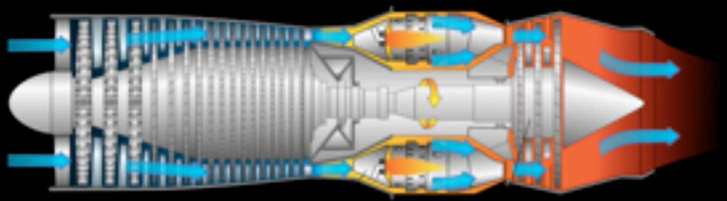
Traditional
Engineering Artifact

Optimization
goal

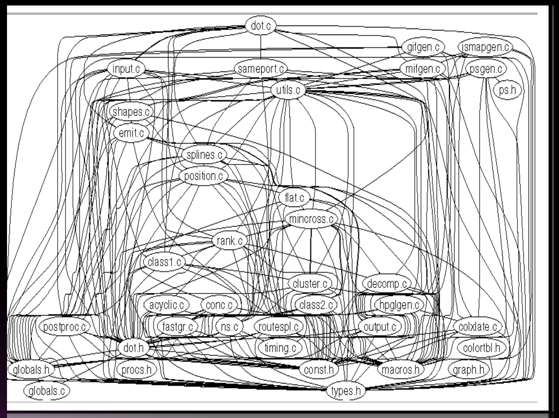
Maximize compression

Minimize fuel consumption

Fitness computed
on a representation



Software
Engineering Artifact



Optimization
goal

Maximize cohesion

Minimize coupling

Fitness computed
Directly

spot the difference

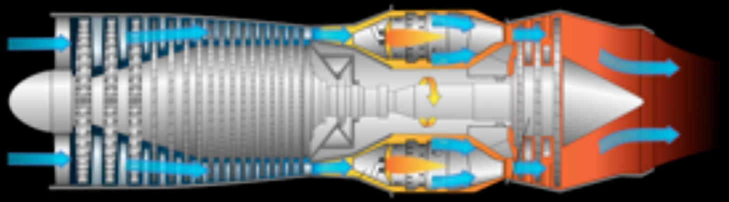
Traditional Engineering Artifact

Optimization goal

Fitness computed on a representation

Maximize compression

Minimize fuel consumption



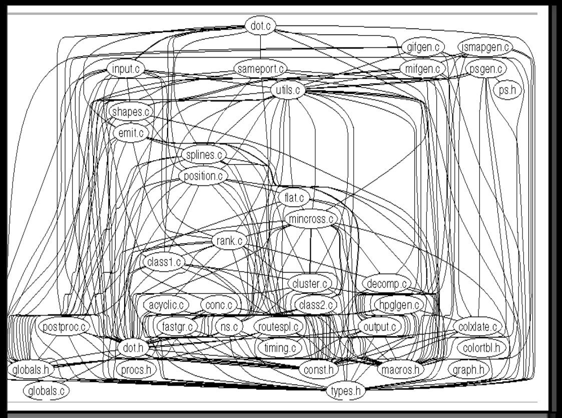
Software Engineering Artifact

Optimization goal

Fitness computed Directly

Maximize cohesion

Minimize coupling



DAASE

spot the difference

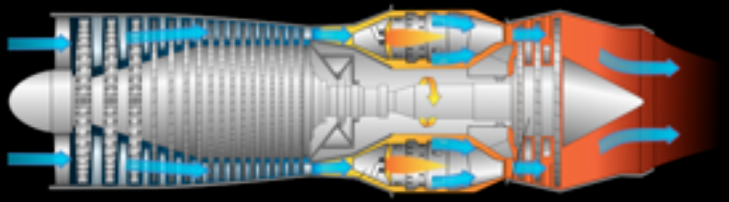
Traditional Engineering Artifact

Optimization goal

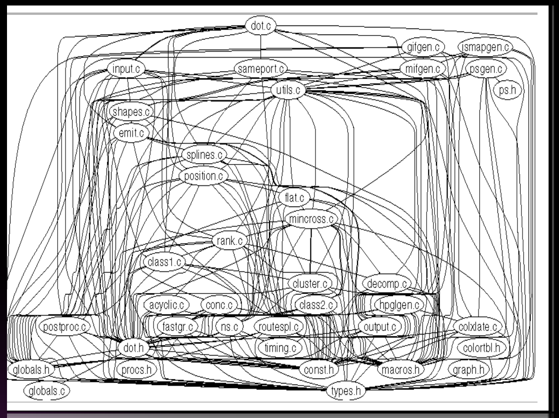
Maximize compression

Minimize fuel consumption

Fitness computed on a representation



Software Engineering Artifact

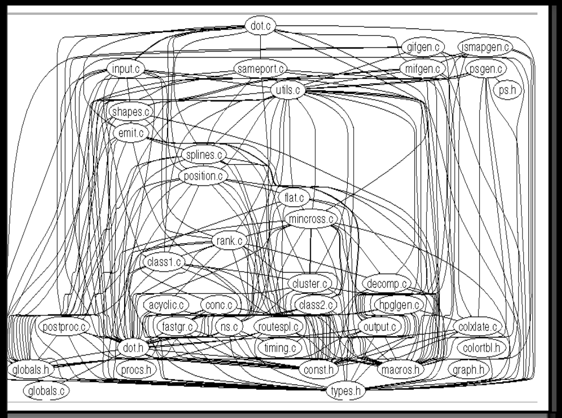


Optimization goal

Maximize cohesion

Minimize coupling

Fitness computed Directly



spot the difference

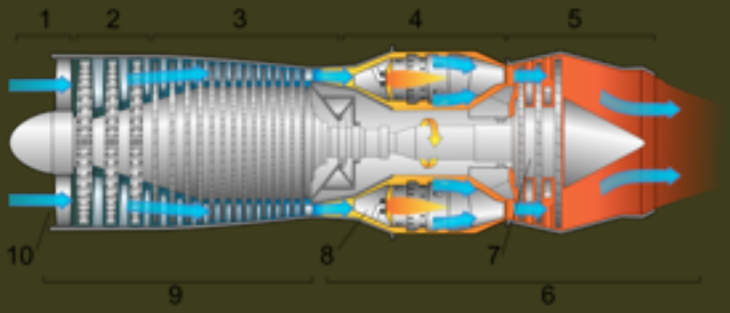
Traditional Engineering Artifact

Optimization goal

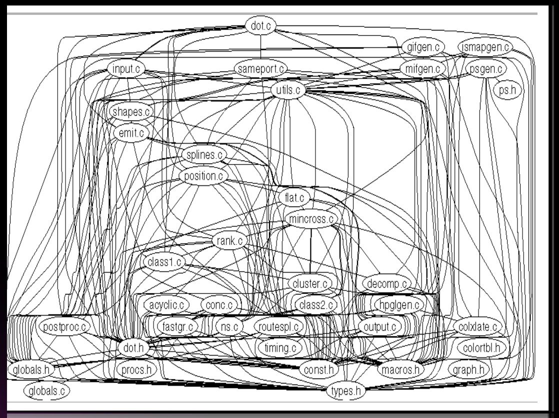
Maximize compression

Minimize fuel consumption

Fitness computed on a representation



Software Engineering Artifact

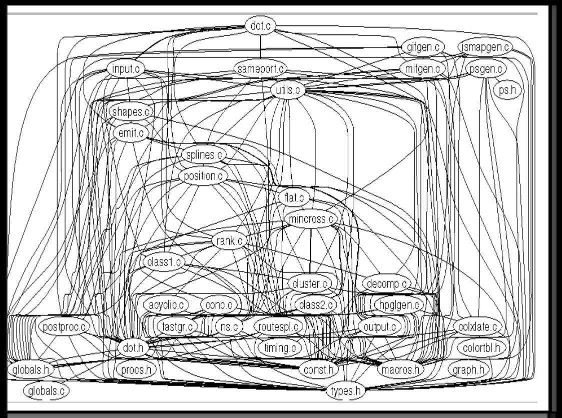


Optimization goal

Maximize cohesion

Minimize coupling

Fitness computed Directly



spot the difference

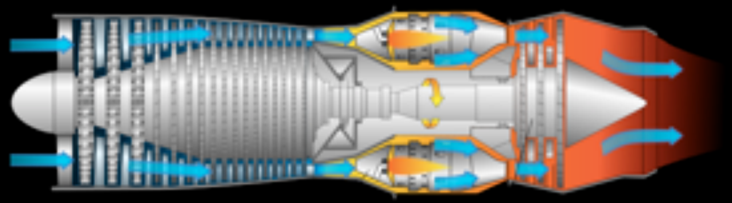
Traditional
Engineering Artifact

Optimization
goal

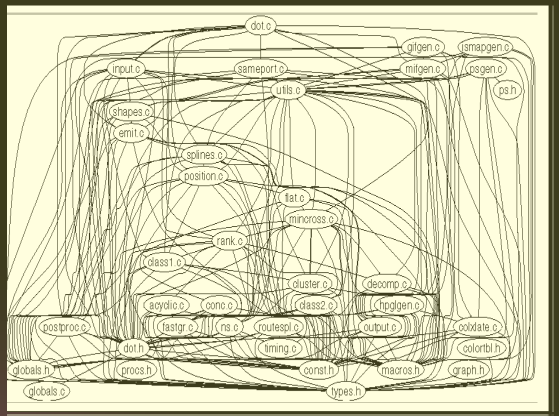
Maximize compression

Minimize fuel consumption

Fitness computed
on a representation



Software
Engineering Artifact

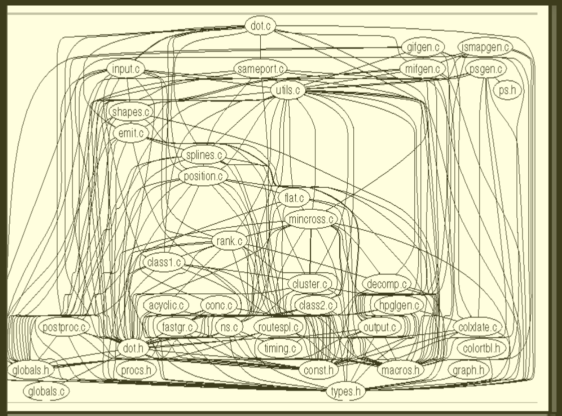


Optimization
goal

Maximize cohesion

Minimize coupling

Fitness computed
Directly



spot the difference

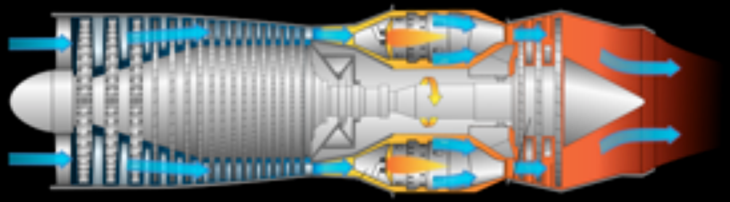
Traditional
Engineering Artifact

Optimization
goal

Maximize compression

Minimize fuel consumption

Fitness computed
on a representation



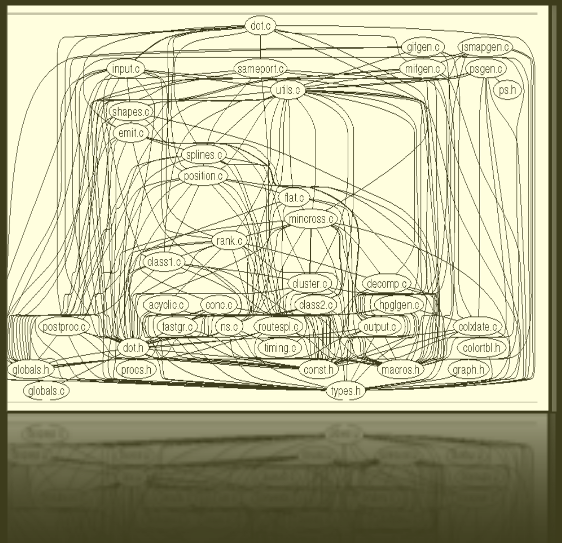
Software
Engineering Artifact

Optimization
goal

Maximize cohesion

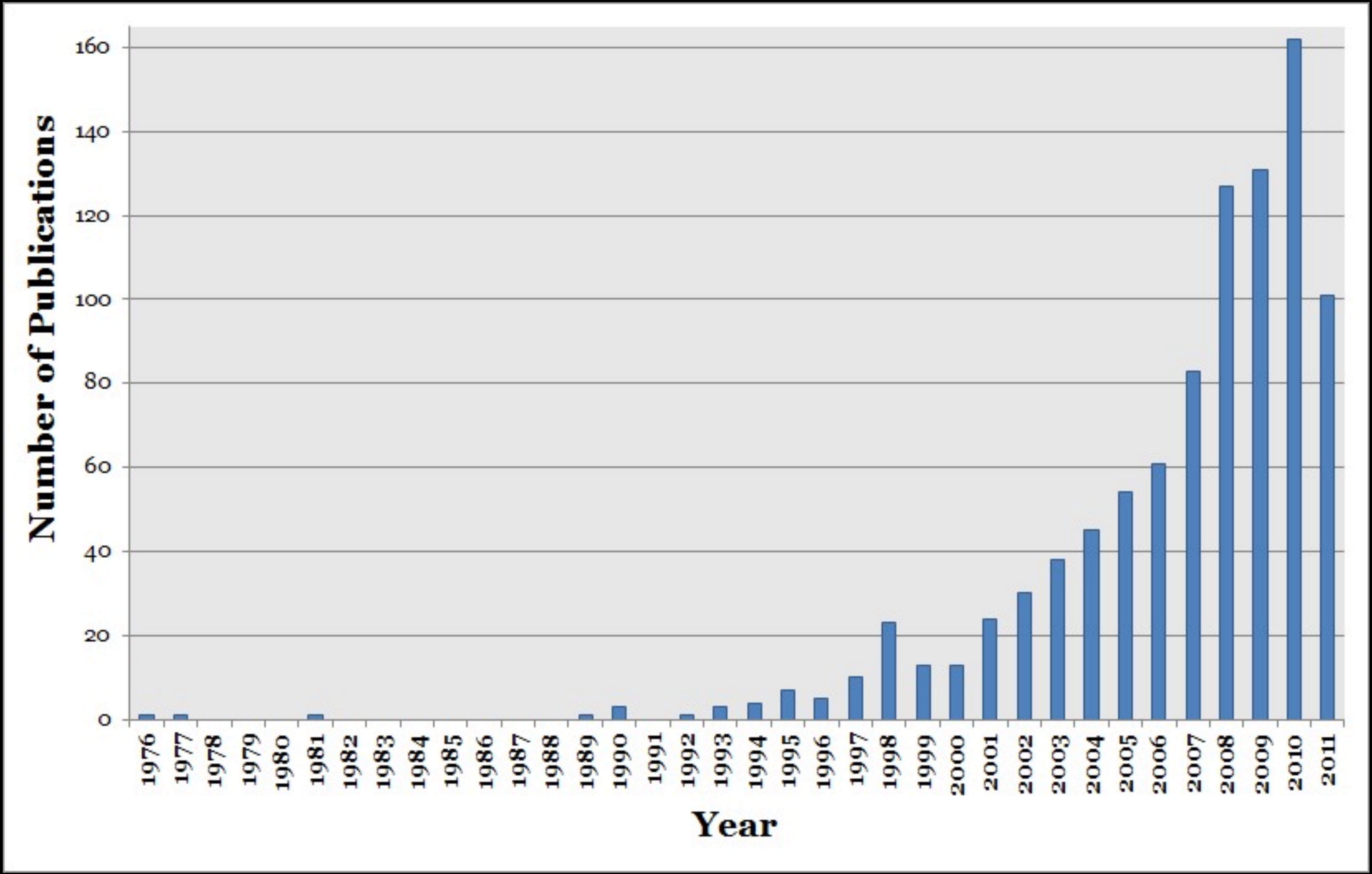
Minimize coupling

Fitness computed
Directly

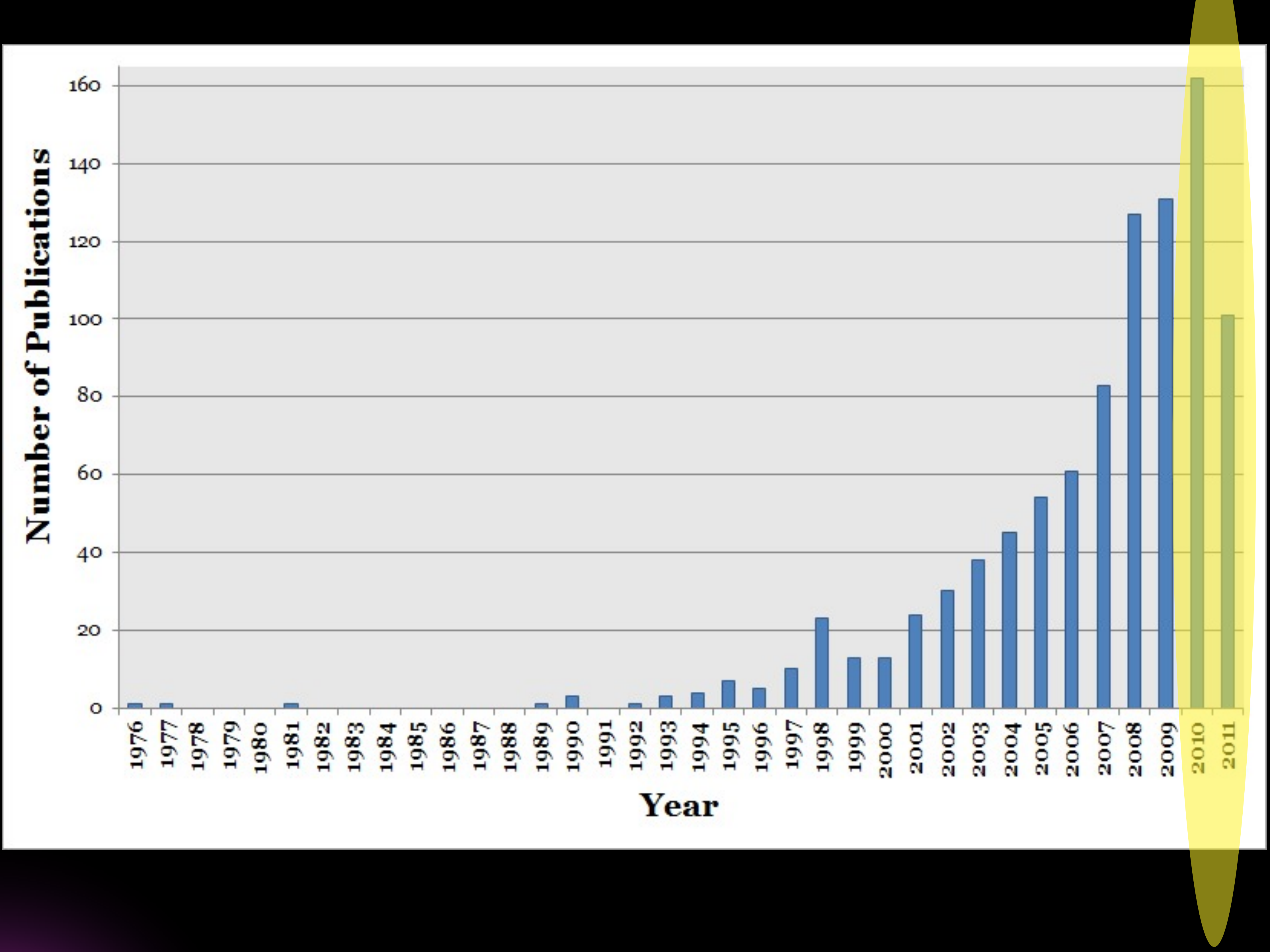


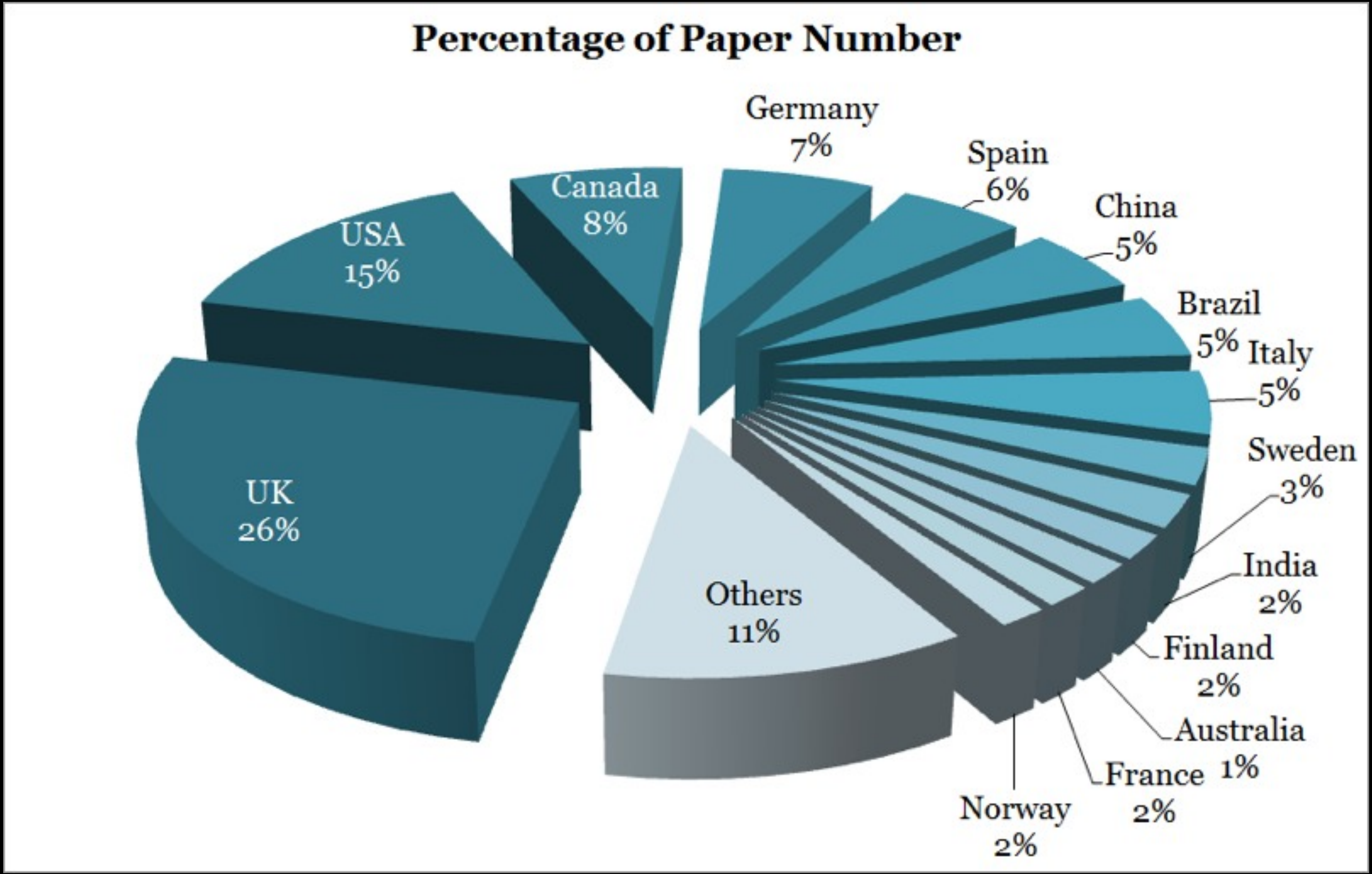
Mark Harman: ETAPS 2010 Keynote paper

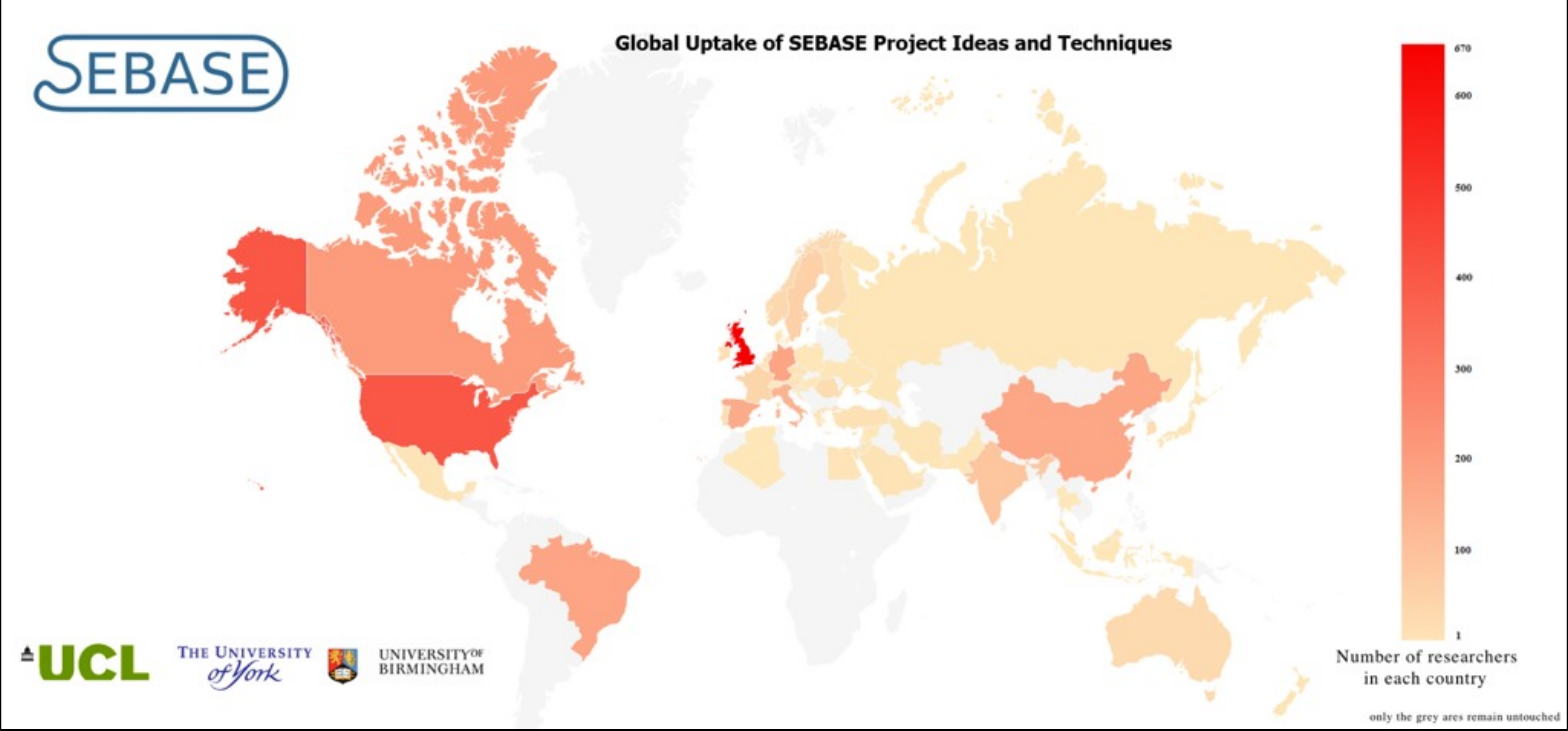
Growth Trends


Number of Publications
0
20
40
60
80
100
120
140
160
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
Year

DAASE


Number of Publications
160
140
120
100
80
60
40
20
0
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
Year


Percentage of Paper Number
UK 26%
USA 15%
Canada 8%
Germany 7%
Spain 6%
China 5%
Brazil 5%
Italy 5%
Sweden 3%
India 2%
Finland 2%
Australia 1%
France 2%
Norway 2%
Others 11%



Global Uptake of SEBASE Project Ideas and Techniques

The First Chinese SBSE Workshop

WESB 2012

3° Workshop de Engenharia de Software Baseada em Buscas

23 de Setembro de 2012 | Natal-RN-Brasil

ssbse

4th Symposium on

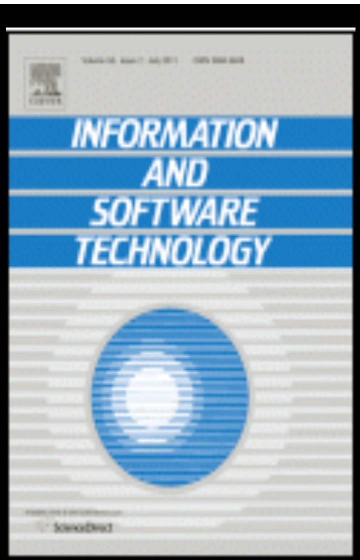
Search Based Software Engineering

September 28th - 30th, 2012
Riva del Garda | Trento | Italy

GECCO

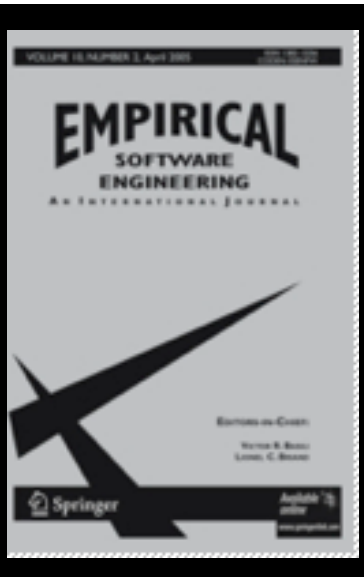
Genetic and Evolutionary Computation Conference

2011


INFORMATION AND SOFTWARE TECHNOLOGY

computers & operations research

JOURNAL OF SOFTWARE MAINTENANCE AND EVOLUTION RESEARCH AND PRACTICE


EMPIRICAL SOFTWARE ENGINEERING

An International Journal

Springer

SOFTWARE PRACTICE & EXPERIENCE

IEEE TRANSACTIONS ON SOFTWARE ENGINEERING

IEEE

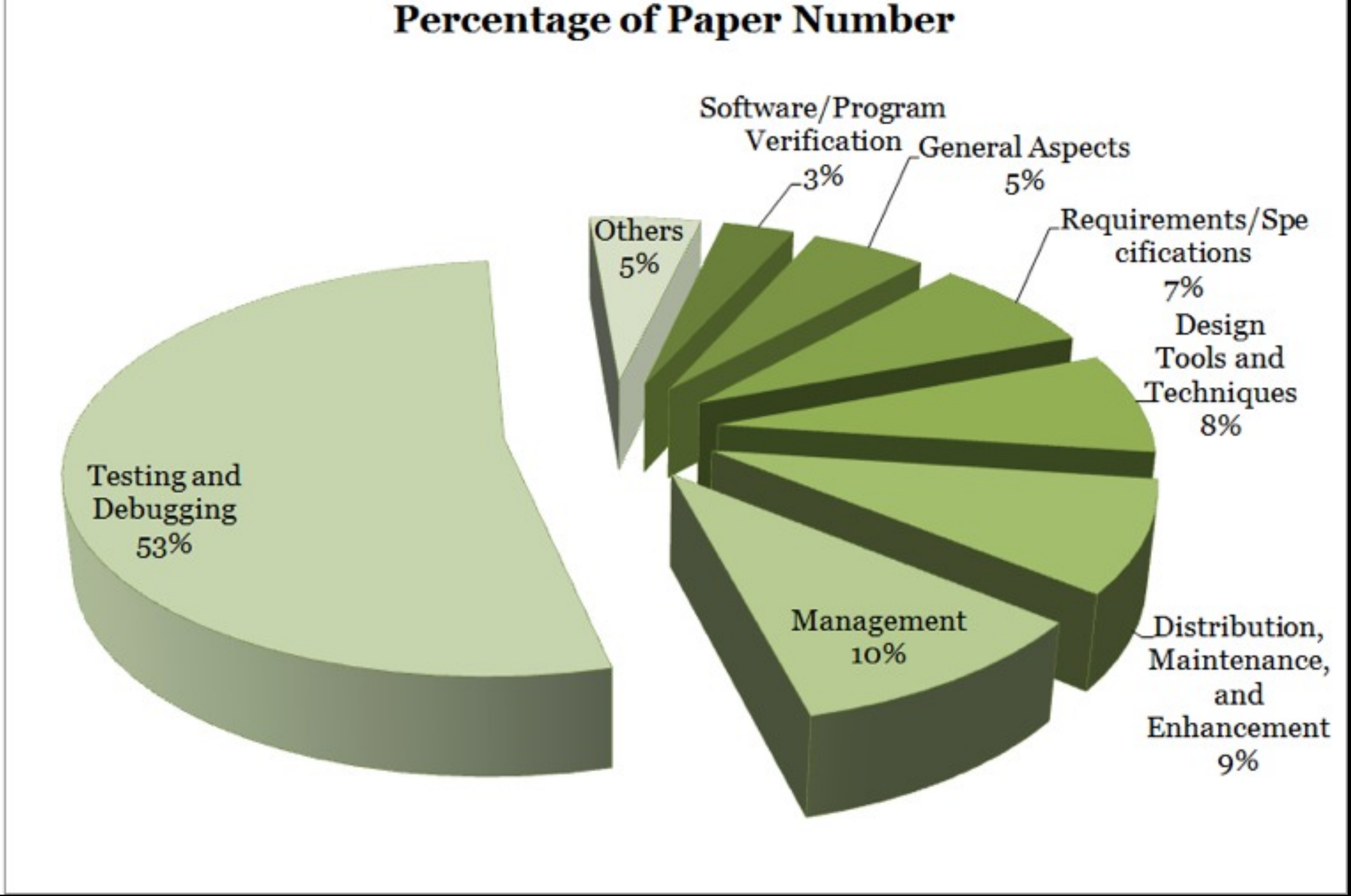
4th International Workshop on

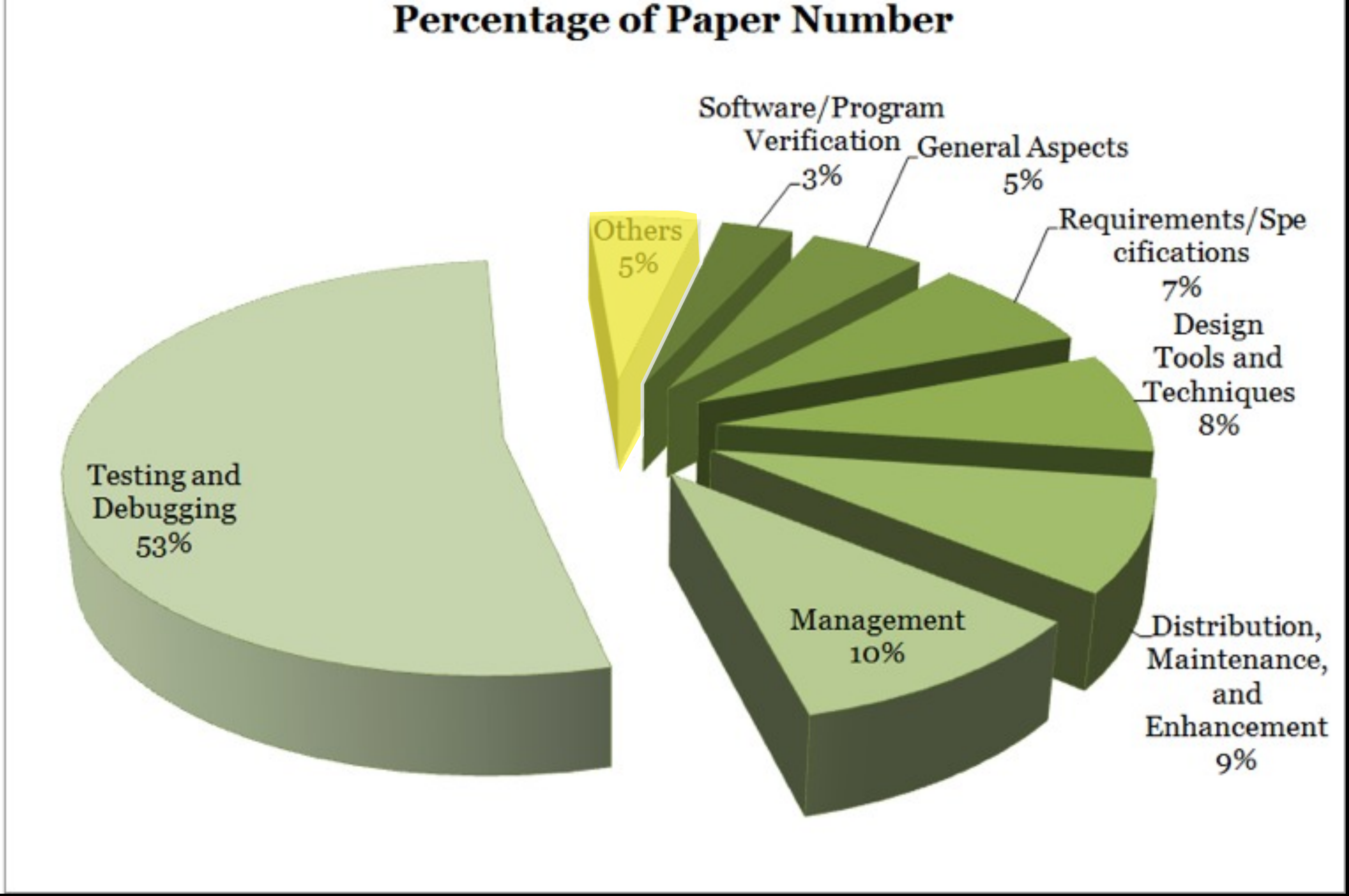
Search-Based Software Testing

March, 2011, Berlin, Germany
In conjunction with ICST 2011
IEEE International Conference on Testing, Verification and Validation

The First Chinese SBSE Workshop

SE Topic coverage


Percentage of Paper Number
Software/Program Verification 3%
General Aspects 5%
Requirements/Specifications 7%
Design Tools and Techniques 8%
Distribution, Maintenance, and Enhancement 9%
Management 10%
Testing and Debugging 53%
Others 5%


Percentage of Paper Number
Software/Program Verification 3%
General Aspects 5%
Requirements/Specifications 7%
Design Tools and Techniques 8%
Distribution, Maintenance, and Enhancement 9%
Management 10%
Testing and Debugging 53%
Others 5%

DAASE

Just some of the many SBSE applications

Just some of the many SBSE applications

Agent Oriented
Aspect Oriented
Assertion Generation
Bug Fixing
Component Oriented
Design
Effort Estimation
Heap Optimisation
Model Checking
Predictive Modelling
Probe distribution
Program Analysis
Program Comprehension
Program Transformation
Project Management
Protocol Optimisation
QoS
Refactoring
Regression Testing
Requirements
Reverse Engineering
SOA
Software Maintenance and Evolution
Test Generation
UIO generation

Tutorial Paper

Mark Harman, Phil McMinn, Jerffeson Teixeira de Souza and Shin Yoo.
Search Based Software Engineering: Techniques, Taxonomy, Tutorial.

in LNCS 7007.
Editors: Bertrand Meyer and Martin Nordio.

google: search based software engineering tutorial

PDF also freely available on my website

Dynamic Adaptive SBSE

Compile SBSE into deployed Software

Dynamic Adaptive SBSE

Compile SBSE into deployed Software

Dynamic Adaptive SBSE

Compile SBSE into deployed Software

functional vs. non functional

Requirements

Functional Requirements

Non-Functional Requirements

Functional Requirements

Non-Functional Requirements

Execution Time

Memory

Bandwidth

Battery

Size

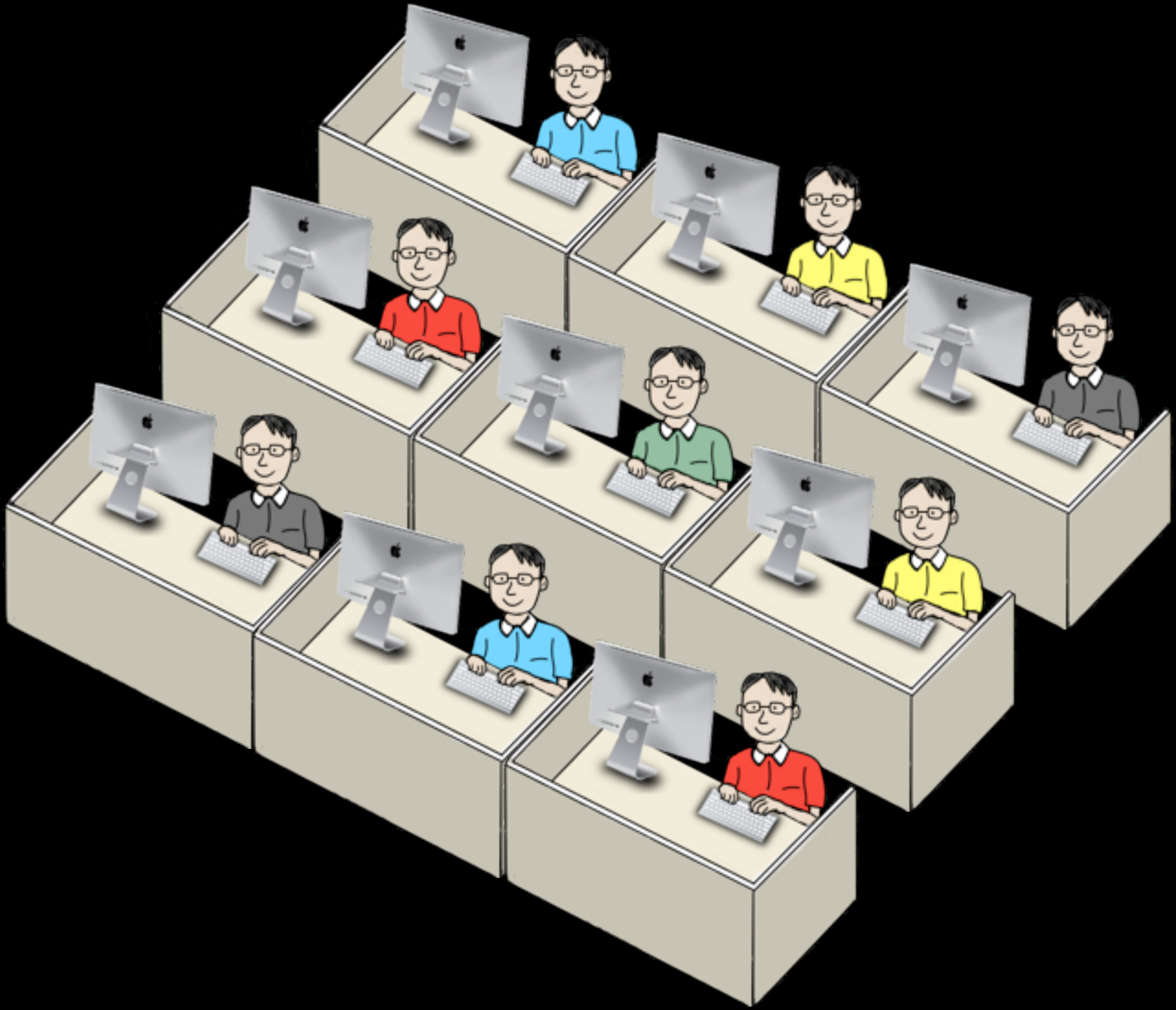
Functional Requirements

functionality of the Program

Non-Functional Requirements

- Execution Time
- Memory
- Bandwidth
- Battery
- Size

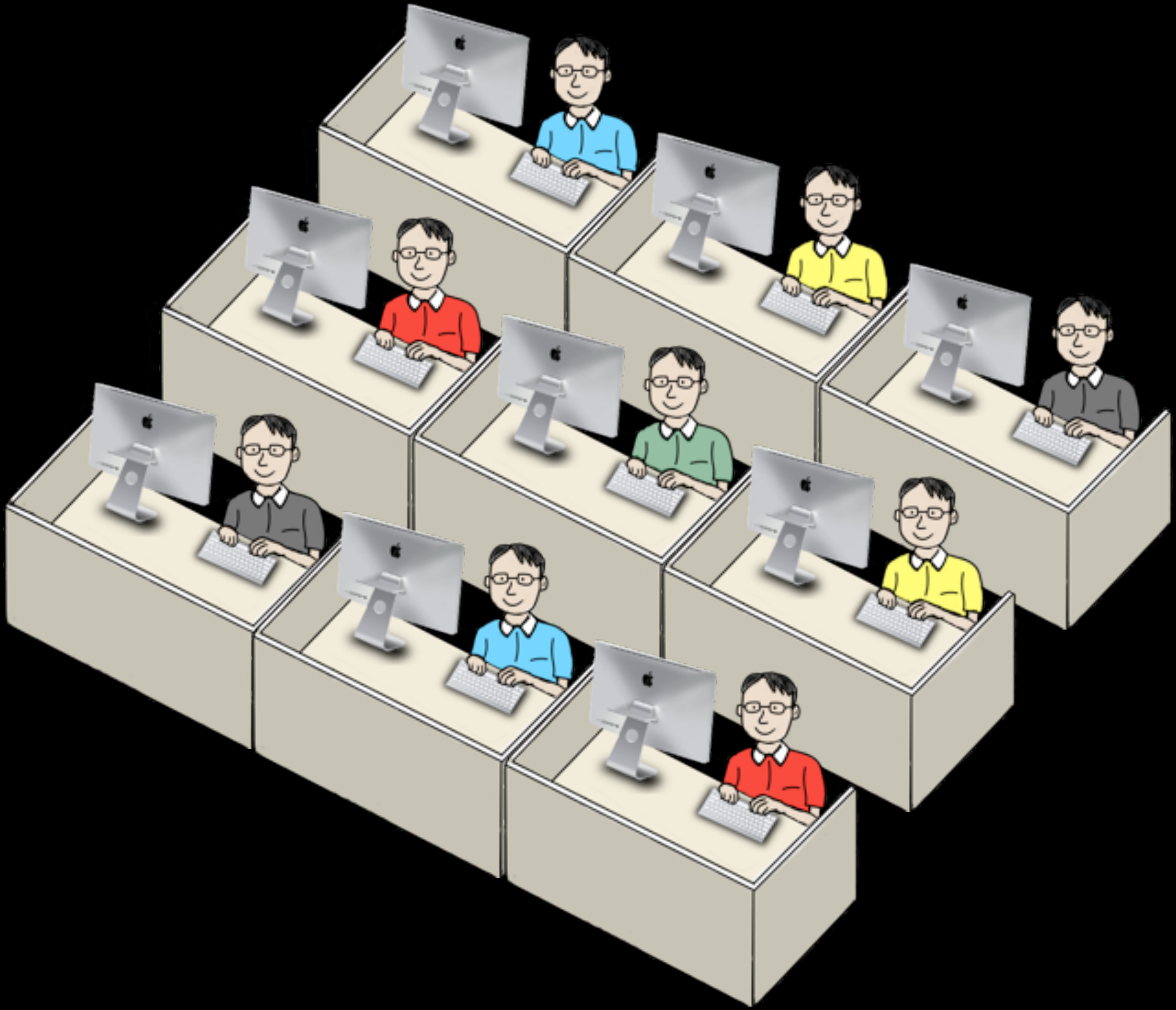
Software Design Process



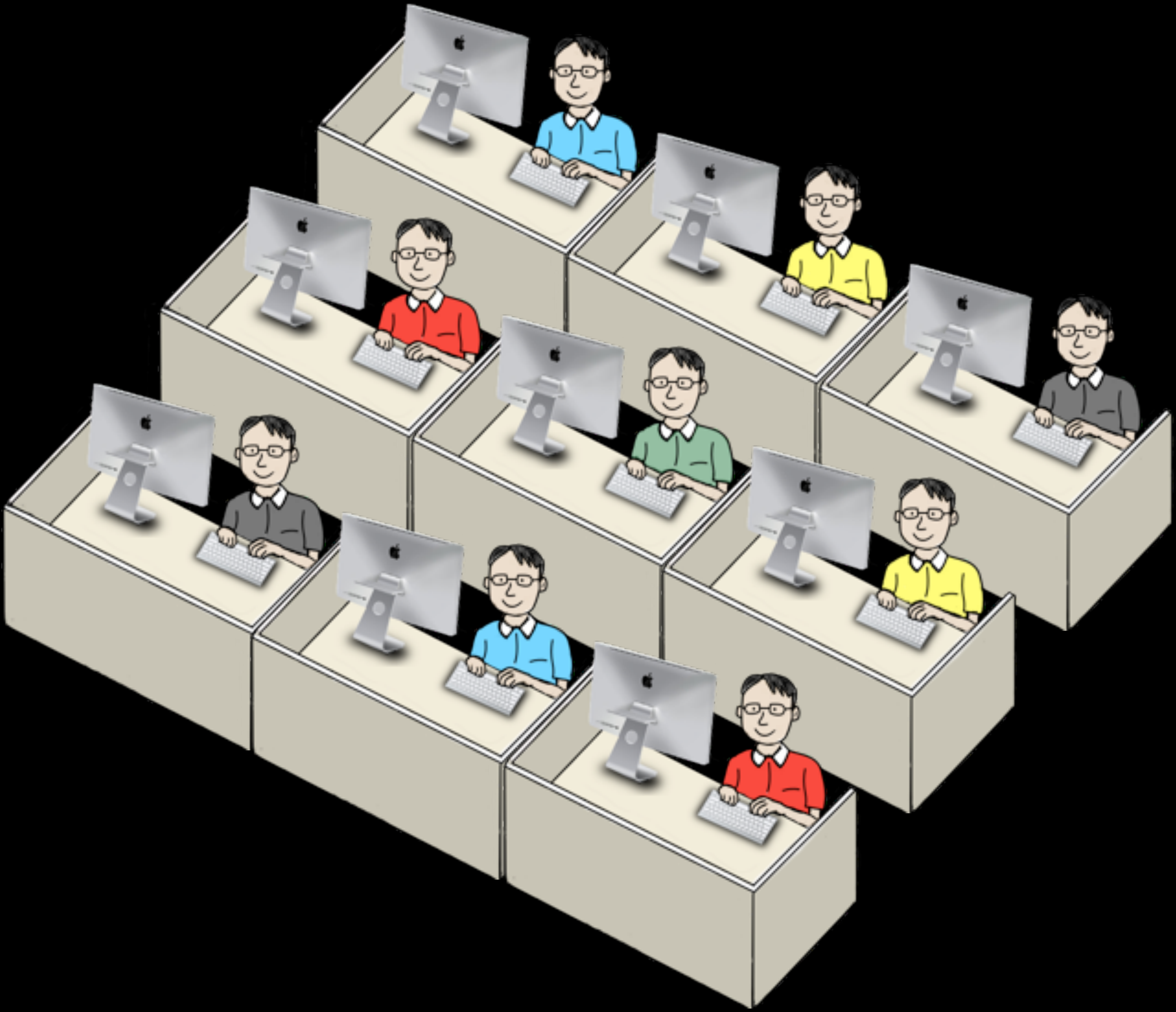
DAASE

Mark Harman, CREST

Software Design Process



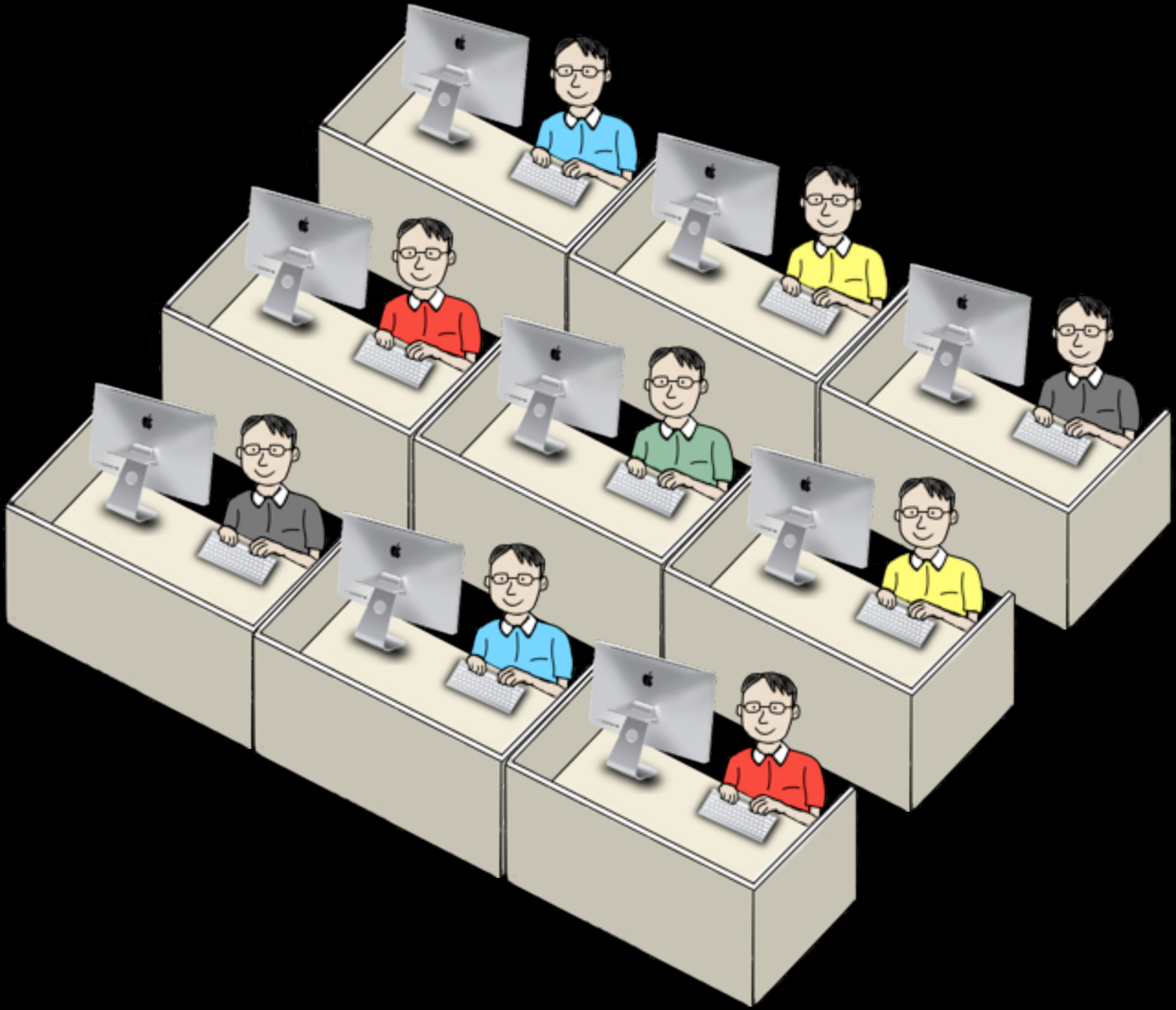
Software Design Process



DAASE

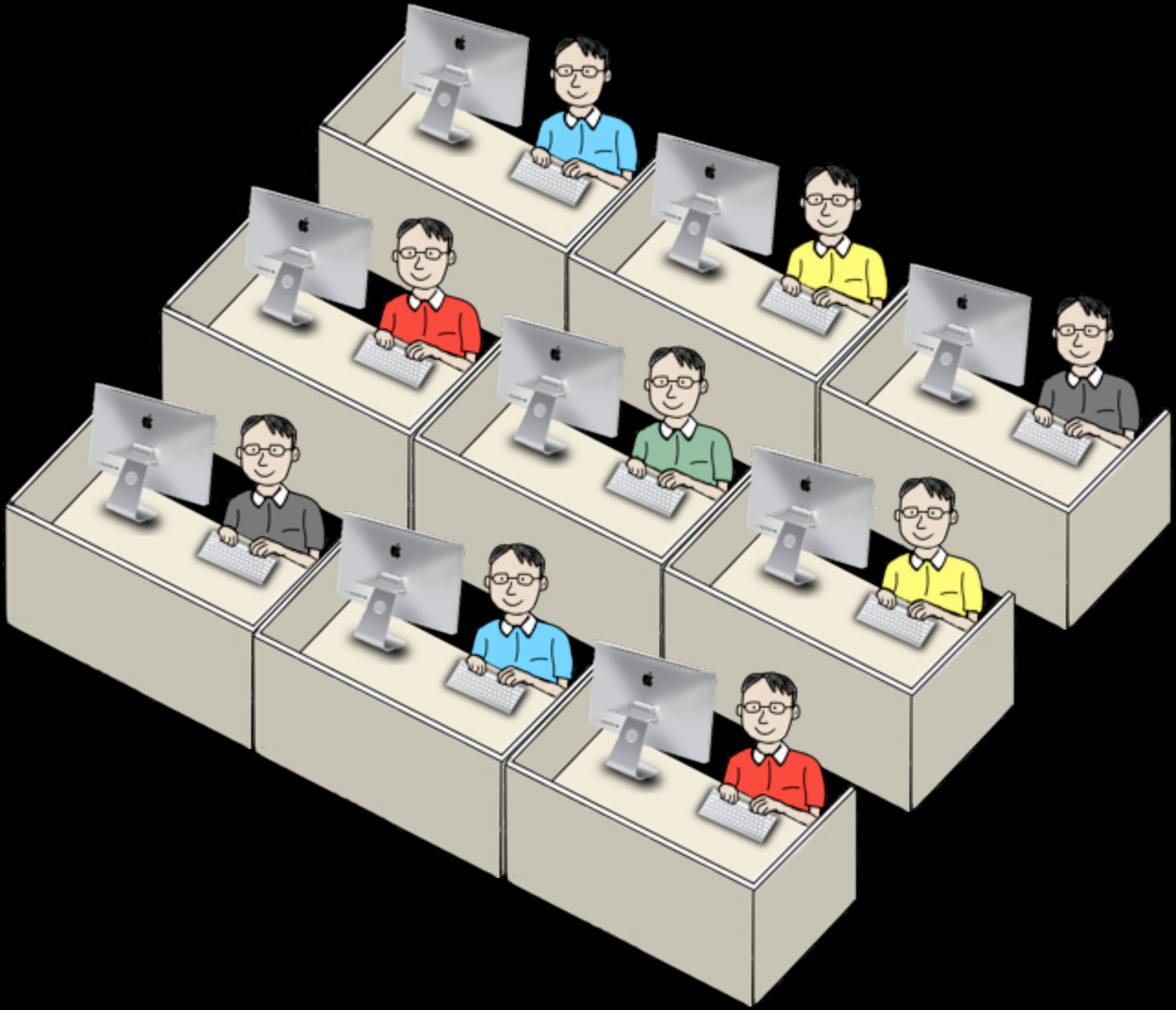
Mark Harman, CREST

Software Design Process

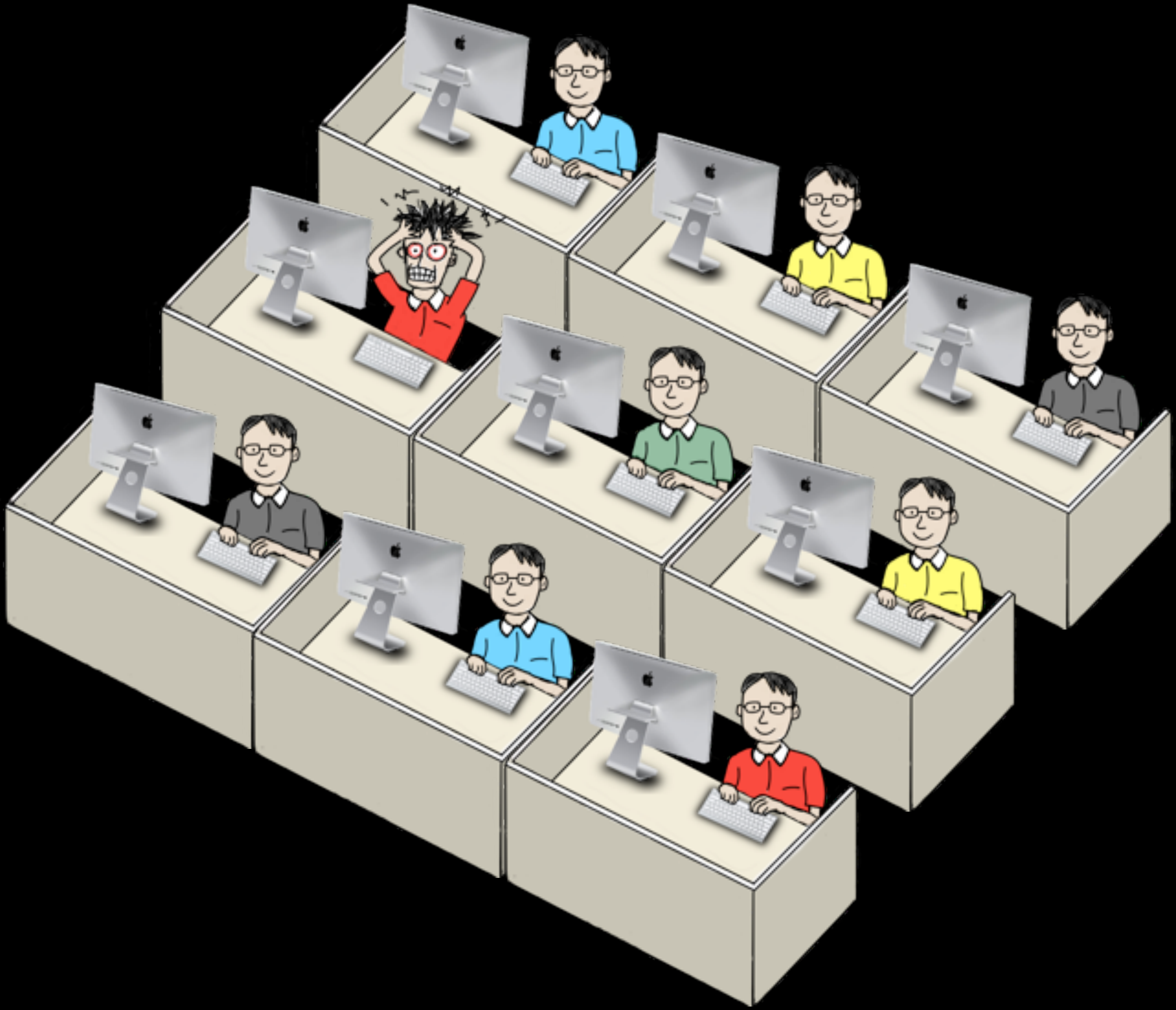


DAASE

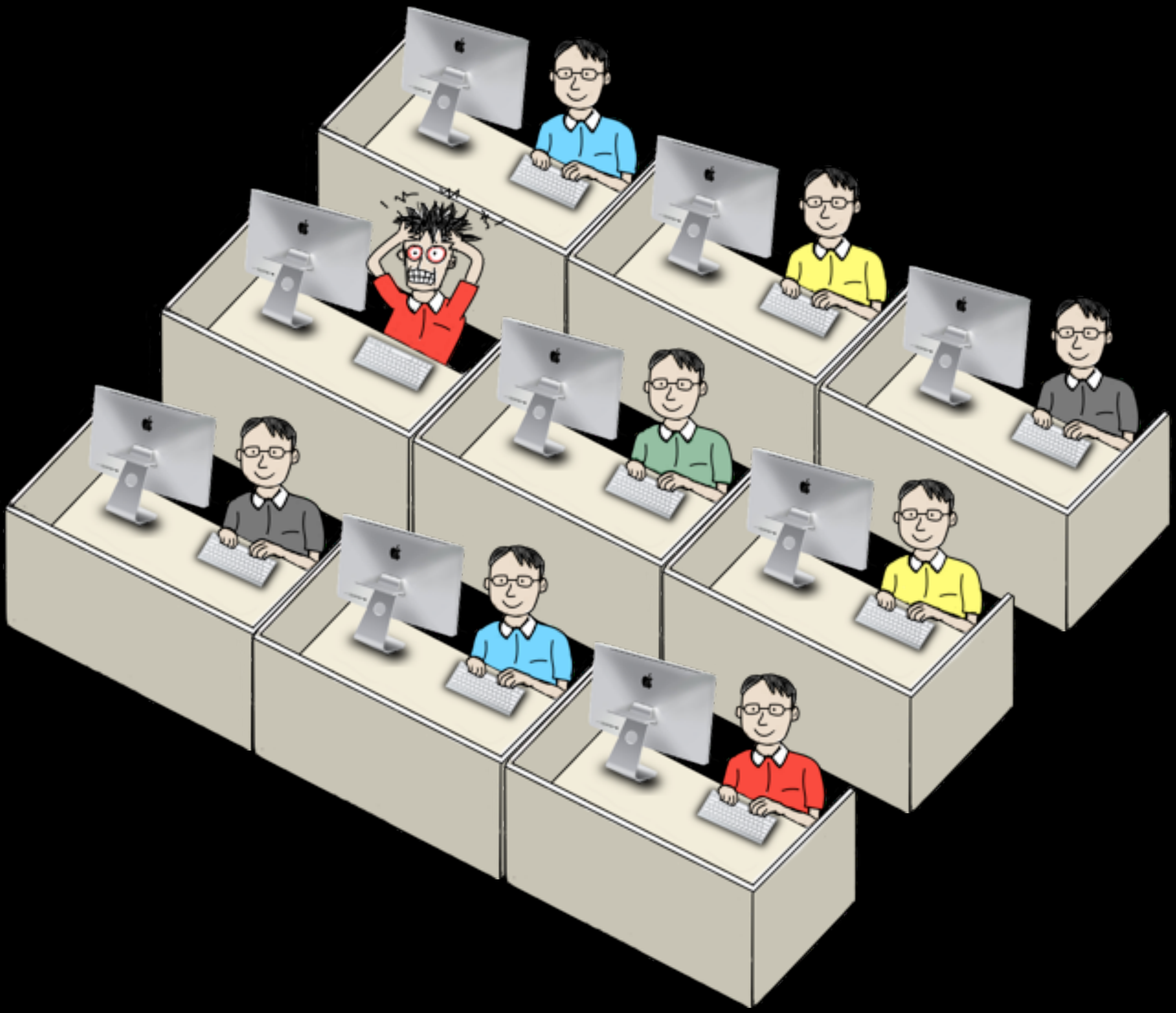
Software Design Process



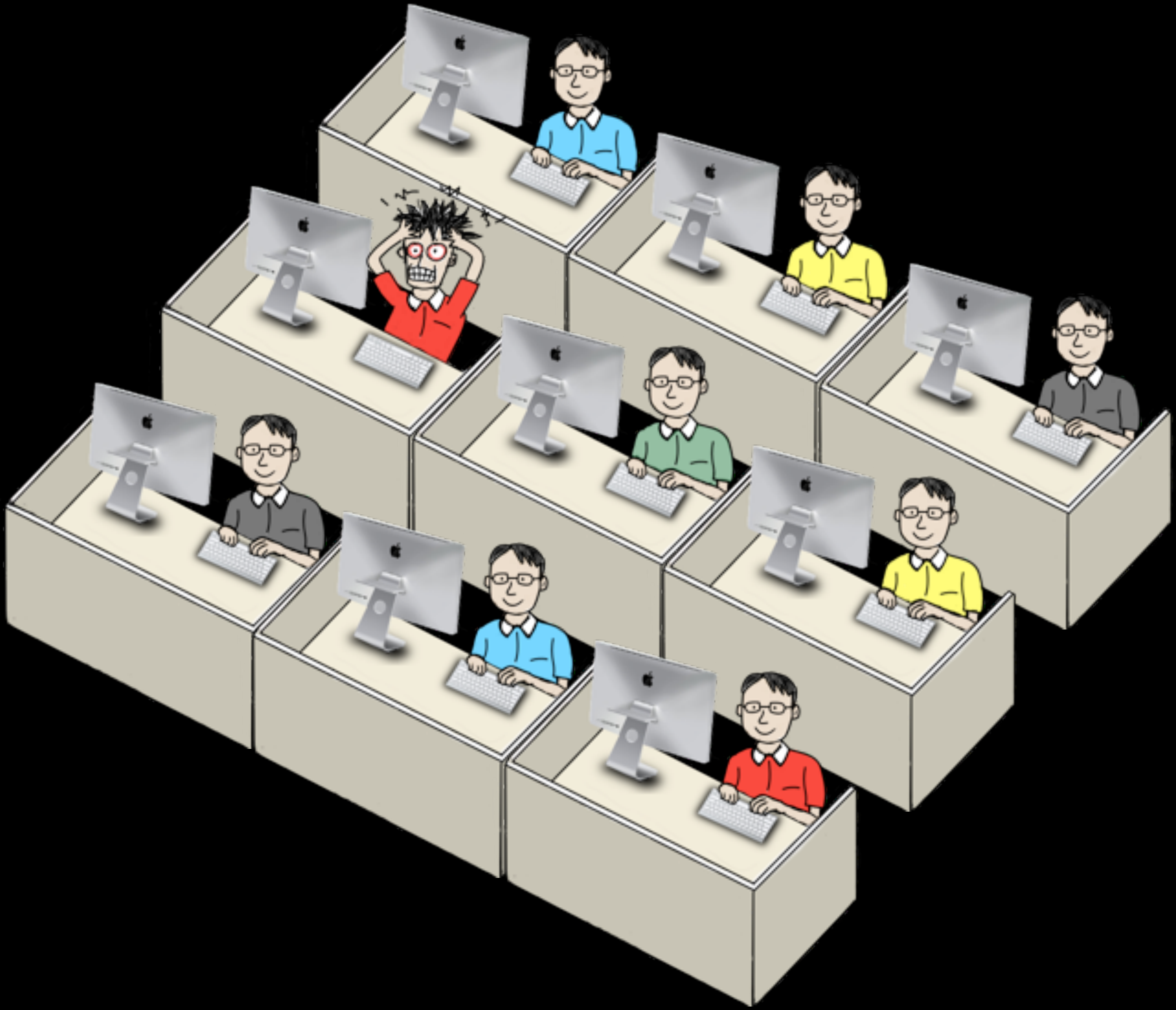
Software Design Process



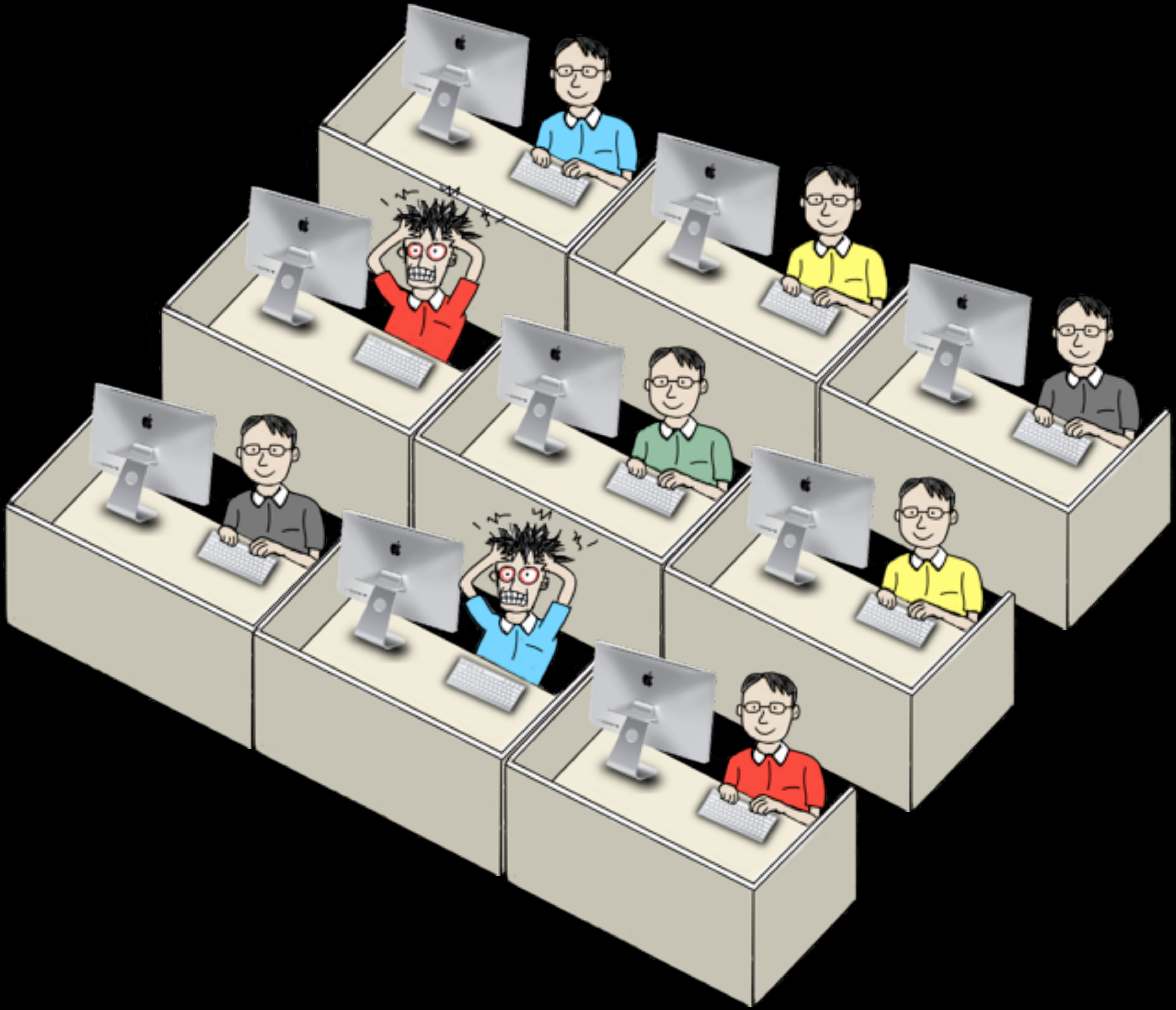
Software Design Process



Software Design Process

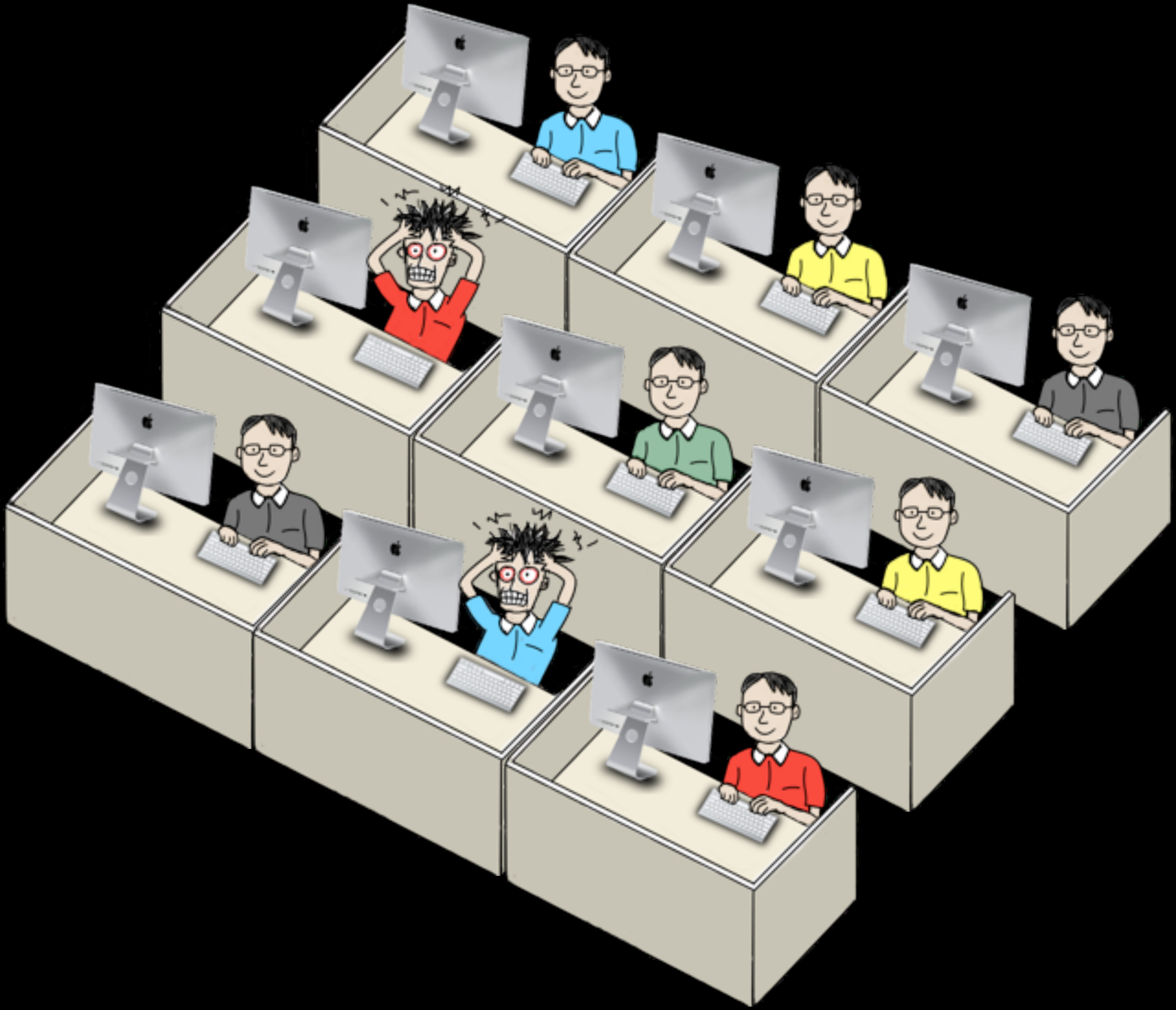


Software Design Process

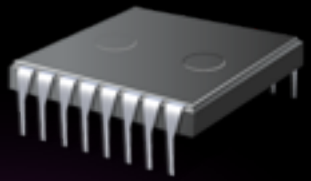


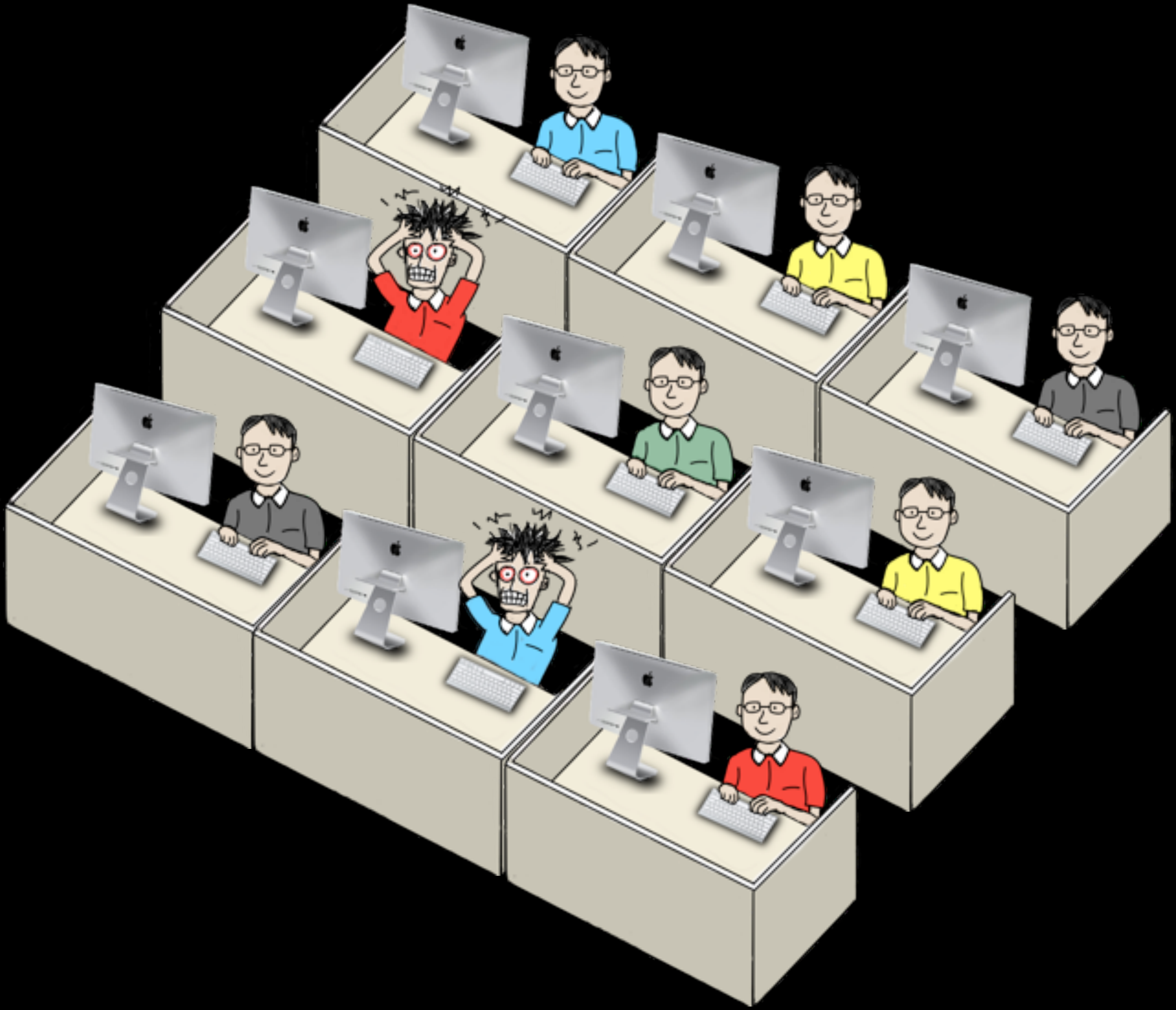
Mark Harman, CREST

Software Design Process

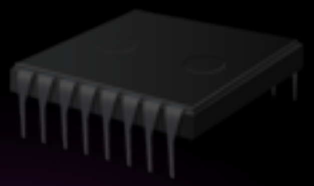


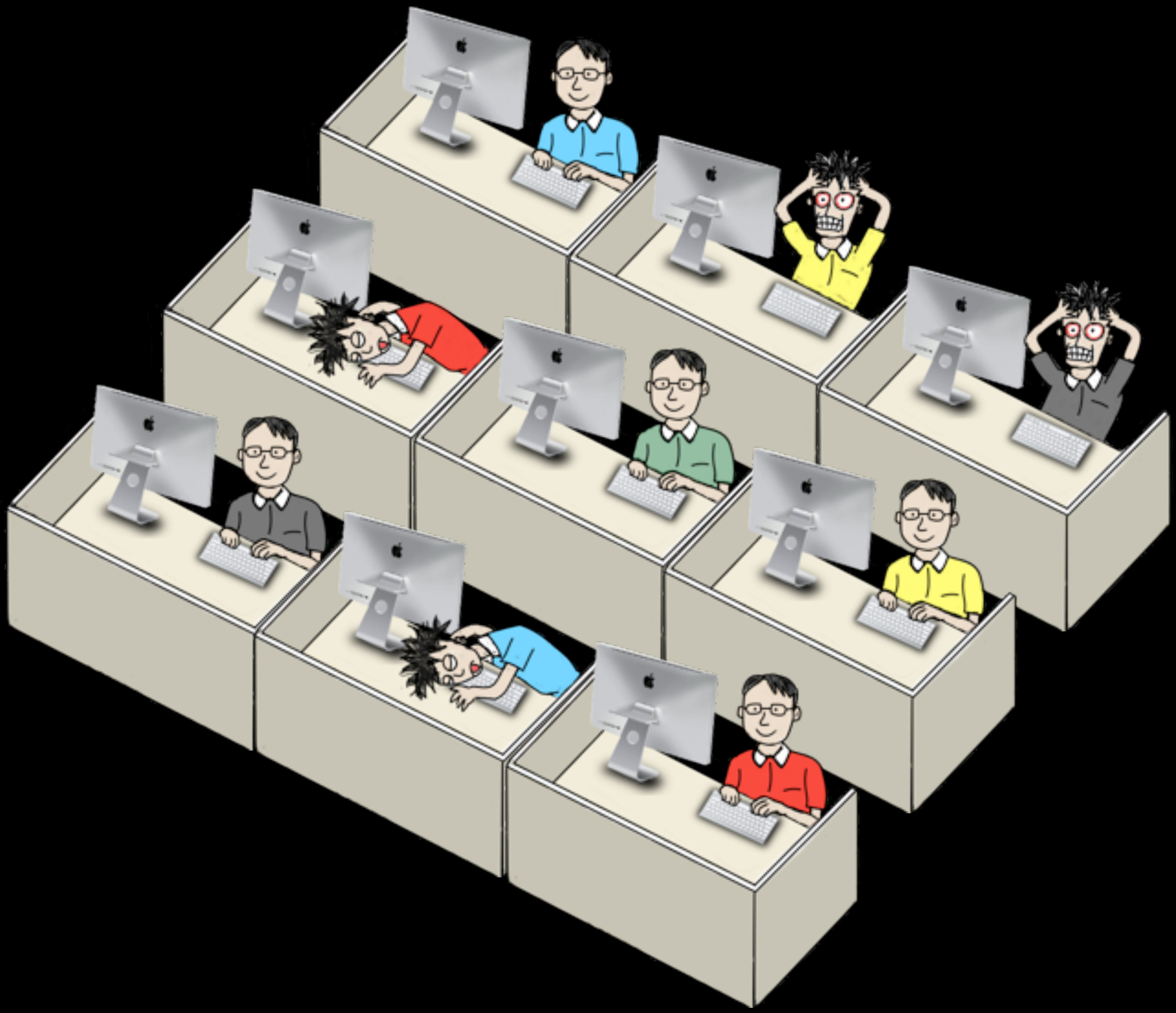
Software Design Process



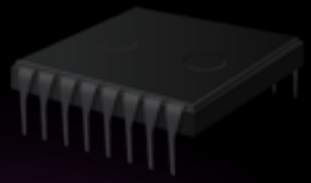


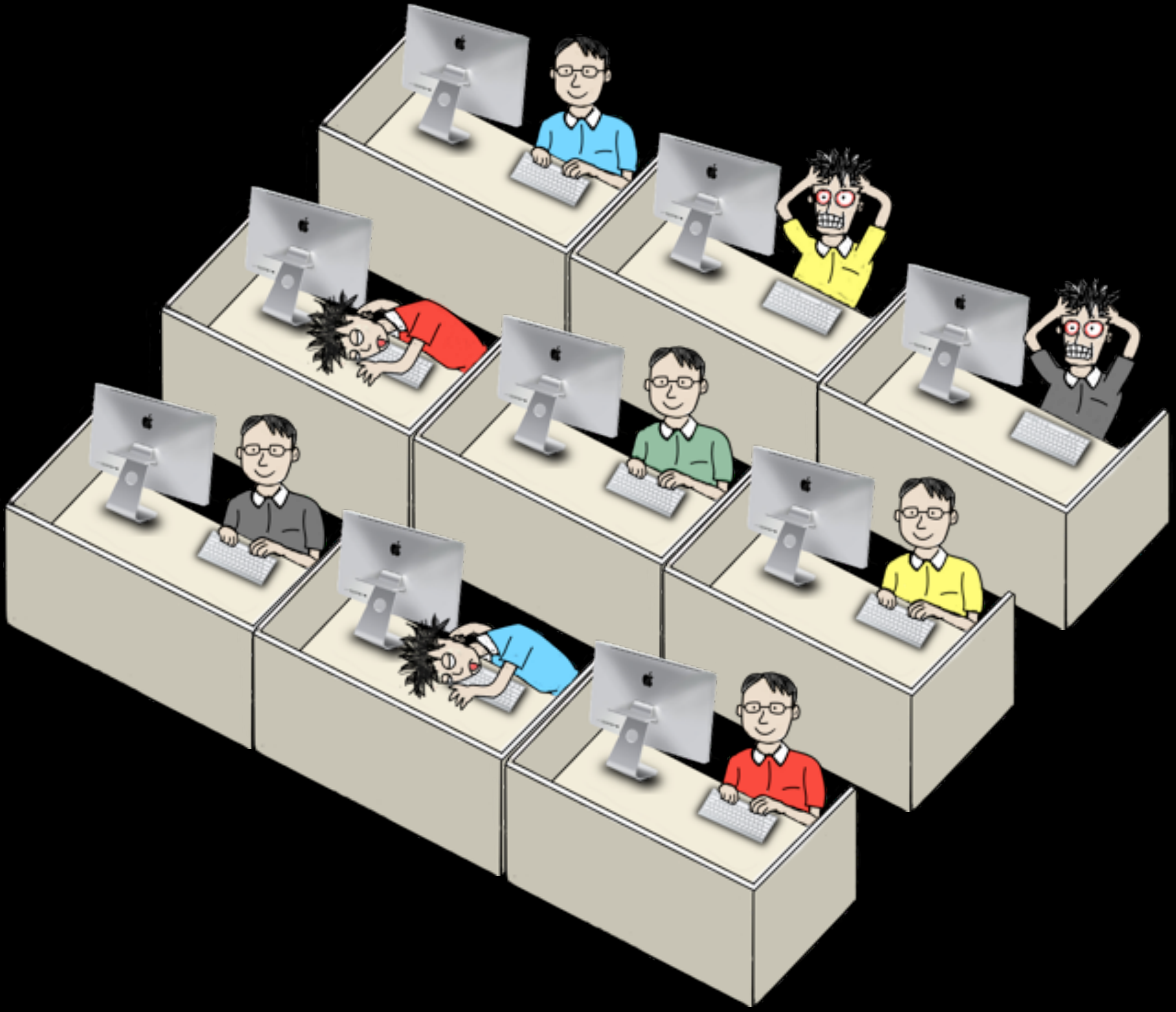
Software Design Process



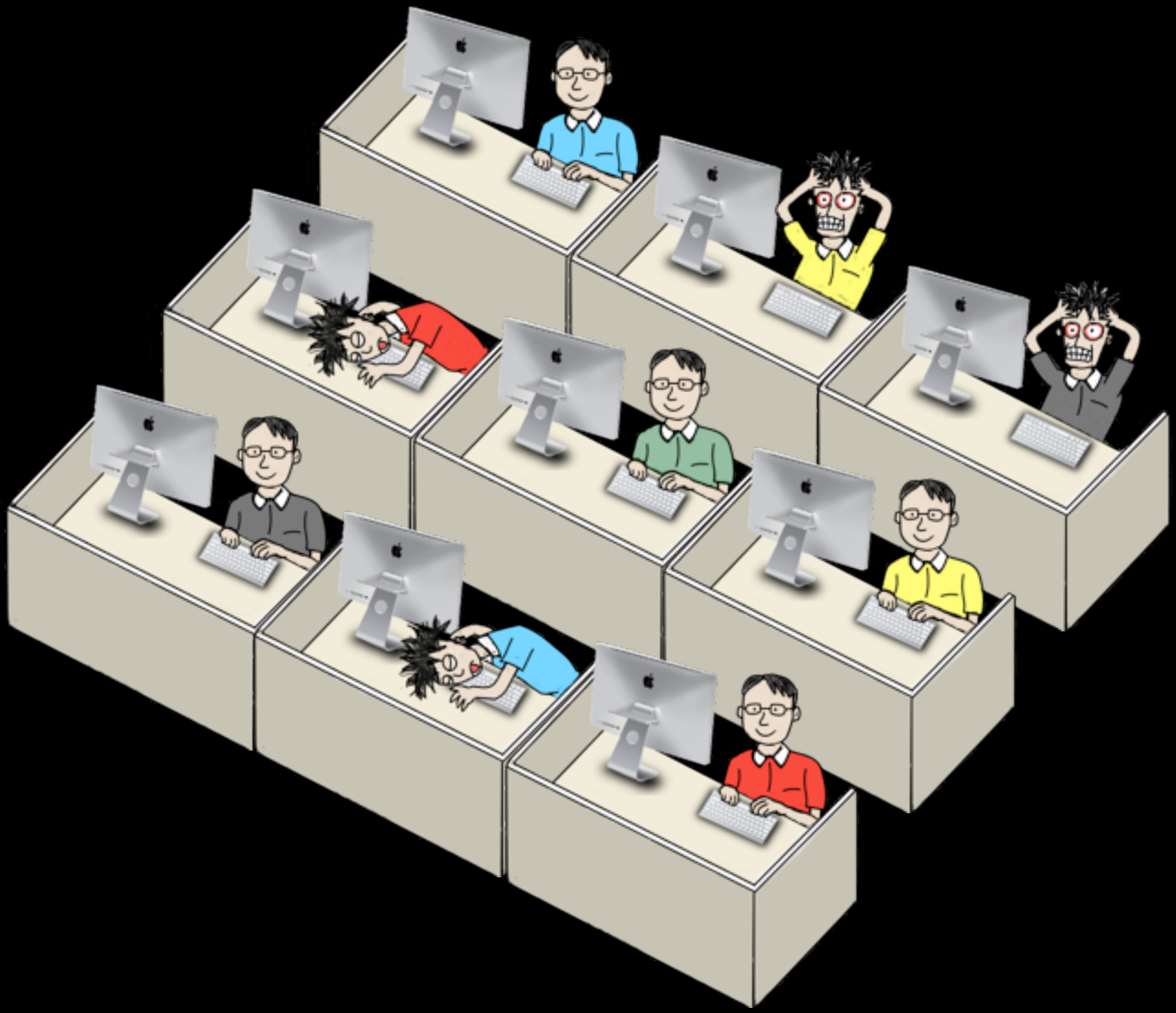


Software Design Process

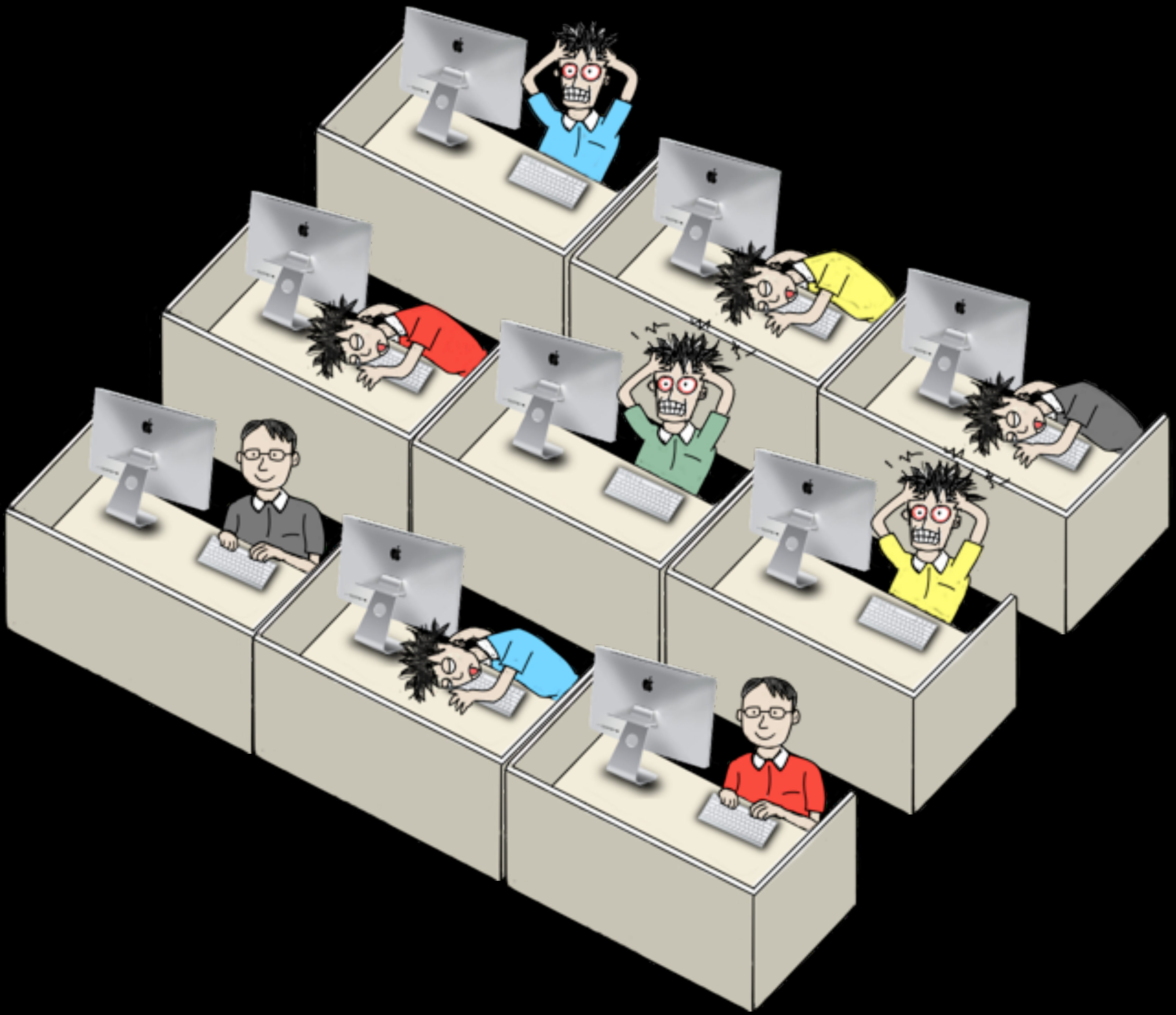




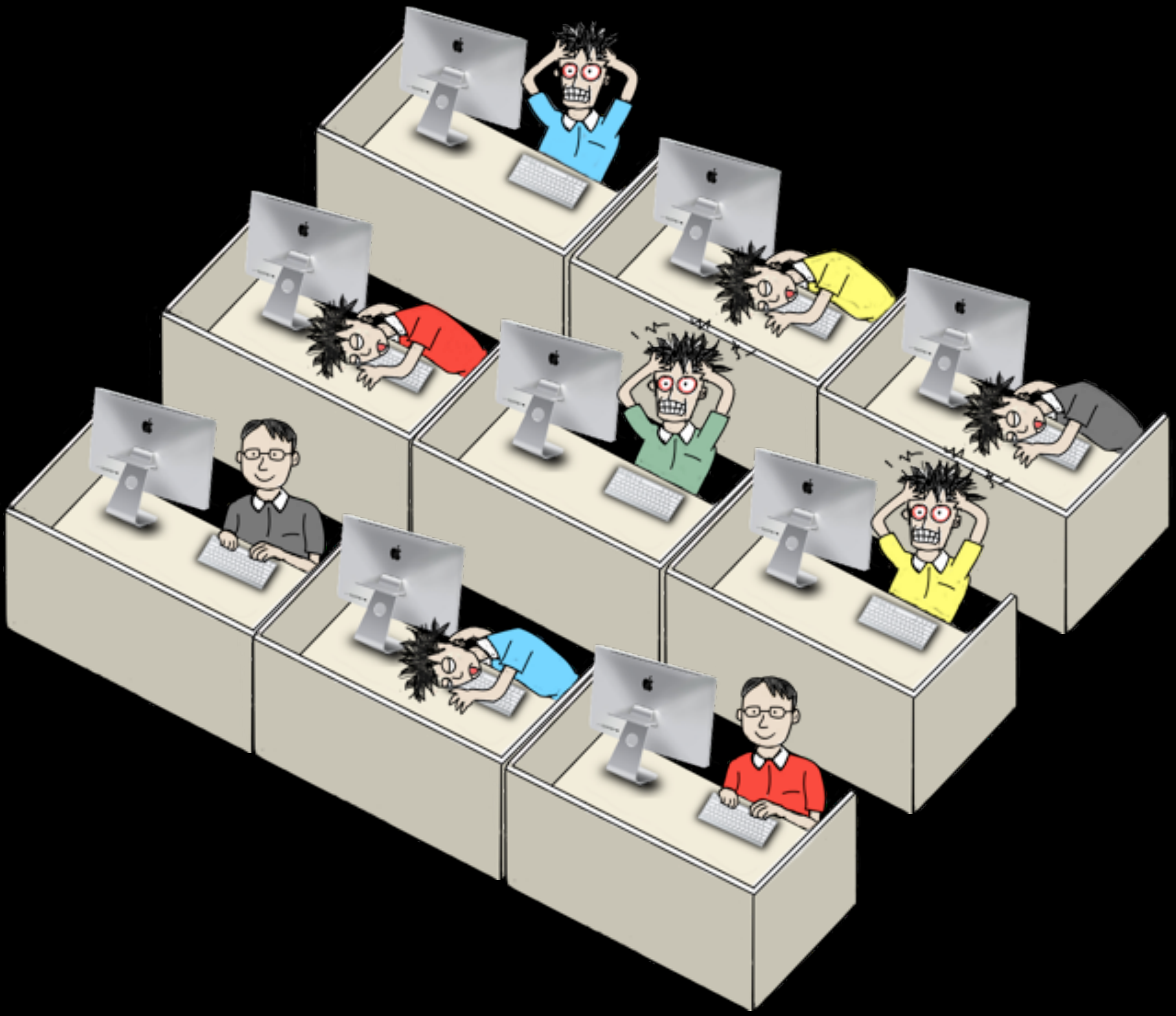
Software Design Process



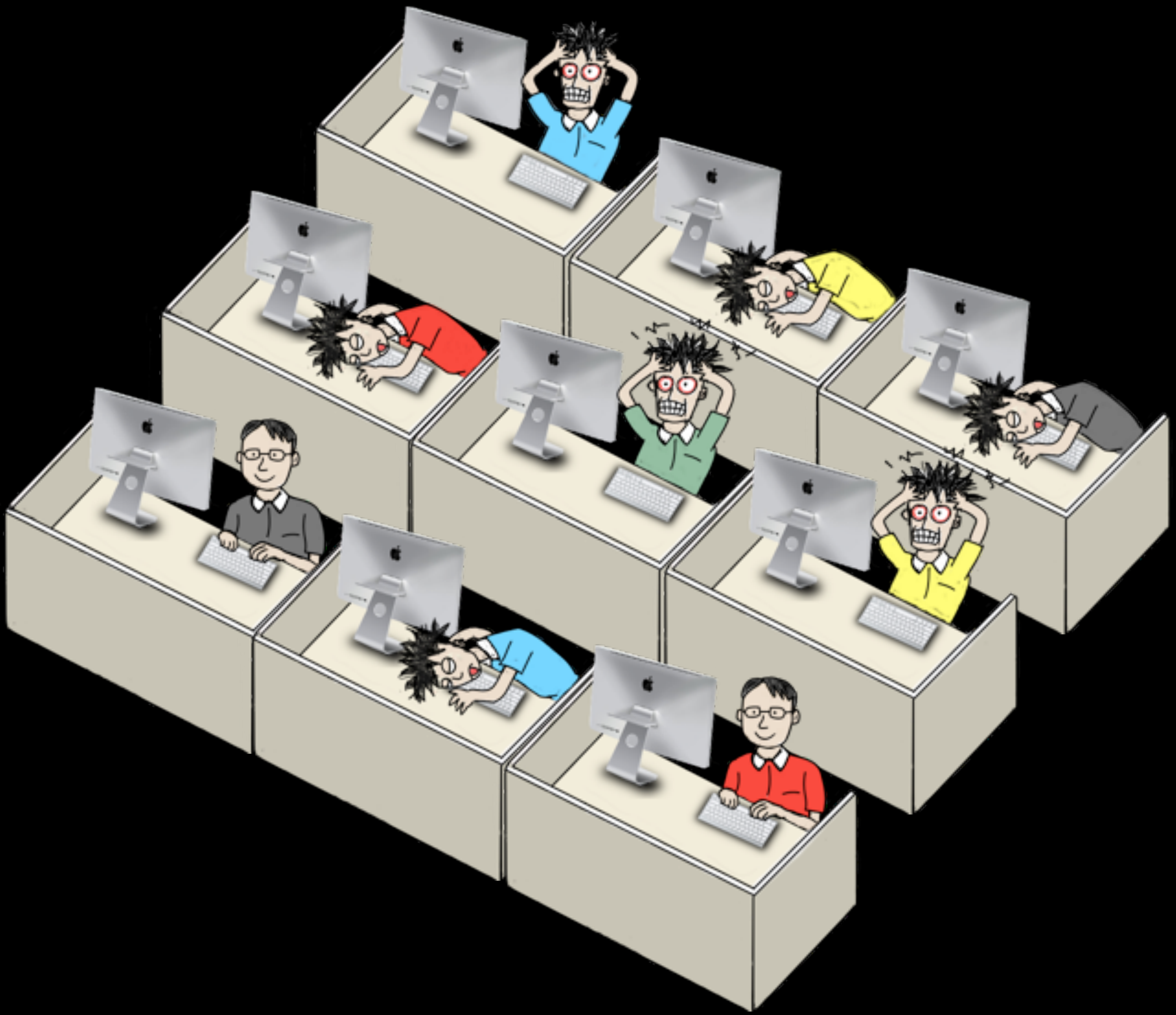
Software Design Process



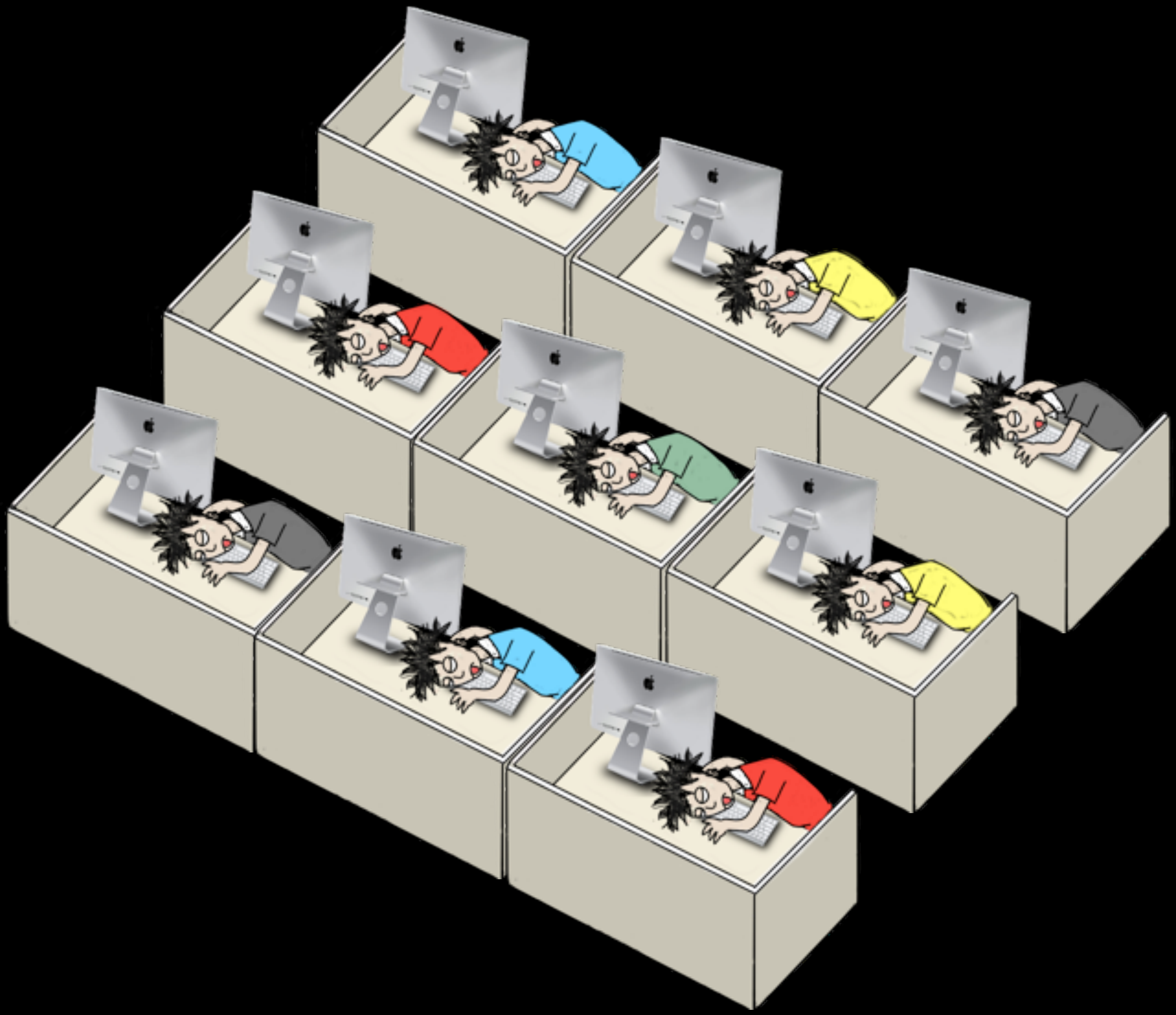
Software Design Process



Software Design Process

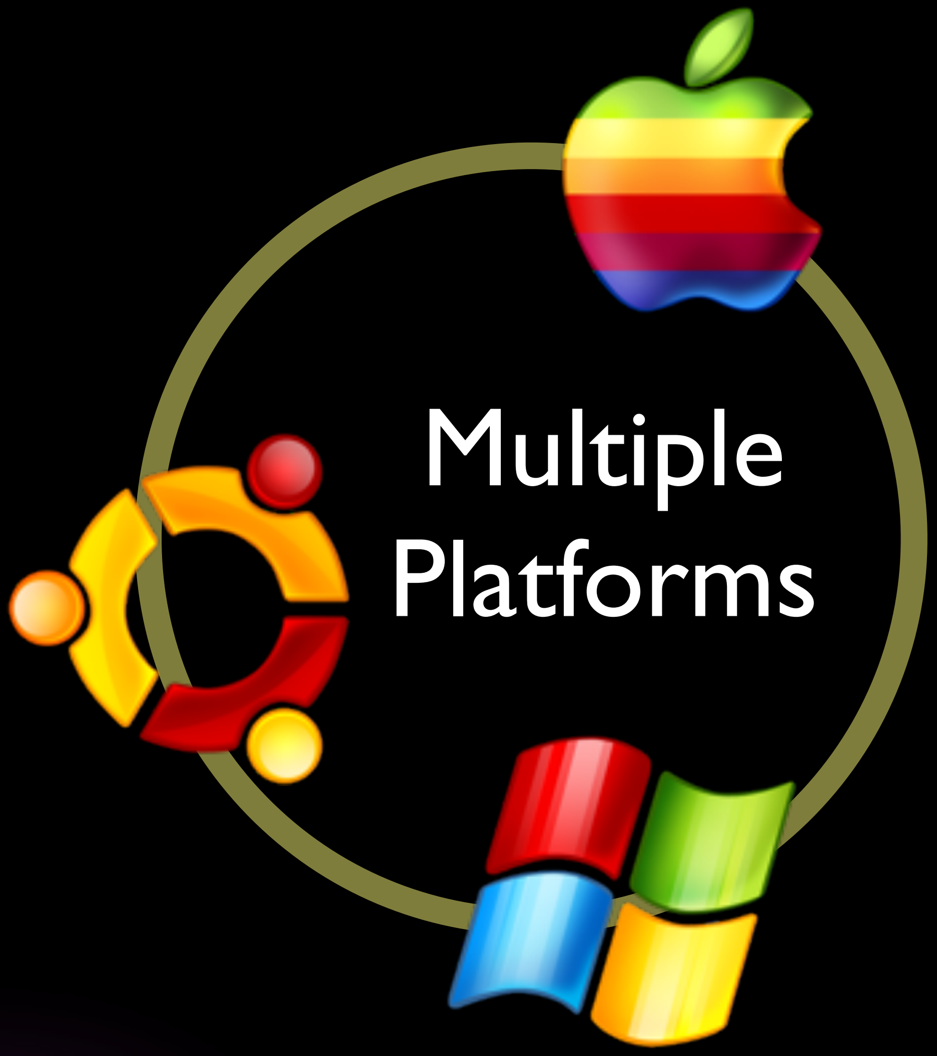


Software Design Process



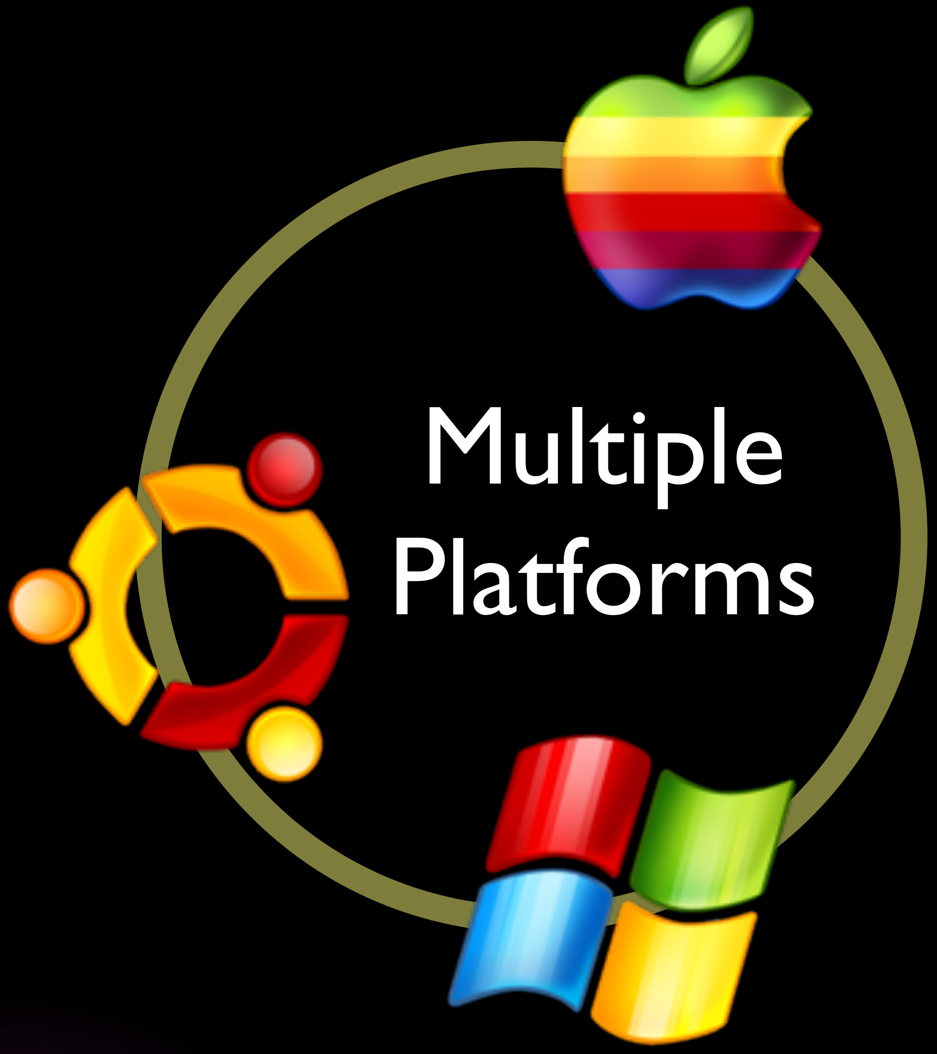
Multiplicity

Multiplicity



Multiple Platforms

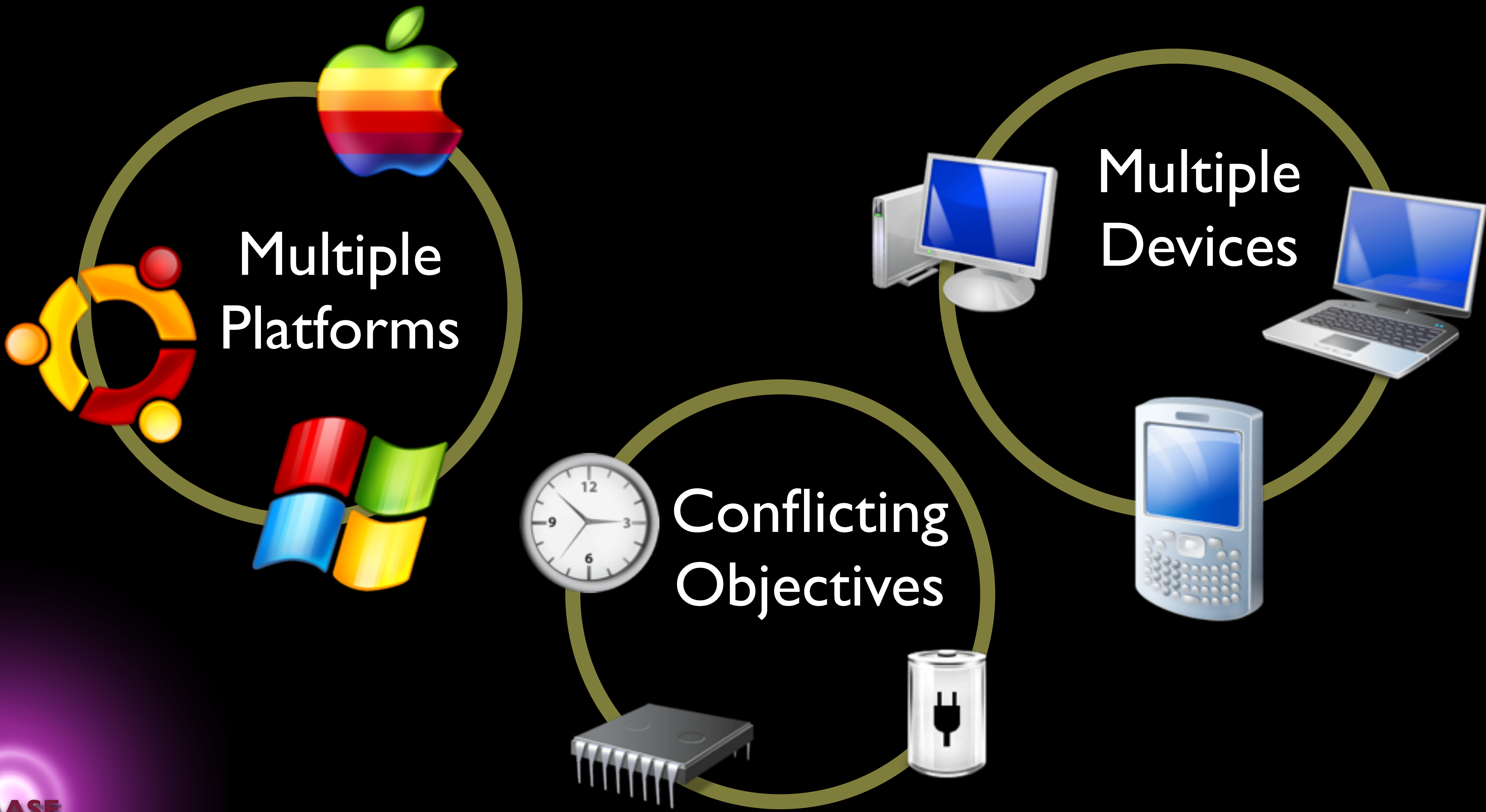
Multiplicity



Multiple Platforms

Multiple Devices

Multiplicity



Multiple Platforms

Conflicting Objectives

Multiple Devices

Why is the programmer human?

DAASE

Which requirements must be human coded ?

Which requirements must be human coded ?

Functional Requirements

Non-Functional Requirements

Which requirements must be human coded ?

Functional Requirements

Non-Functional Requirements

humans have to define these

Which requirements must be human coded ?

Functional Requirements

humans have to define these

Non-Functional Requirements

a machine can optimise these

Which requirements are essential to human ?

Functional Requirements

Non-Functional Requirements

humans have to define these

a machine can optimise these

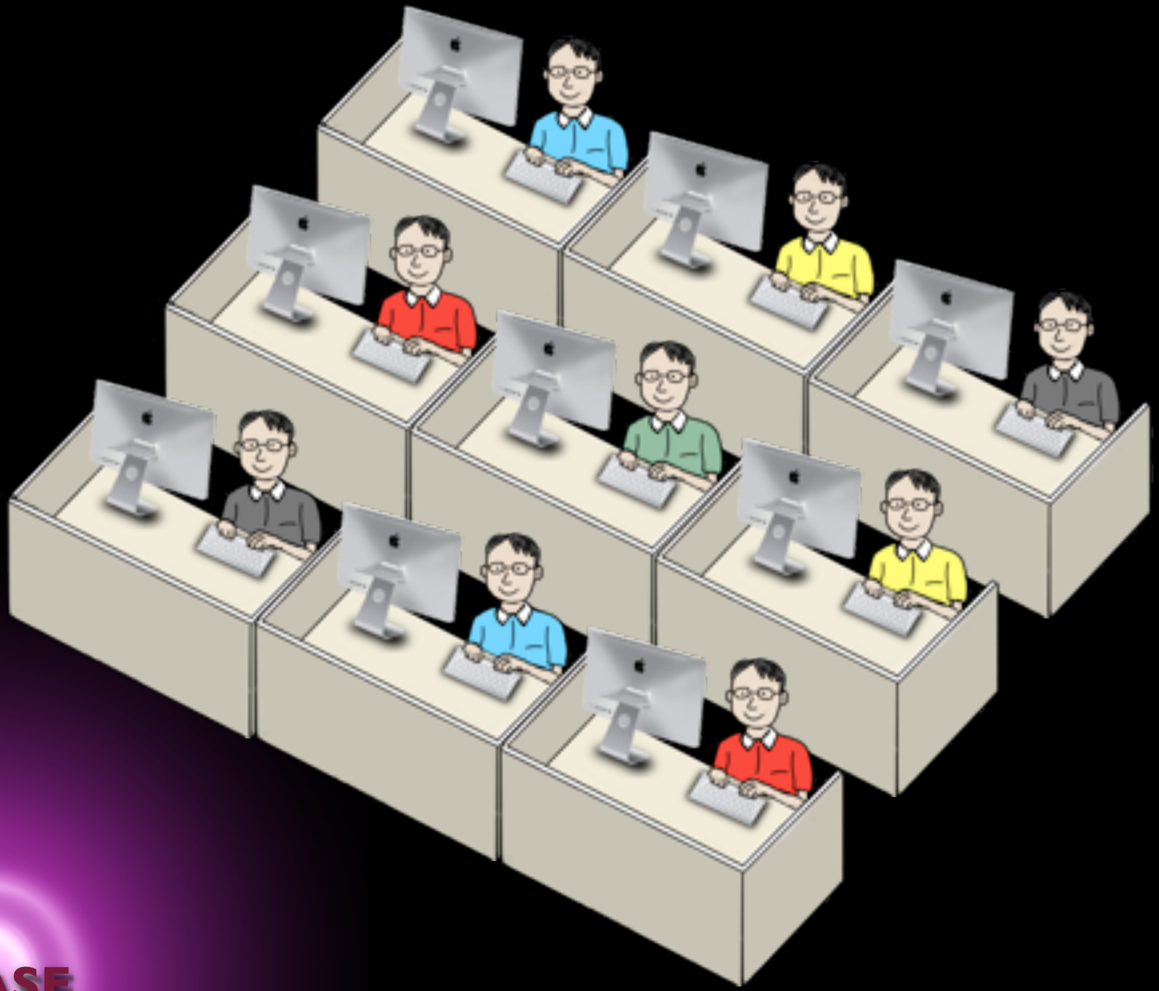
Pickering's Harem

Pickering's Harem

This is what computers looked like 100 years ago

Pickering's Harem

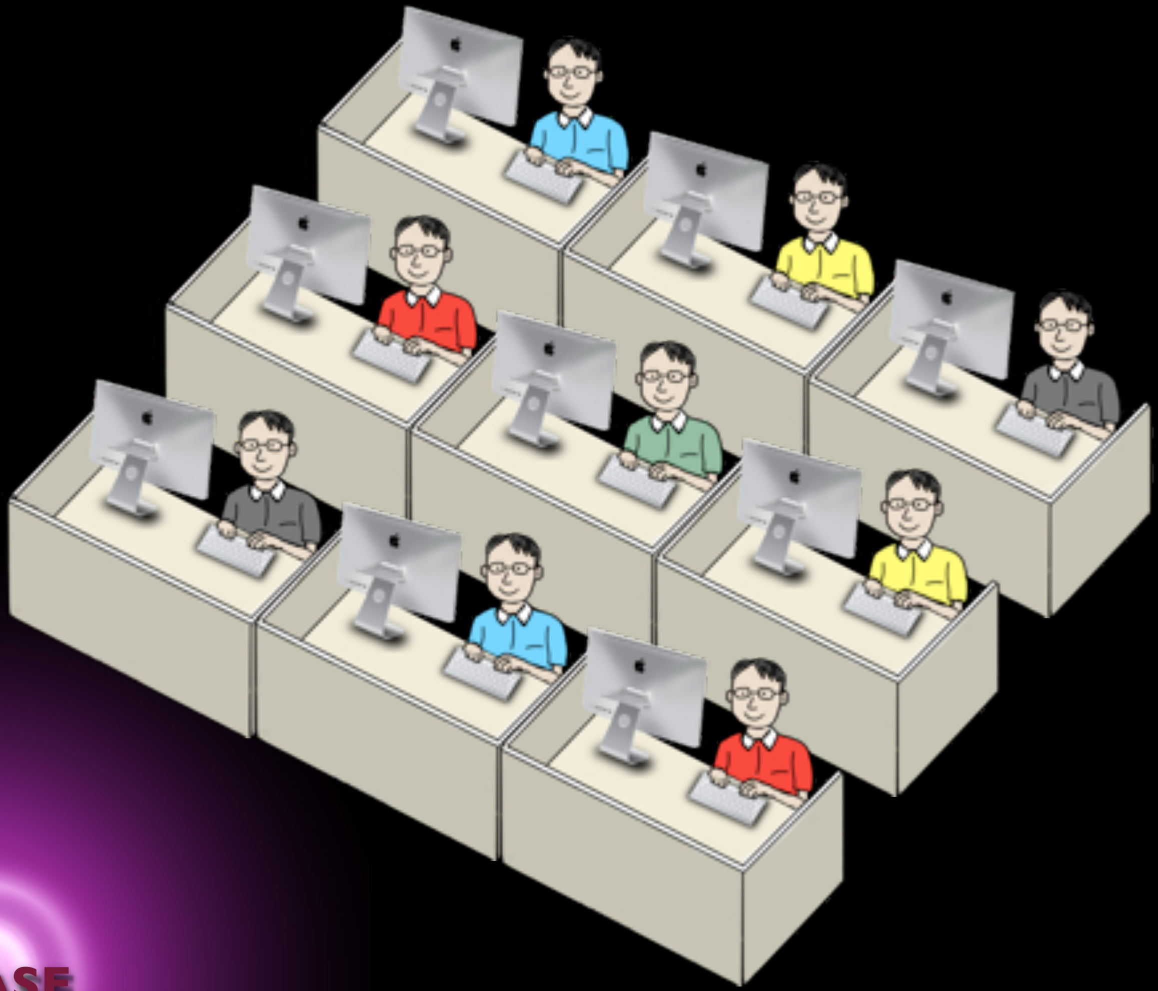
This is what computers looked like 100 years ago



Dilbert's Cube Farm

Pickering's Harem

This is what computers looked like 100 years ago

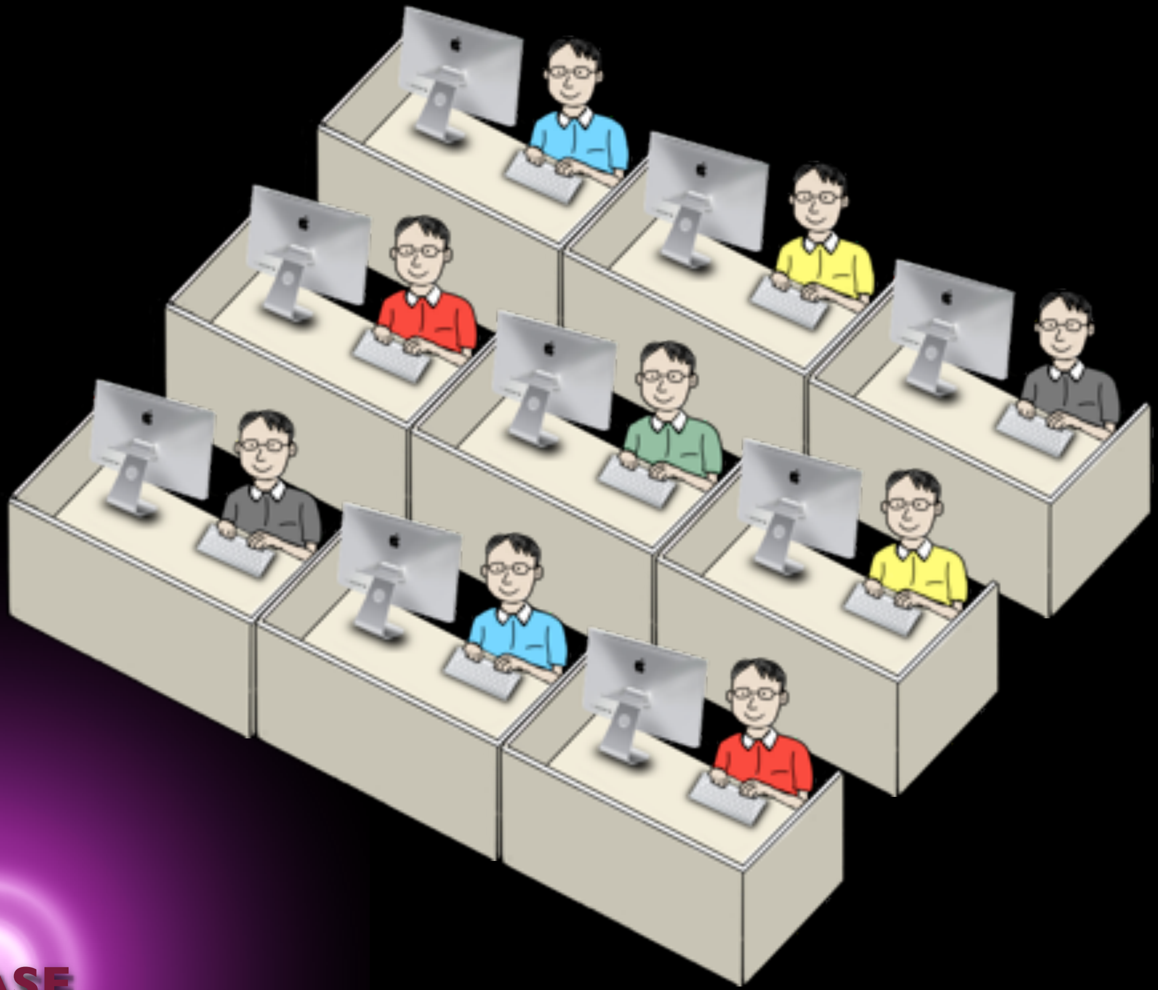


Dilbert's Cube Farm

This is what programmers look like today

DAASE

Computers ... ?



Programmers ... ?

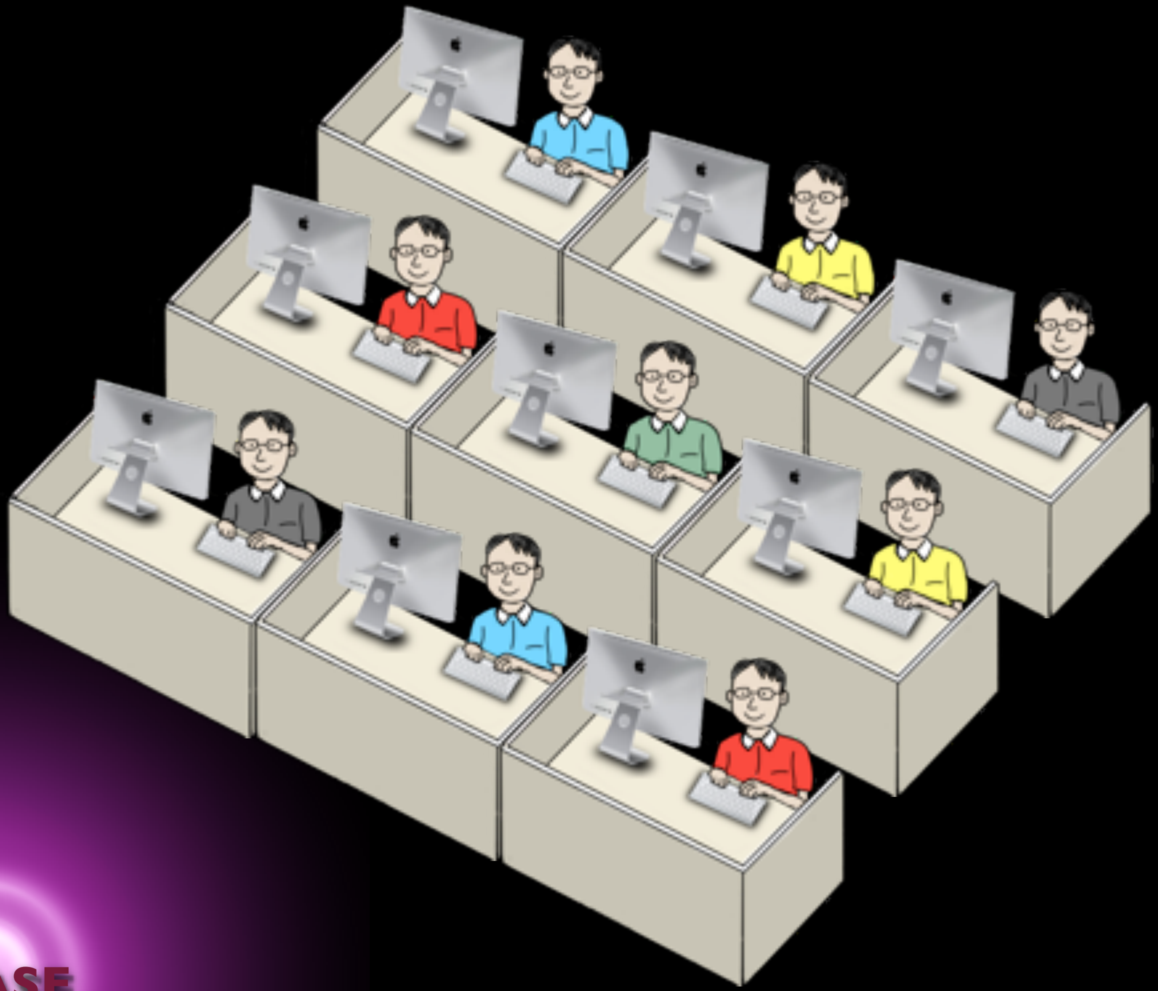
DAASE

Mark Harman, CREST

Saturday, 16 February 13

Computers ... ?

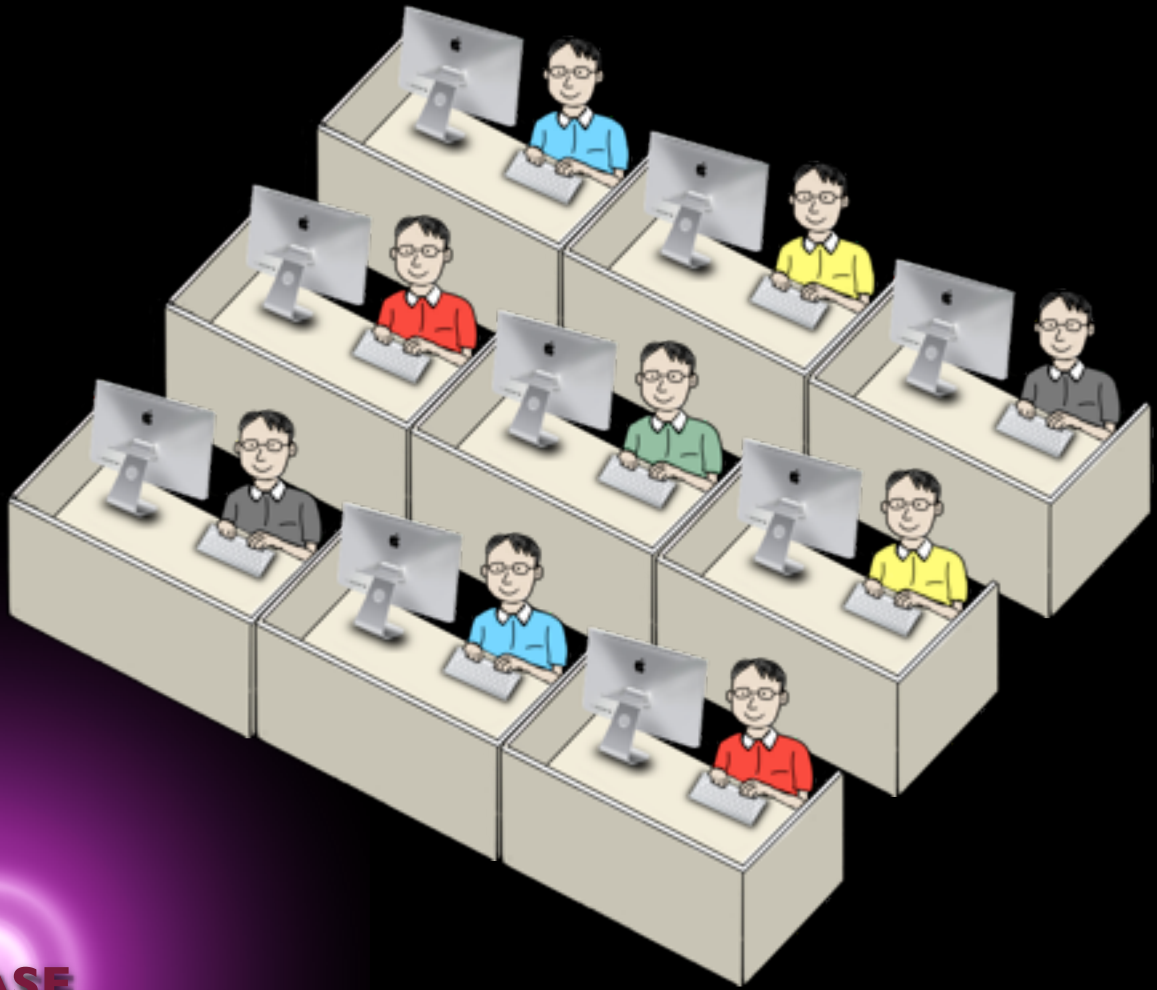
how quaint!



Programmers ... ?

Computers ... ?

how quaint!



Programmers ... ?

how quaint!

Dynamic Adaptive SBSE

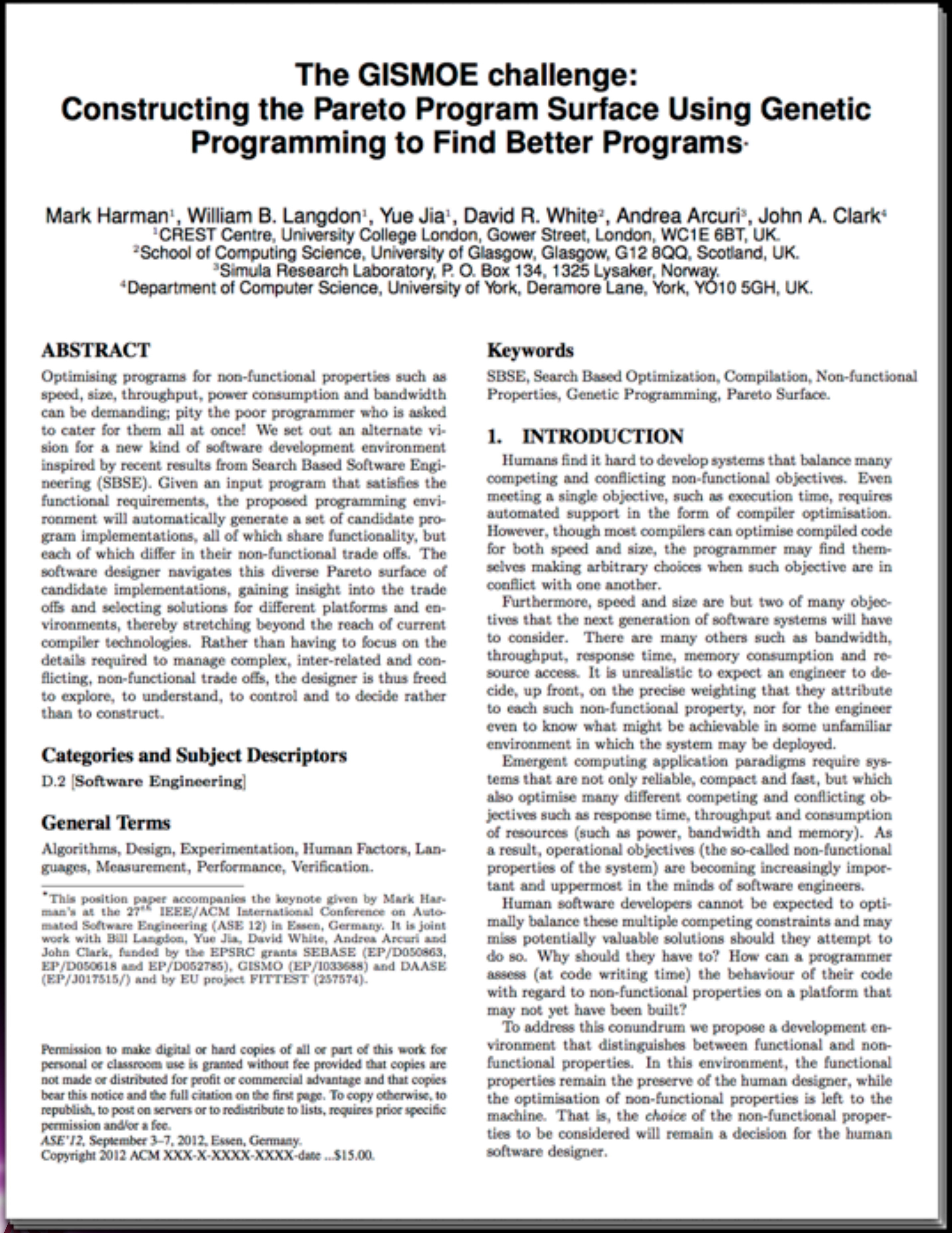
Compile SBSE into deployed Software

Dynamic Adaptive SBSE

Compile SBSE into deployed Software

First achieve “Static Adaptive SBSE!”

ASE 2012 keynote paper



The GISMOE challenge: Constructing the Pareto Program Surface Using Genetic Programming to Find Better Programs*

Mark Harman[1], William B. Langdon[1], Yue Jia[1], David R. White[2], Andrea Arcuri[3], John A. Clark[4]
[1]CREST Centre, University College London, Gower Street, London, WC1E 6BT, UK.
[2]School of Computing Science, University of Glasgow, Glasgow, G12 8QQ, Scotland, UK.
[3]Simula Research Laboratory, P. O. Box 134, 1325 Lysaker, Norway.
[4]Department of Computer Science, University of York, Deramore Lane, York, YO10 5GH, UK.

ABSTRACT

Optimising programs for non-functional properties such as speed, size, throughput, power consumption and bandwidth can be demanding; pity the poor programmer who is asked to cater for them all at once! We set out an alternate vision for a new kind of software development environment inspired by recent results from Search Based Software Engineering (SBSE). Given an input program that satisfies the functional requirements, the proposed programming environment will automatically generate a set of candidate program implementations, all of which share functionality, but each of which differ in their non-functional trade offs. The software designer navigates this diverse Pareto surface of candidate implementations, gaining insight into the trade offs and selecting solutions for different platforms and environments, thereby stretching beyond the reach of current compiler technologies. Rather than having to focus on the details required to manage complex, inter-related and conflicting, non-functional trade offs, the designer is thus freed to explore, to understand, to control and to decide rather than to construct.

Categories and Subject Descriptors

D.2 [**Software Engineering**]

General Terms

Algorithms, Design, Experimentation, Human Factors, Languages, Measurement, Performance, Verification.

ASE'12, September 3–7, 2012, Essen, Germany.

Keywords

SBSE, Search Based Optimization, Compilation, Non-functional Properties, Genetic Programming, Pareto Surface.

1. INTRODUCTION

Humans find it hard to develop systems that balance many competing and conflicting non-functional objectives. Even meeting a single objective, such as execution time, requires automated support in the form of compiler optimisation. However, though most compilers can optimise compiled code for both speed and size, the programmer may find themselves making arbitrary choices when such objective are in conflict with one another.

Furthermore, speed and size are but two of many objectives that the next generation of software systems will have to consider. There are many others such as bandwidth, throughput, response time, memory consumption and resource access. It is unrealistic to expect an engineer to decide, up front, on the precise weighting that they attribute to each such non-functional property, nor for the engineer even to know what might be achievable in some unfamiliar environment in which the system may be deployed.

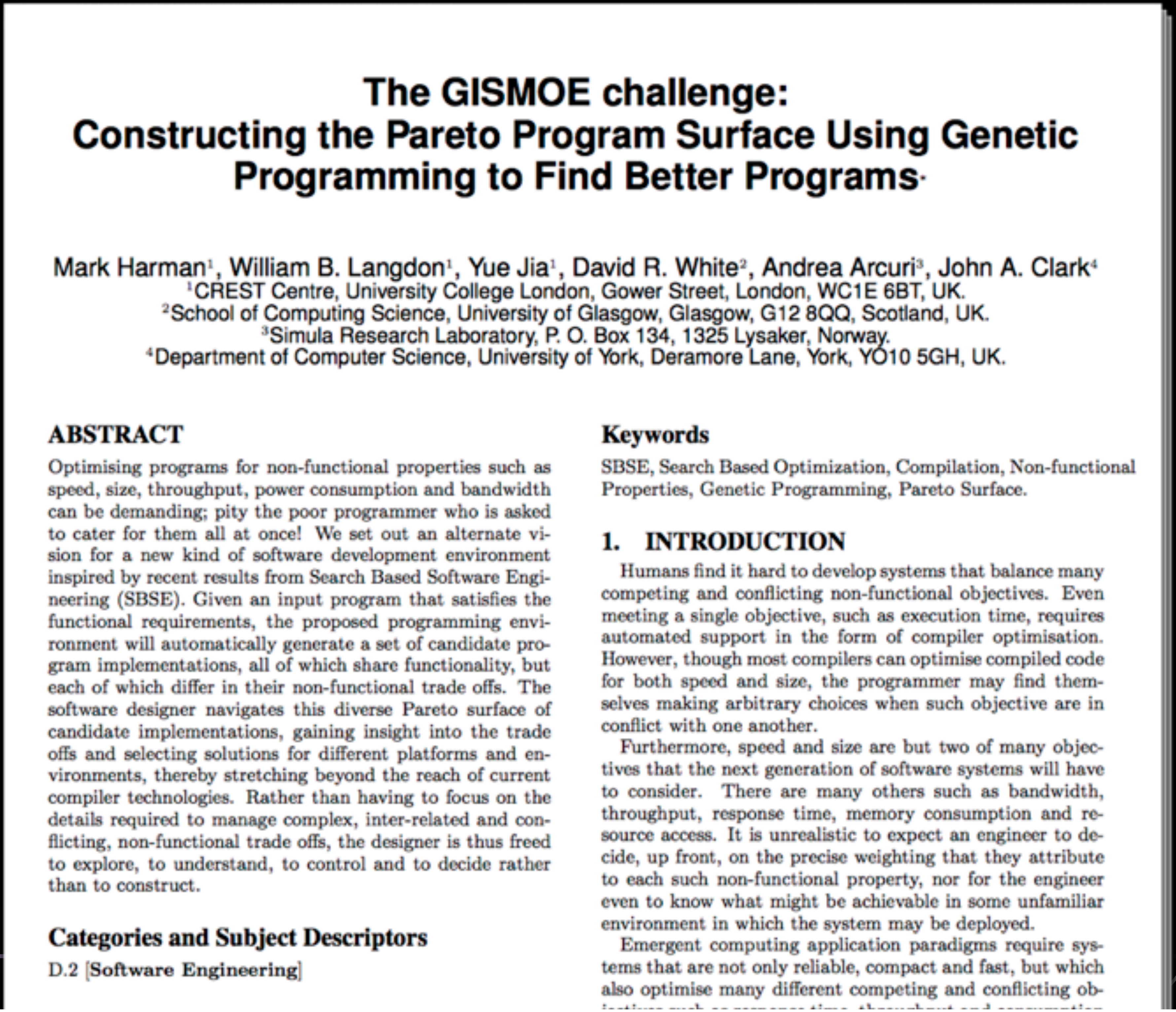
Emergent computing application paradigms require systems that are not only reliable, compact and fast, but which also optimise many different competing and conflicting objectives such as response time, throughput and consumption of resources (such as power, bandwidth and memory). As a result, operational objectives (the so-called non-functional properties of the system) are becoming increasingly important and uppermost in the minds of software engineers.

Human software developers cannot be expected to optimally balance these multiple competing constraints and may miss potentially valuable solutions should they attempt to do so. Why should they have to? How can a programmer assess (at code writing time) the behaviour of their code with regard to non-functional properties on a platform that may not yet have been built?

To address this conundrum we propose a development environment that distinguishes between functional and non-functional properties. In this environment, the functional properties remain the preserve of the human designer, while the optimisation of non-functional properties is left to the machine. That is, the *choice* of the non-functional properties to be considered will remain a decision for the human software designer.

[*] This position paper accompanies the keynote given by Mark Harman's at the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE 12) in Essen, Germany. It is joint work with Bill Langdon, Yue Jia, David White, Andrea Arcuri and John Clark, funded by the EPSRC grants SEBASE (EP/D050863, EP/D050618 and EP/D052785), GISMO (EP/I033688) and DAASE (EP/J017515/) and by EU project FITTEST (257574).

ASE 2012 keynote paper



The GISMOE challenge: Constructing the Pareto Program Surface Using Genetic Programming to Find Better Programs

Mark Harman[1], William B. Langdon[1], Yue Jia[1], David R. White[2], Andrea Arcuri[3], John A. Clark[4]
[1]CREST Centre, University College London, Gower Street, London, WC1E 6BT, UK.
[2]School of Computing Science, University of Glasgow, Glasgow, G12 8QQ, Scotland, UK.
[3]Simula Research Laboratory, P. O. Box 134, 1325 Lysaker, Norway.
[4]Department of Computer Science, University of York, Deramore Lane, York, YO10 5GH, UK.

ABSTRACT

Optimising programs for non-functional properties such as speed, size, throughput, power consumption and bandwidth can be demanding; pity the poor programmer who is asked to cater for them all at once! We set out an alternate vision for a new kind of software development environment inspired by recent results from Search Based Software Engineering (SBSE). Given an input program that satisfies the functional requirements, the proposed programming environment will automatically generate a set of candidate program implementations, all of which share functionality, but each of which differ in their non-functional trade offs. The software designer navigates this diverse Pareto surface of candidate implementations, gaining insight into the trade offs and selecting solutions for different platforms and environments, thereby stretching beyond the reach of current compiler technologies. Rather than having to focus on the details required to manage complex, inter-related and conflicting, non-functional trade offs, the designer is thus freed to explore, to understand, to control and to decide rather than to construct.

Categories and Subject Descriptors

D.2 [Software Engineering]

Keywords

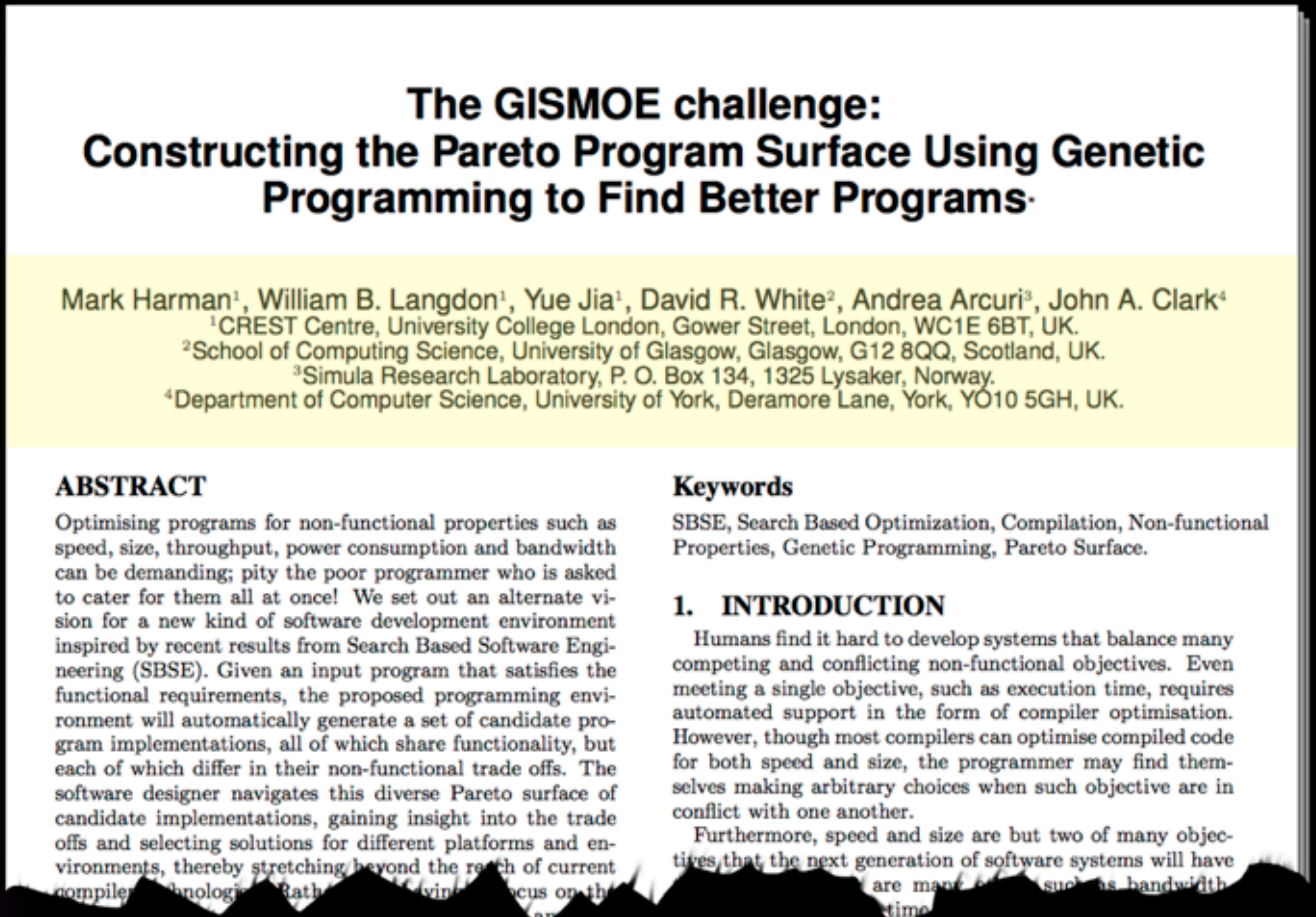
SBSE, Search Based Optimization, Compilation, Non-functional Properties, Genetic Programming, Pareto Surface.

1. INTRODUCTION

Humans find it hard to develop systems that balance many competing and conflicting non-functional objectives. Even meeting a single objective, such as execution time, requires automated support in the form of compiler optimisation. However, though most compilers can optimise compiled code for both speed and size, the programmer may find themselves making arbitrary choices when such objective are in conflict with one another.

Furthermore, speed and size are but two of many objectives that the next generation of software systems will have to consider. There are many others such as bandwidth, throughput, response time, memory consumption and resource access. It is unrealistic to expect an engineer to decide, up front, on the precise weighting that they attribute to each such non-functional property, nor for the engineer even to know what might be achievable in some unfamiliar environment in which the system may be deployed.

Emergent computing application paradigms require systems that are not only reliable, compact and fast, but which also optimise many different competing and conflicting ob-



The GISMOE challenge: Constructing the Pareto Program Surface Using Genetic Programming to Find Better Programs

Mark Harman[1], William B. Langdon[1], Yue Jia[1], David R. White[2], Andrea Arcuri[3], John A. Clark[4]
[1]CREST Centre, University College London, Gower Street, London, WC1E 6BT, UK.
[2]School of Computing Science, University of Glasgow, Glasgow, G12 8QQ, Scotland, UK.
[3]Simula Research Laboratory, P. O. Box 134, 1325 Lysaker, Norway.
[4]Department of Computer Science, University of York, Deramore Lane, York, YO10 5GH, UK.

ABSTRACT

Optimising programs for non-functional properties such as speed, size, throughput, power consumption and bandwidth can be demanding; pity the poor programmer who is asked to cater for them all at once! We set out an alternate vision for a new kind of software development environment inspired by recent results from Search Based Software Engineering (SBSE). Given an input program that satisfies the functional requirements, the proposed programming environment will automatically generate a set of candidate program implementations, all of which share functionality, but each of which differ in their non-functional trade offs. The software designer navigates this diverse Pareto surface of candidate implementations, gaining insight into the trade offs and selecting solutions for different platforms and environments, thereby stretching beyond the reach of current

Keywords

SBSE, Search Based Optimization, Compilation, Non-functional Properties, Genetic Programming, Pareto Surface.

1. INTRODUCTION

Humans find it hard to develop systems that balance many competing and conflicting non-functional objectives. Even meeting a single objective, such as execution time, requires automated support in the form of compiler optimisation. However, though most compilers can optimise compiled code for both speed and size, the programmer may find themselves making arbitrary choices when such objective are in conflict with one another.

Furthermore, speed and size are but two of many objectives that the next generation of software systems will have

Dynamic Adaptive SBSE

Compile SBSE into deployed Software

Dynamic Adaptive SBSE

Compile SBSE into deployed Software

... what’s the difference between ASE and ESEM keynote?

ASE 2012 Keynote

Static Adaptive SBSE

ESEM 2012 Keynote

Dynamic Adaptive SBSE

DAASE

Dynamic Adaptive SBSE

Compile SBSE into deployed Software

Dynamic Adaptive SBSE

Compile SBSE into deployed Software

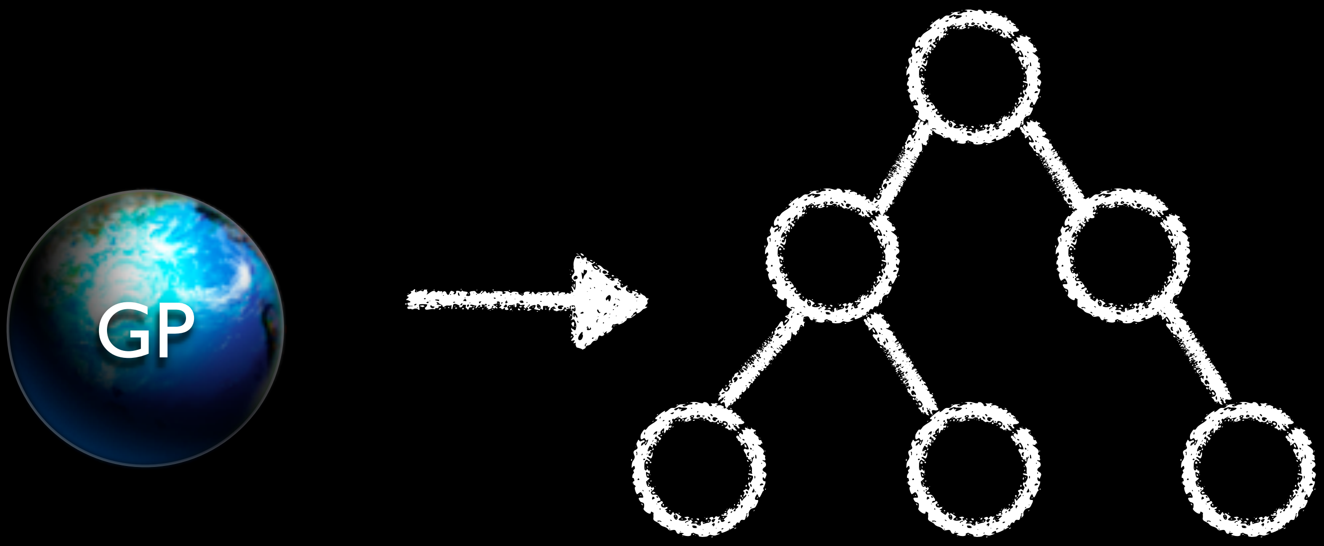
... where’s the evidence that this is feasible?

Exciting evidence ...

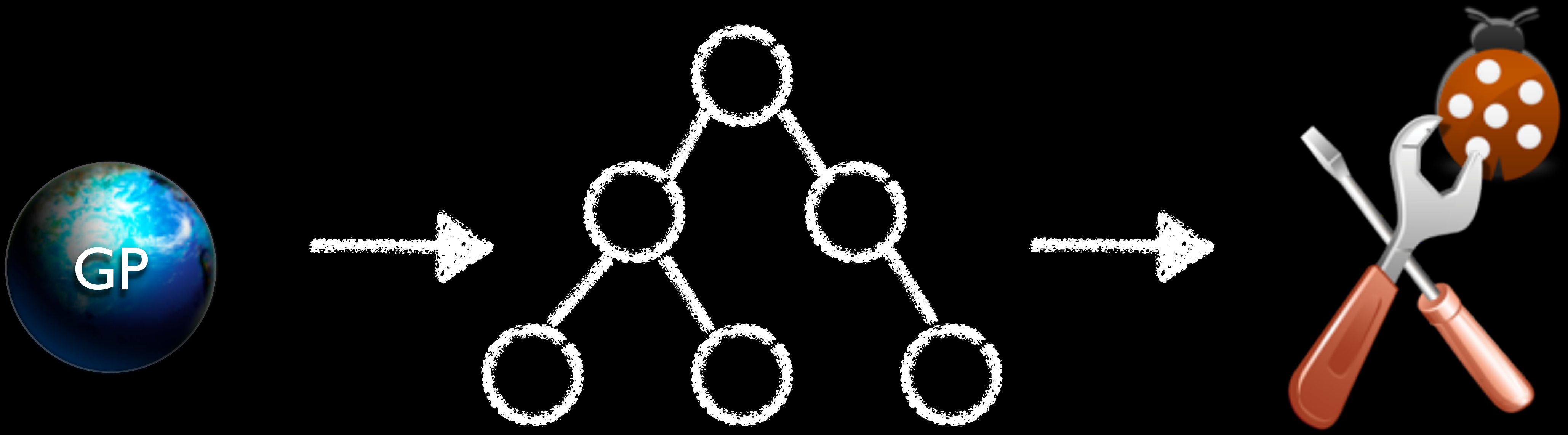
Bug Fixing

Bug Fixing

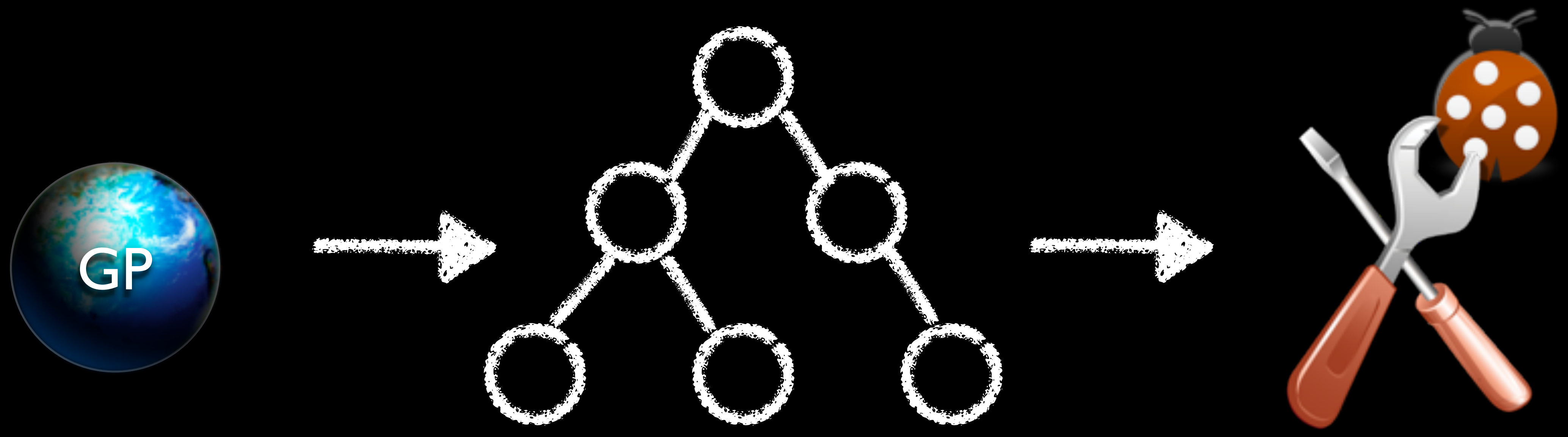
Bug Fixing



Bug Fixing

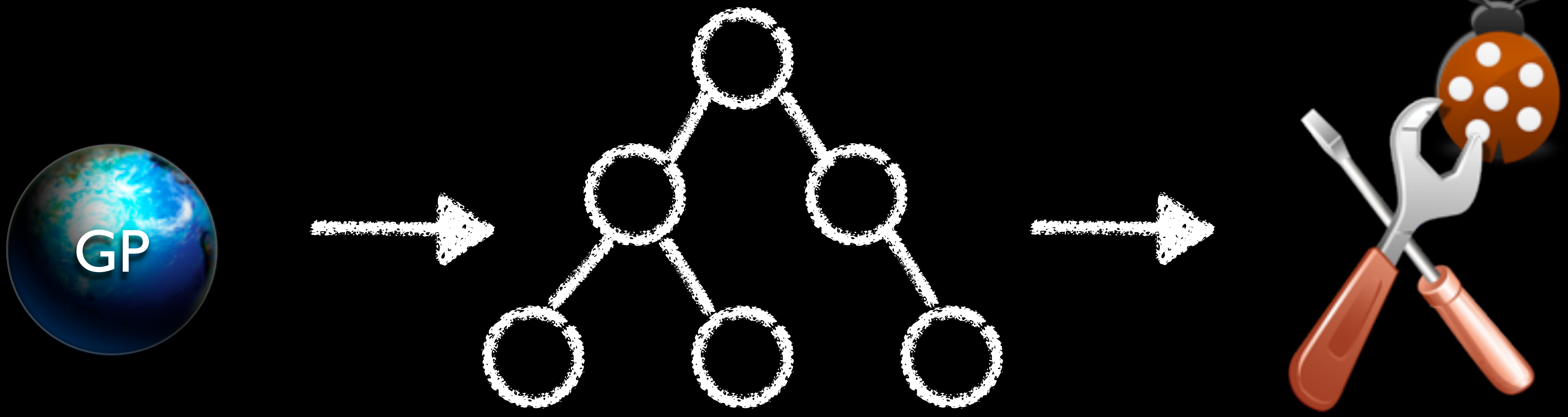


Bug Fixing

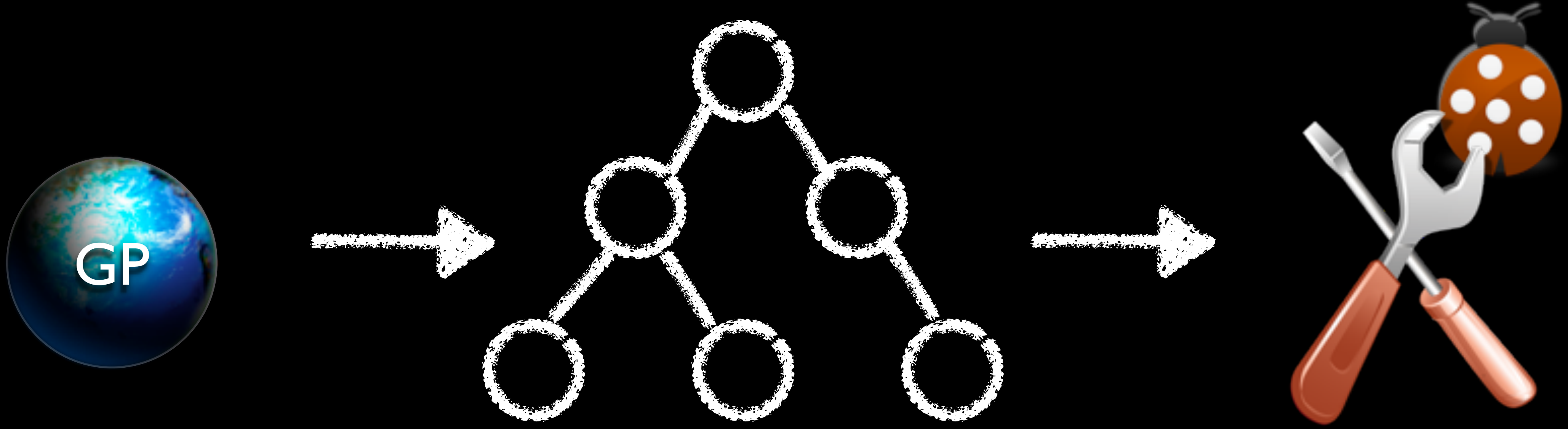


A. Arcuri and X. Yao. A Novel
Co-evolutionary Approach to Automatic Software Bug Fixing. (CEC '08)

Bug Fixing



A. Arcuri and X. Yao. A Novel
Co-evolutionary Approach to Automatic Software Bug Fixing. (CEC '08)

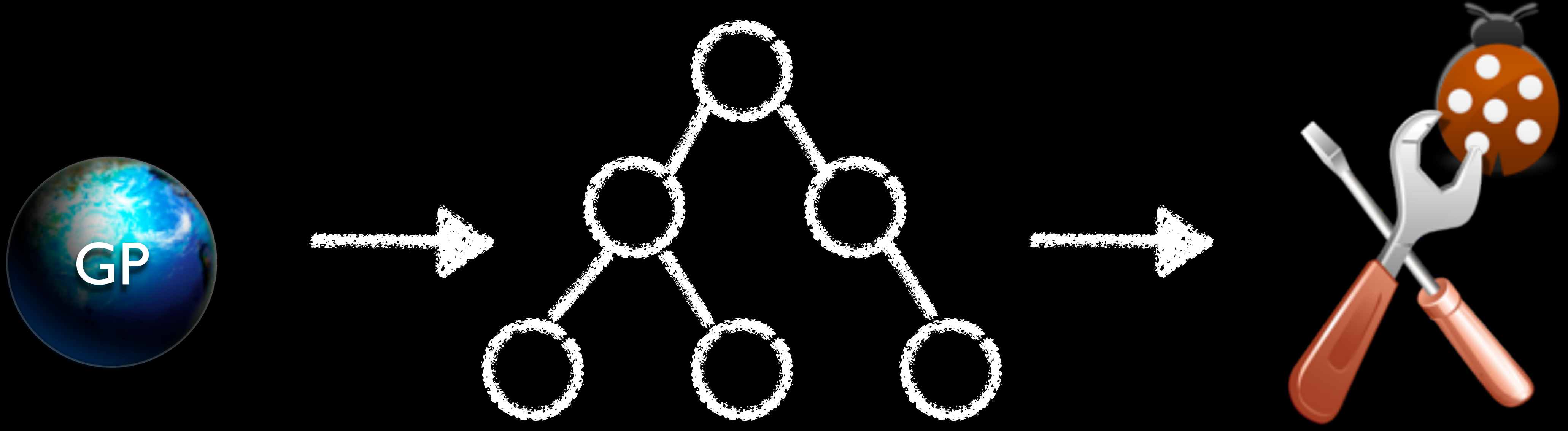


A. Arcuri and X. Yao. A Novel Co-evolutionary Approach to Automatic Software Bug Fixing. (CEC ’08)

C. L. Goues, M. Dewey-Vogt, S. Forrest, and W. Weimer. A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each. (ICSE‘12)

C. Le Goues, T. Nguyen, S. Forrest, and W. Weimer. GenProg: A generic method for automatic software repair. （TSE’12）

W. Weimer, T. V. Nguyen, C. L. Goues, and S. Forrest. Automatically finding patches using genetic programming. In International Conference on Software Engineering (ICSE‘09)



A. Arcuri and X. Yao. A Novel
Co-evolutionary Approach to Automatic Software Bug Fixing. (CEC '08)

C. L. Goues, M. Dewey-Vogt, S. Forrest, and W. Weimer. A systematic study of
automated program repair: Fixing 55 out of 105 bugs for $8 each. (ICSE'12)

C. Le Goues, T. Nguyen, S. Forrest, an
for automatic software repair. (TSE'

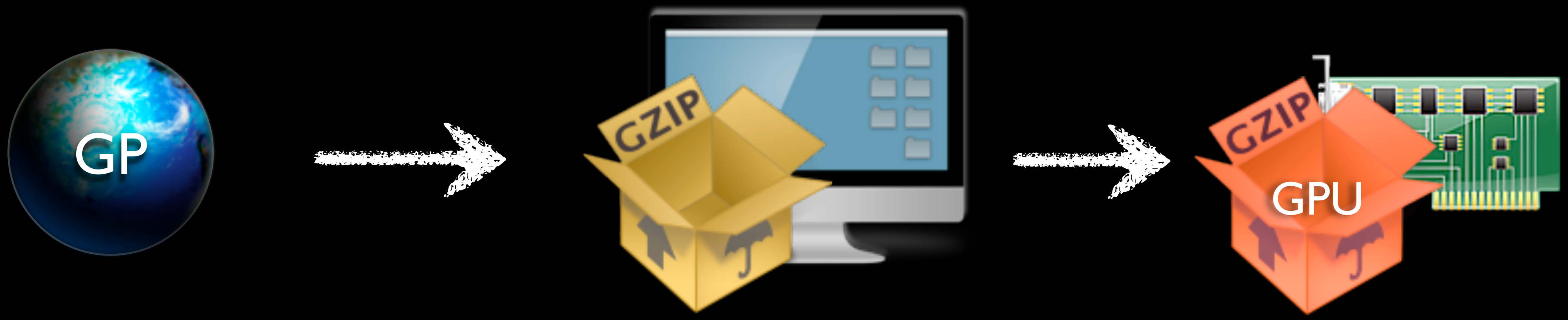
W. Weimer, T.V. Nguyen, C. L. Goues, a
patches using genetic programming. In
Engineering (ICSE'09)

"The original program serves as an ideal oracle for the re-evolution of fragments of new code."

Migration

Migration

Migration



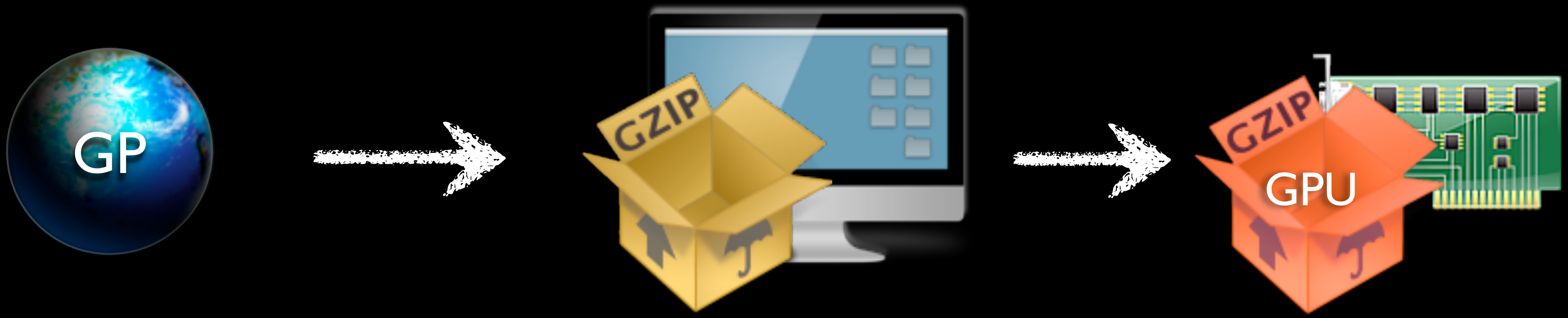
Migration

W. B. Langdon and M. Harman
Evolving a CUDA kernel from an nVidia template (CEC'10)

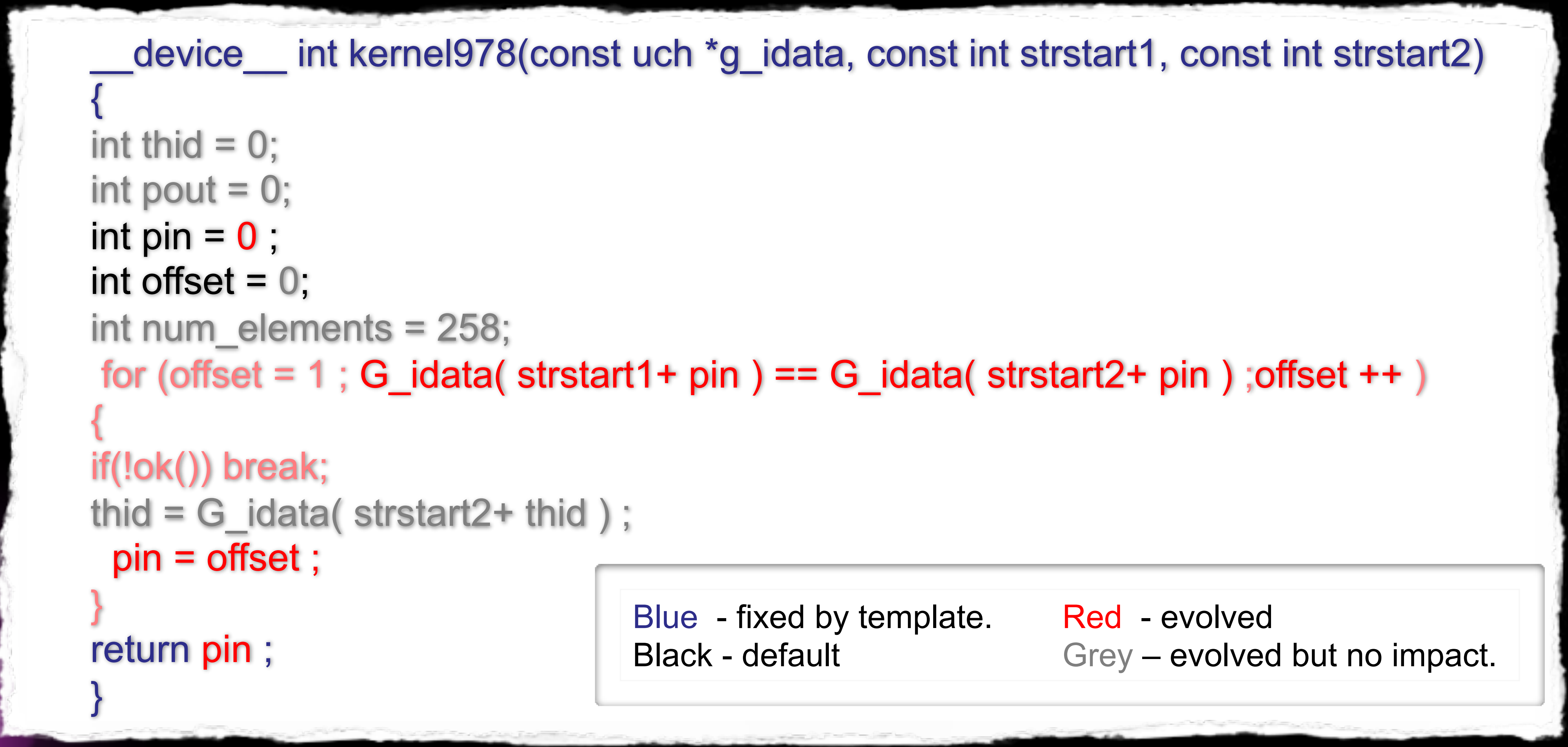
Migration

W. B. Langdon and M. Harman
Evolving a CUDA kernel from an nVidia template (CEC'10)

W. B. Langdon and M. Harman
Evolving a CUDA kernel from an nVidia template (CEC'10)



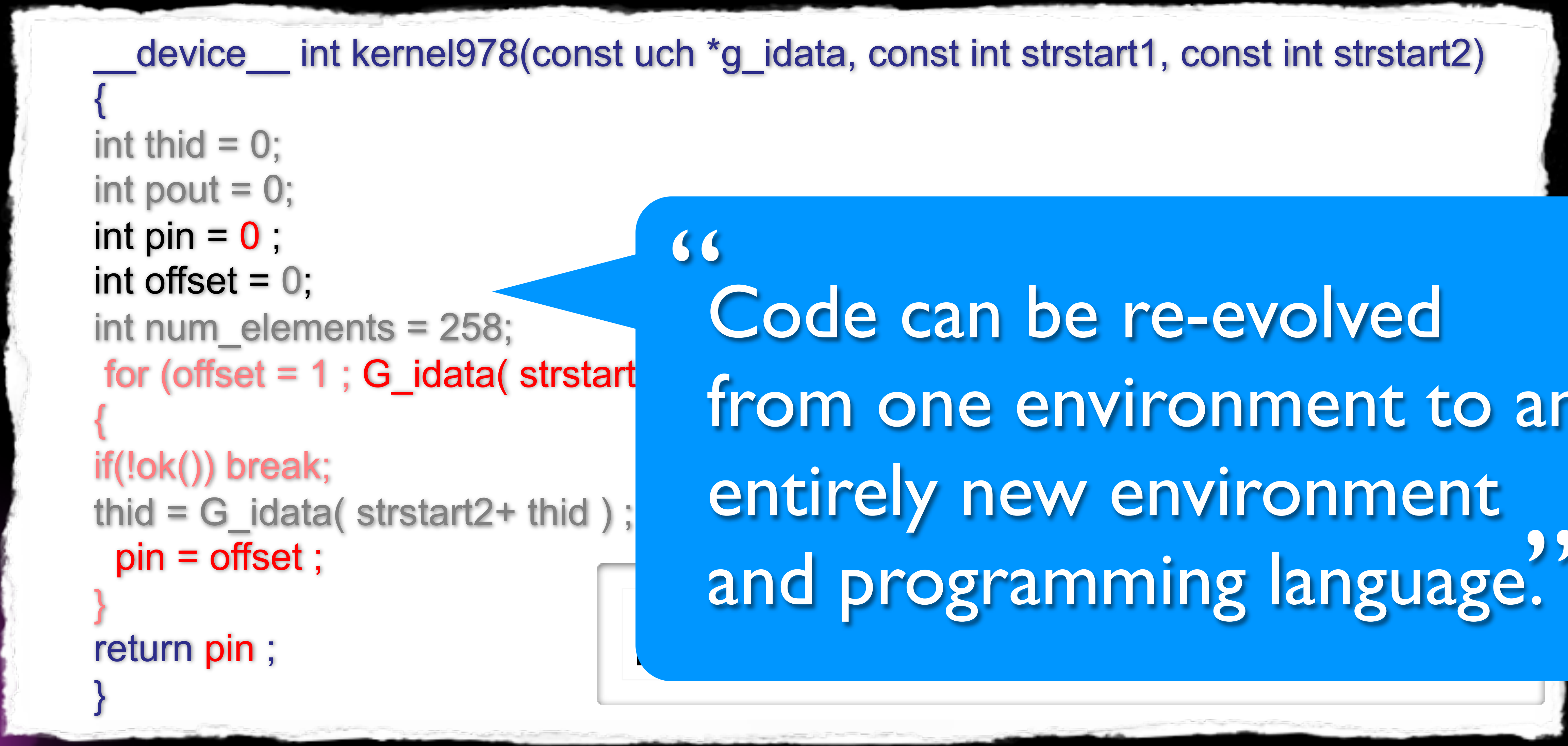
W. B. Langdon and M. Harman
Evolving a CUDA kernel from an nVidia template (CEC'10)



```
__device__ int kernel978(const uch *g_idata, const int strstart1, const int strstart2)
{
int thid = 0;
int pout = 0;
int pin = 0 ;
int offset = 0;
int num_elements = 258;
 for (offset = 1 ; G_idata( strstart1+ pin ) == G_idata( strstart2+ pin ) ;offset ++ )
{
if(!ok()) break;
thid = G_idata( strstart2+ thid ) ;
  pin = offset ;
}
return pin ;
}
```

Blue - fixed by template.   Red - evolved
Black - default   Grey – evolved but no impact.

W. B. Langdon and M. Harman
Evolving a CUDA kernel from an nVidia template (CEC'10)

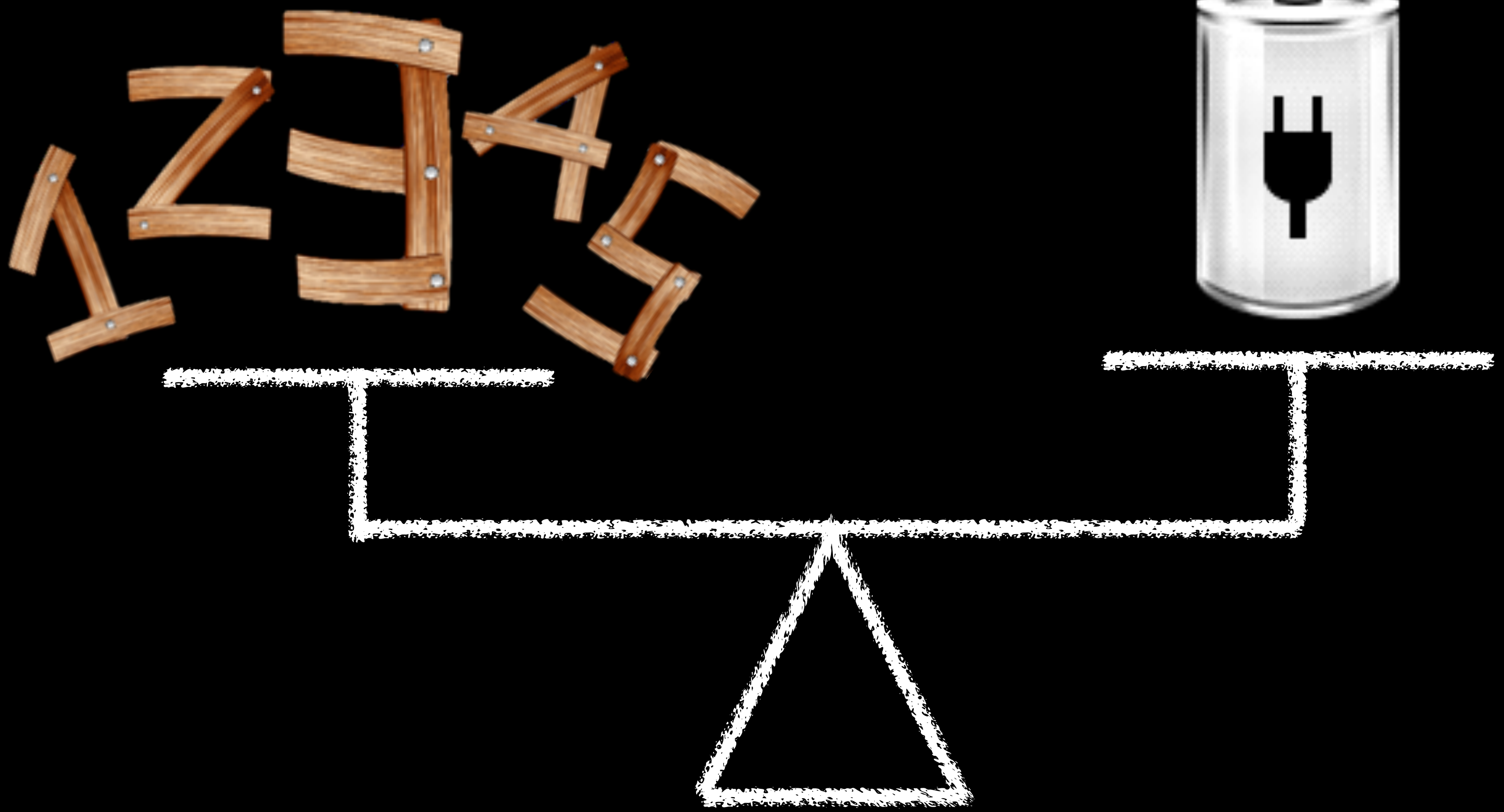


```
__device__ int kernel978(const uch *g_idata, const int strstart1, const int strstart2)
{
int thid = 0;
int pout = 0;
int pin = 0 ;
int offset = 0;
int num_elements = 258;
 for (offset = 1 ; G_idata( strstart
{
if(!ok()) break;
thid = G_idata( strstart2+ thid ) ;
  pin = offset ;
}
return pin ;
}
```

> “Code can be re-evolved from one environment to an entirely new environment and programming language.”

Trading Functional & Non-Functional Requirements

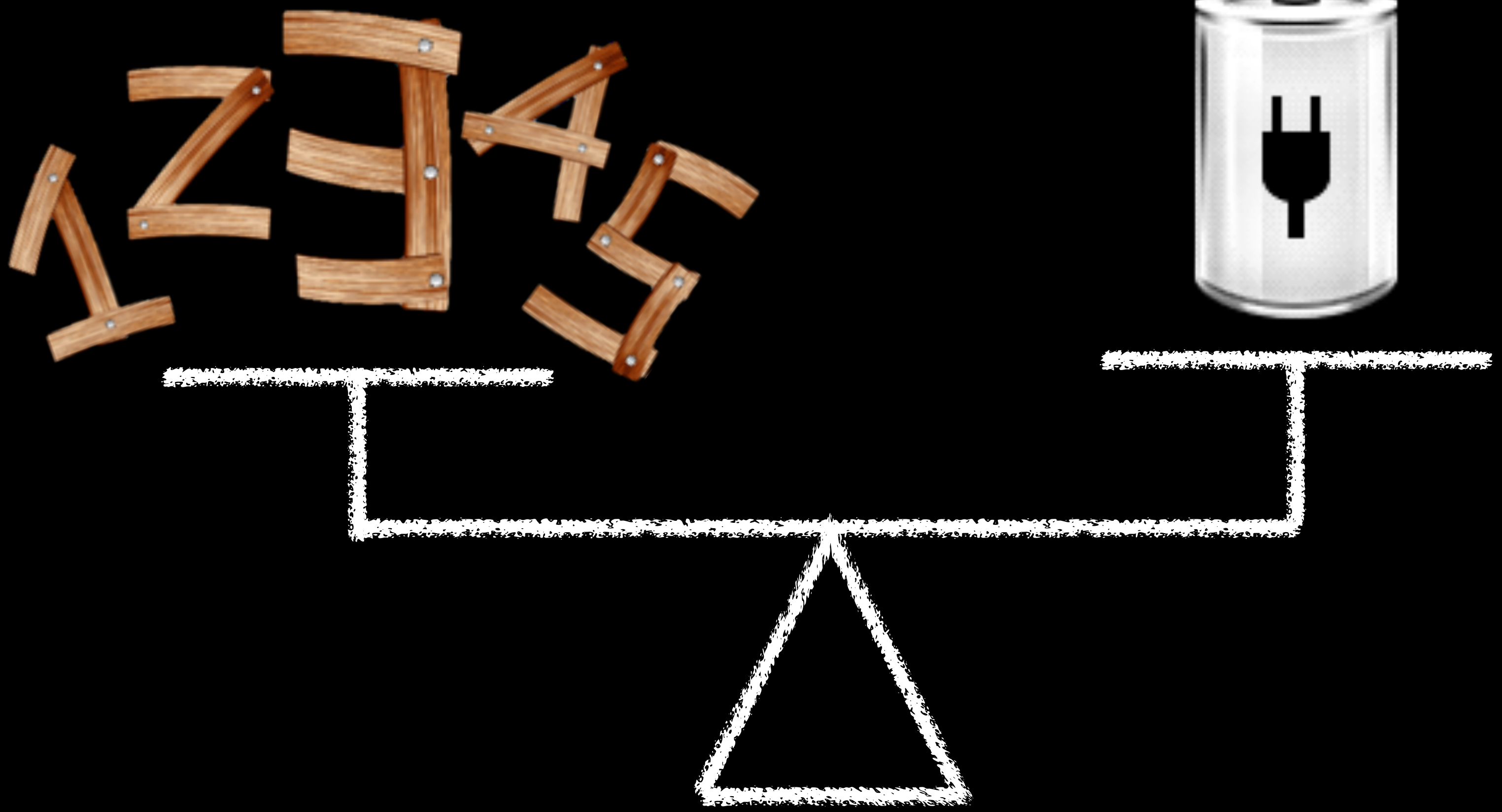
Trading Functional & Non-Functional Requirements



DAASE

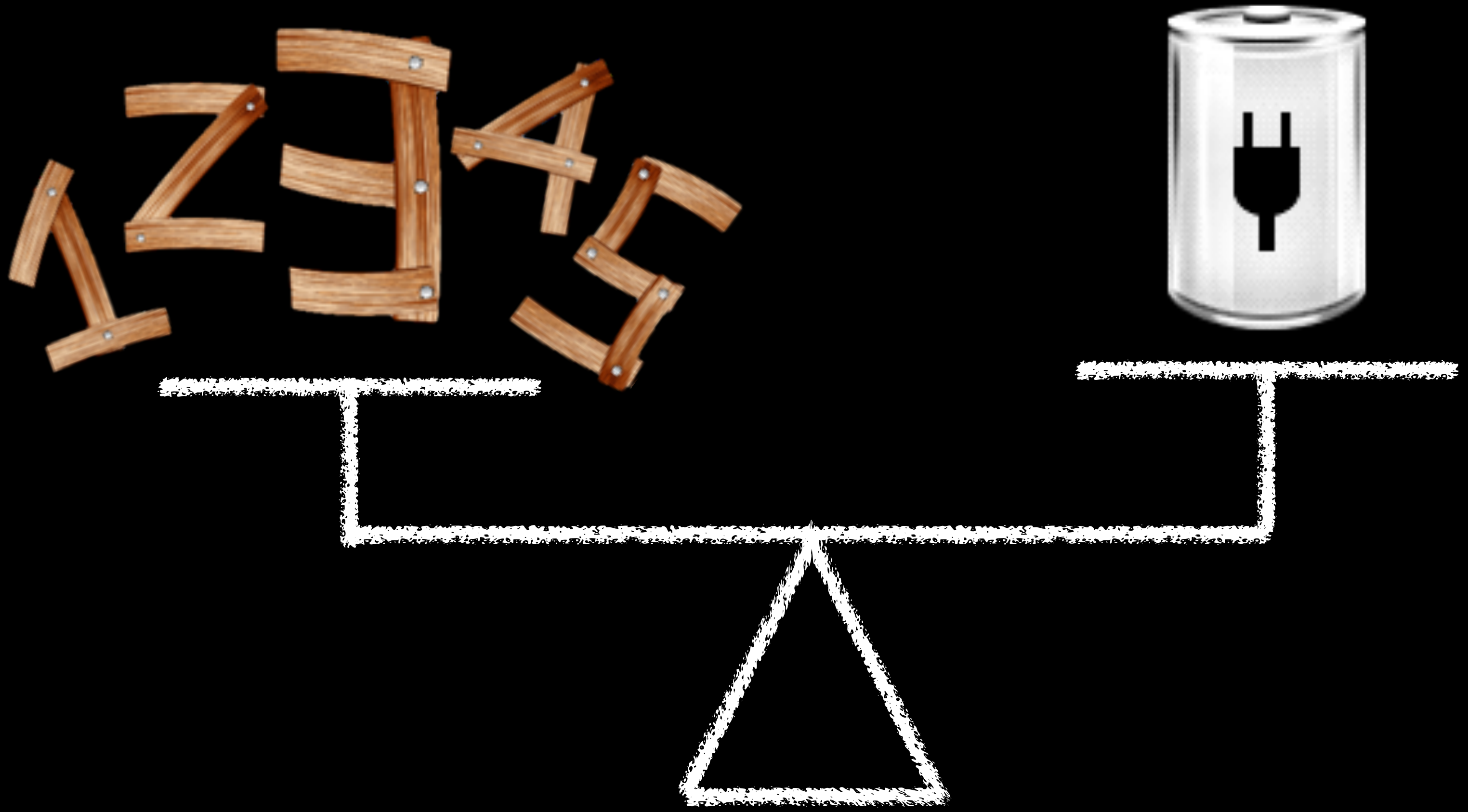
Mark Harman, CREST

Trading Functional & Non-Functional Requirements

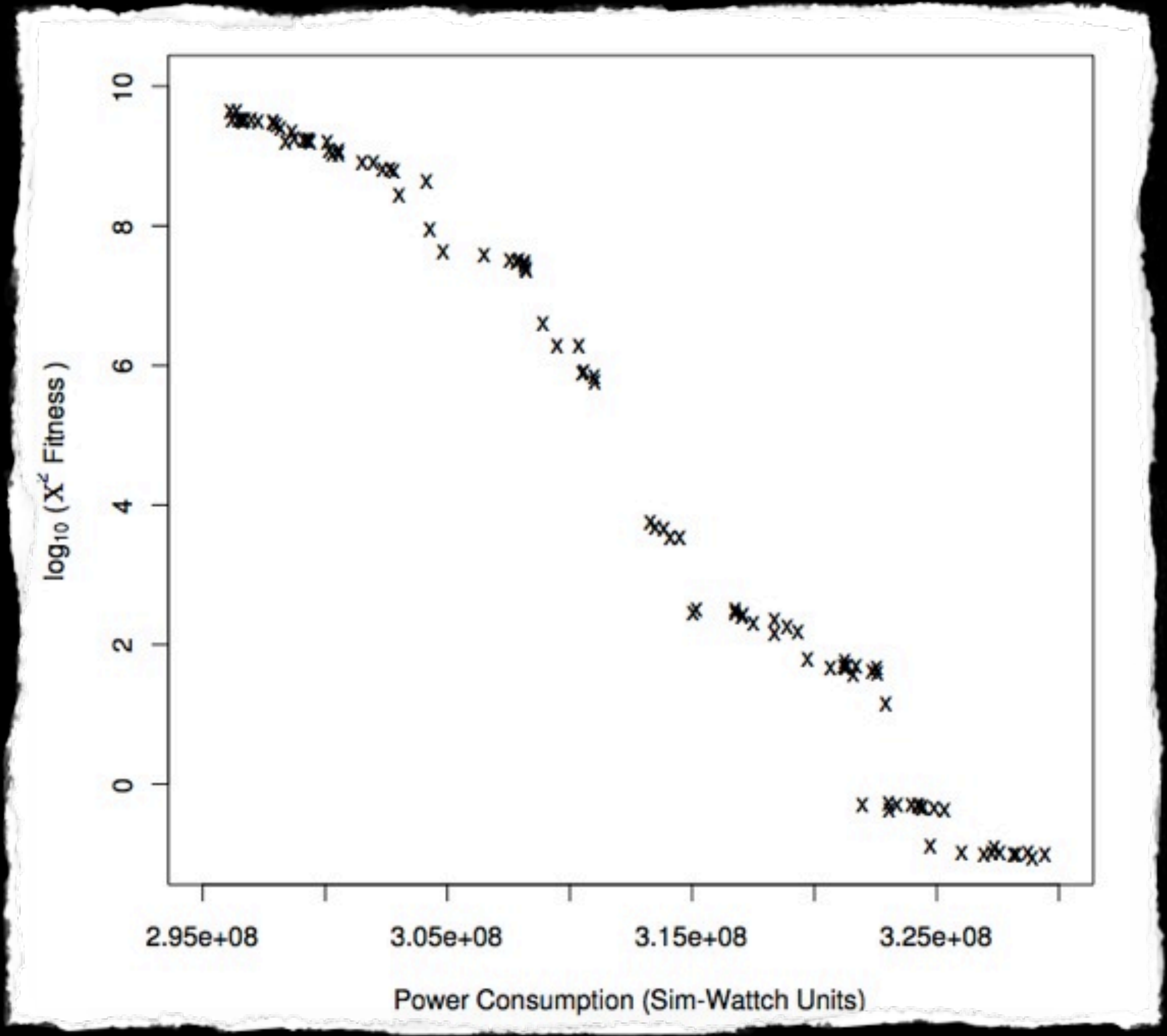


D. R. White, J. Clark, J. Jacob, and S. Poulding.
Searching for resource-efficient programs: Low-power pseudorandom number generators (SEAL 2008)

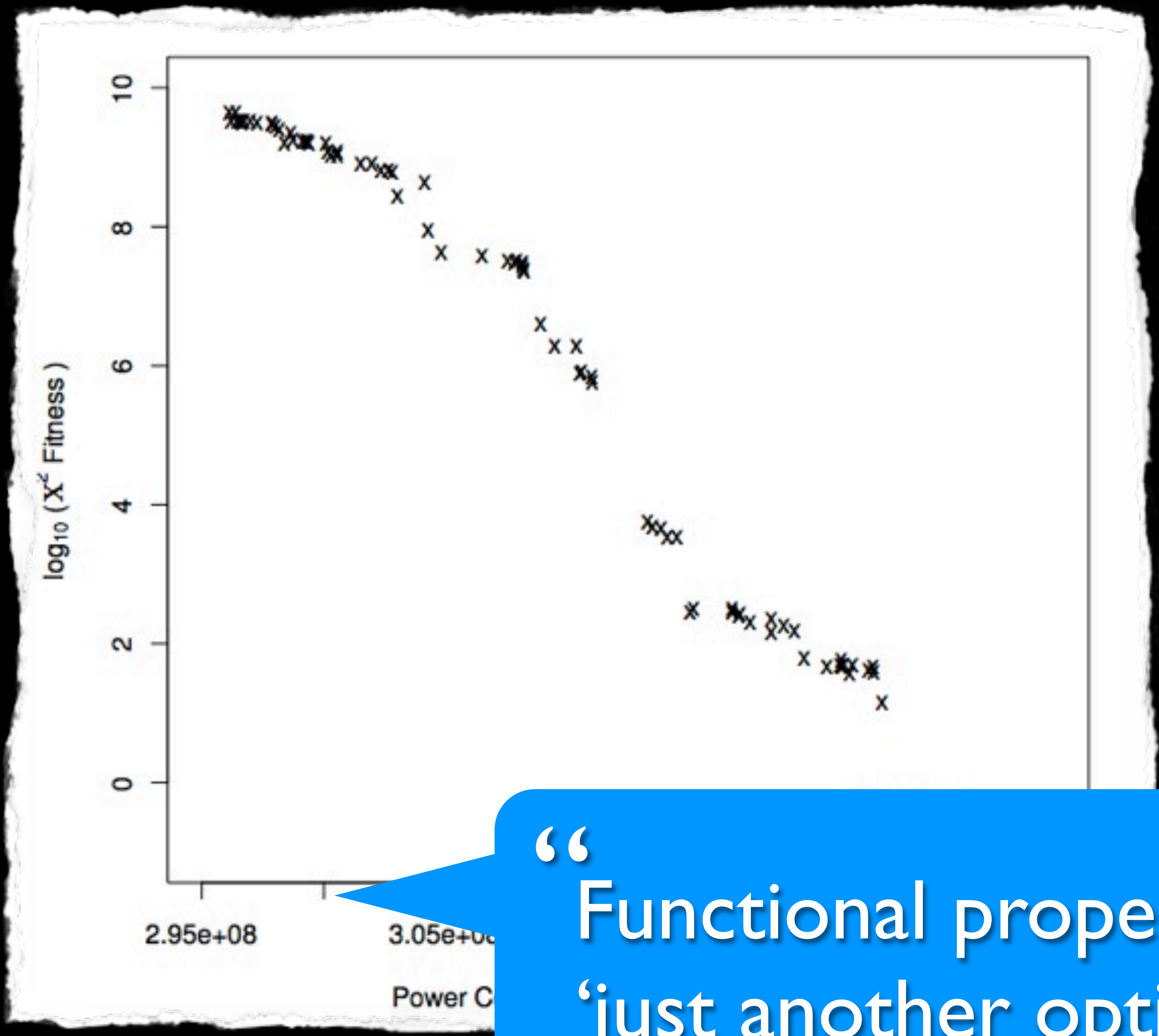
Trading Functional & Non-Functional Requirements



D. R. White, J. Clark, J. Jacob, and S. Poulding.
Searching for resource-efficient programs: Low-power pseudorandom number generators (SEAL 2008)



D. R. White, J. Clark, J. Jacob, and S. Poulding.
Searching for resource-efficient programs: Low-power pseudorandom number generators (SEAL 2008)



“Functional properties are ‘just another optimisation objective’, like non-functional properties. ”

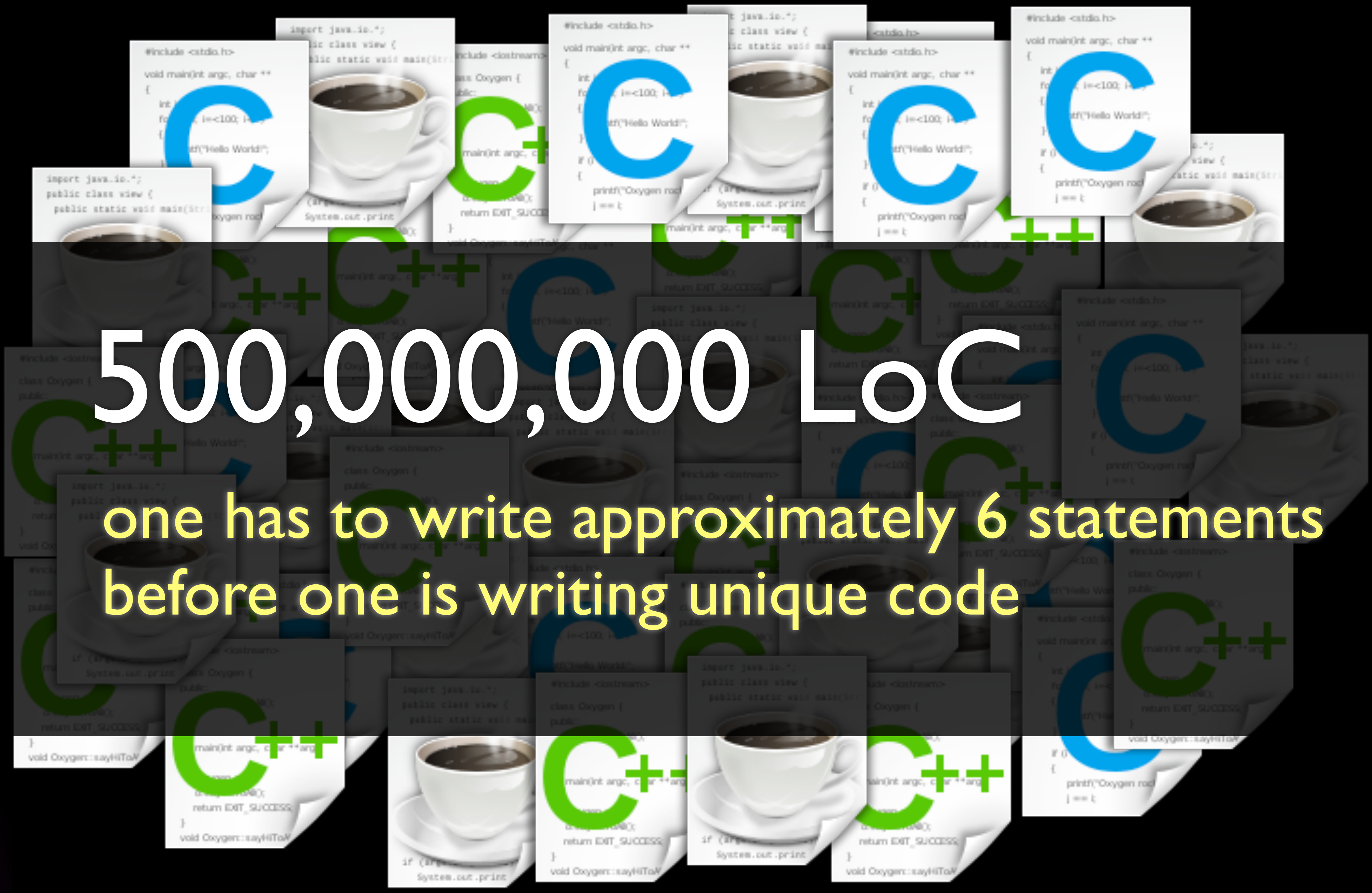
D. R. White, J. Clark, J. Jacob, and S.
Searching for resource-efficient pro r
generators (SEAL 2008)

Software Uniqueness

DAASE

Software Uniqueness

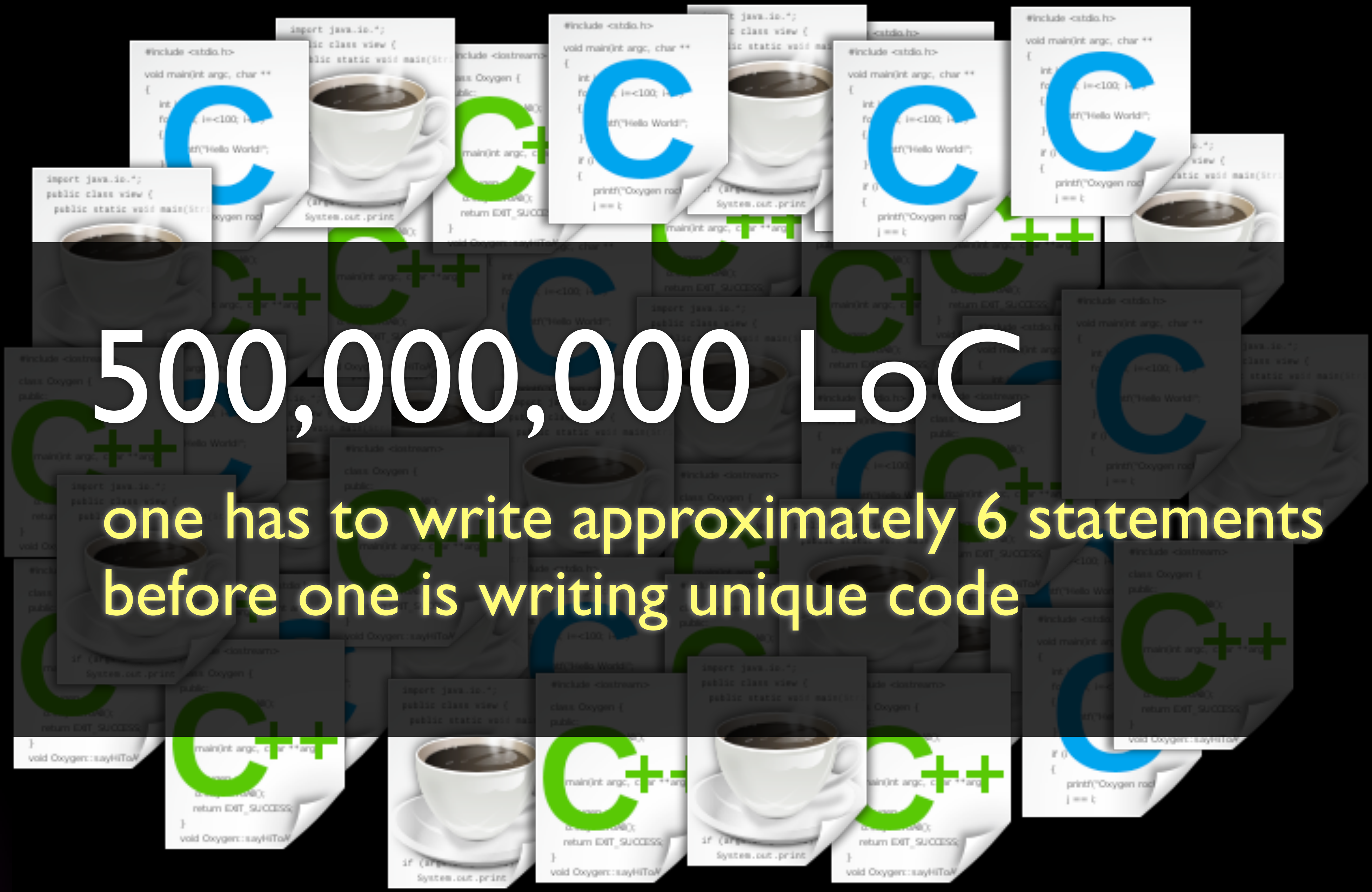
Software Uniqueness



500,000,000 LoC

one has to write approximately 6 statements before one is writing unique code

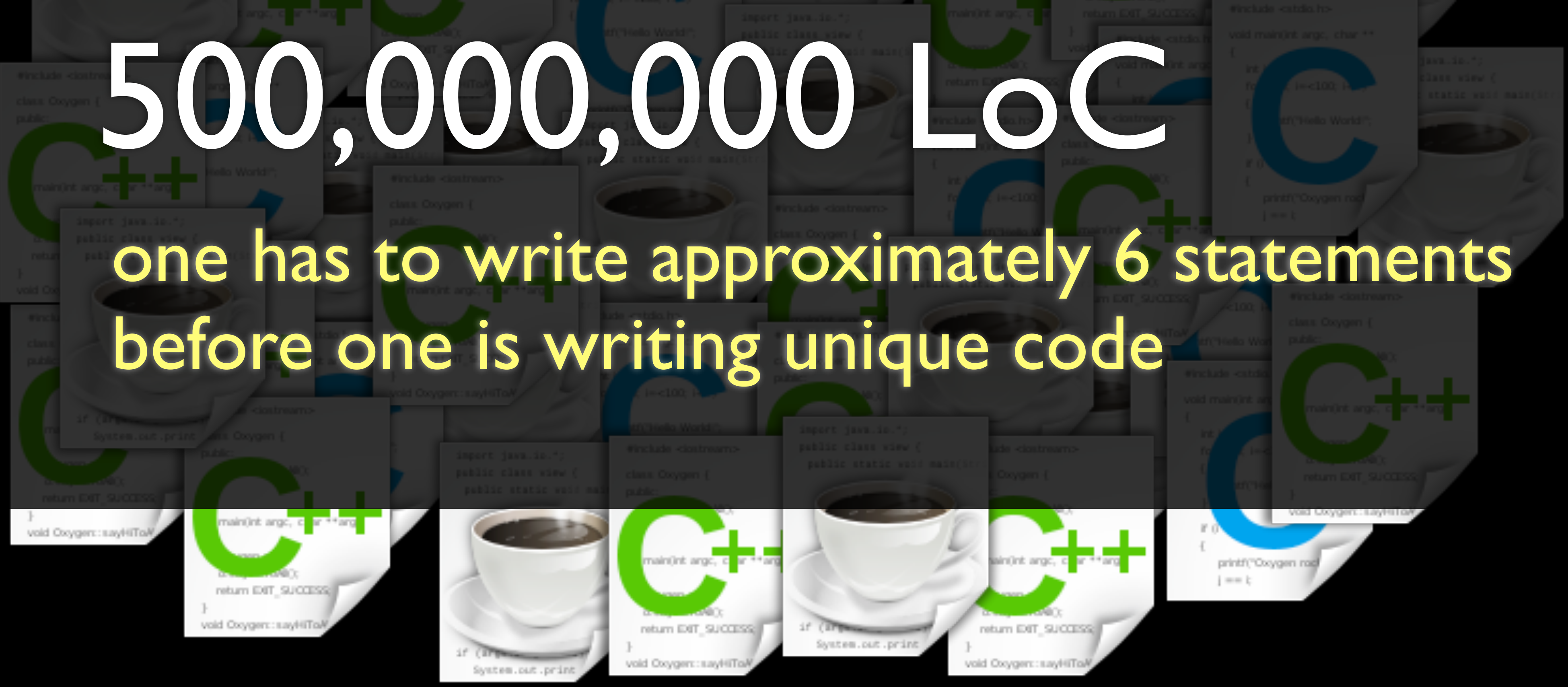
Software Uniqueness



500,000,000 LoC

one has to write approximately 6 statements before one is writing unique code

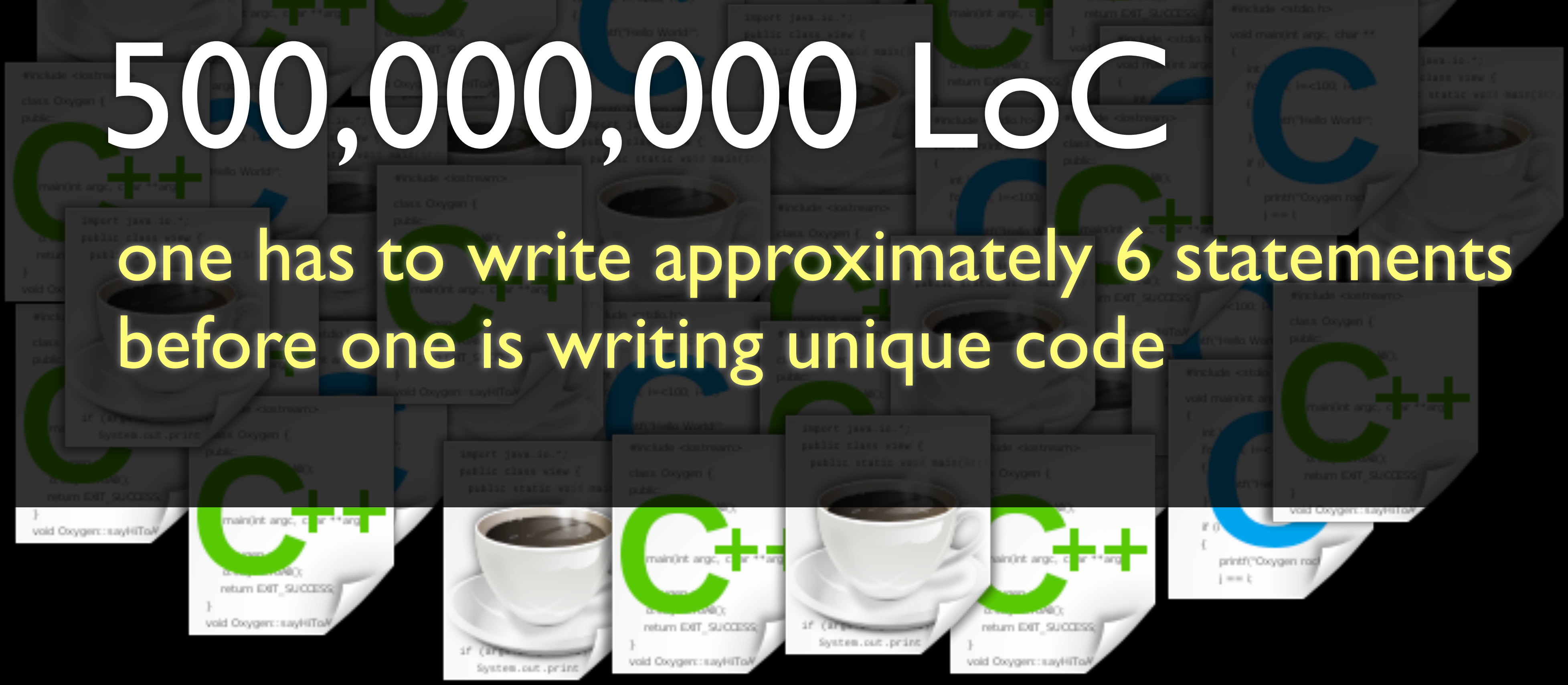
500,000,000 LoC



one has to write approximately 6 statements before one is writing unique code

M. Gabel and Z. Su.
A study of the uniqueness of source code. (FSE 2010)

500,000,000 LoC

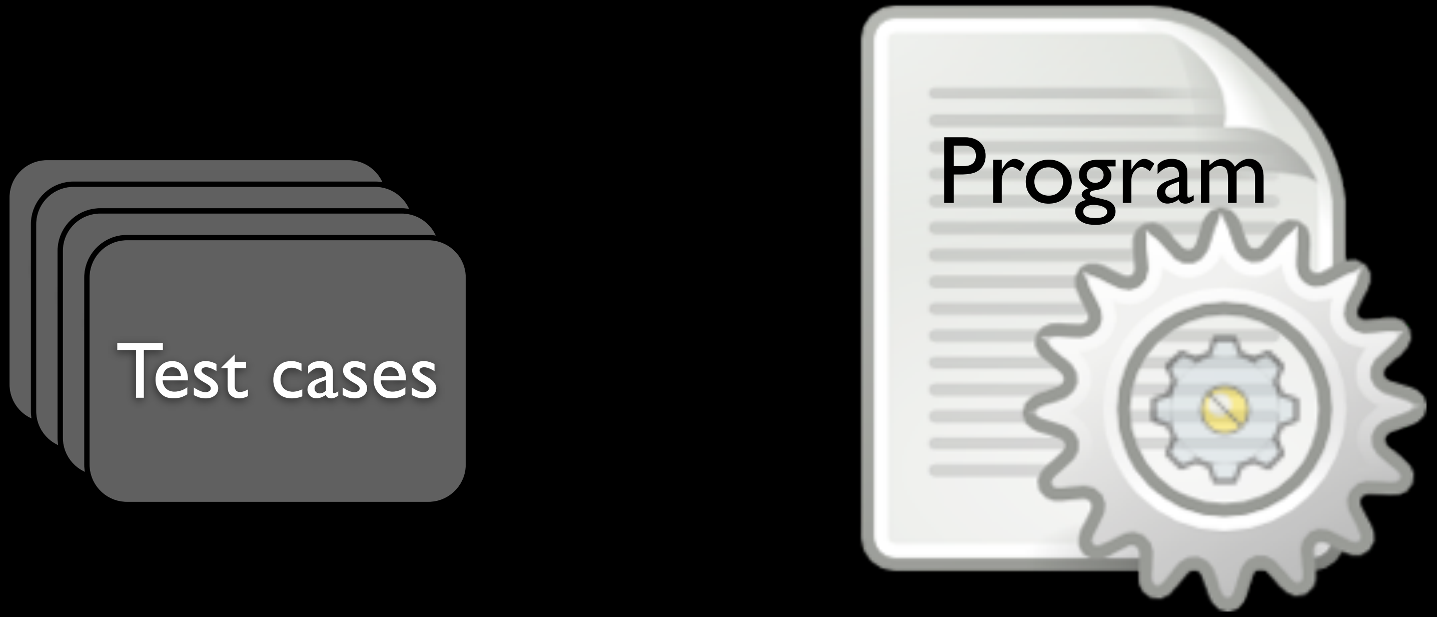


one has to write approximately 6 statements before one is writing unique code

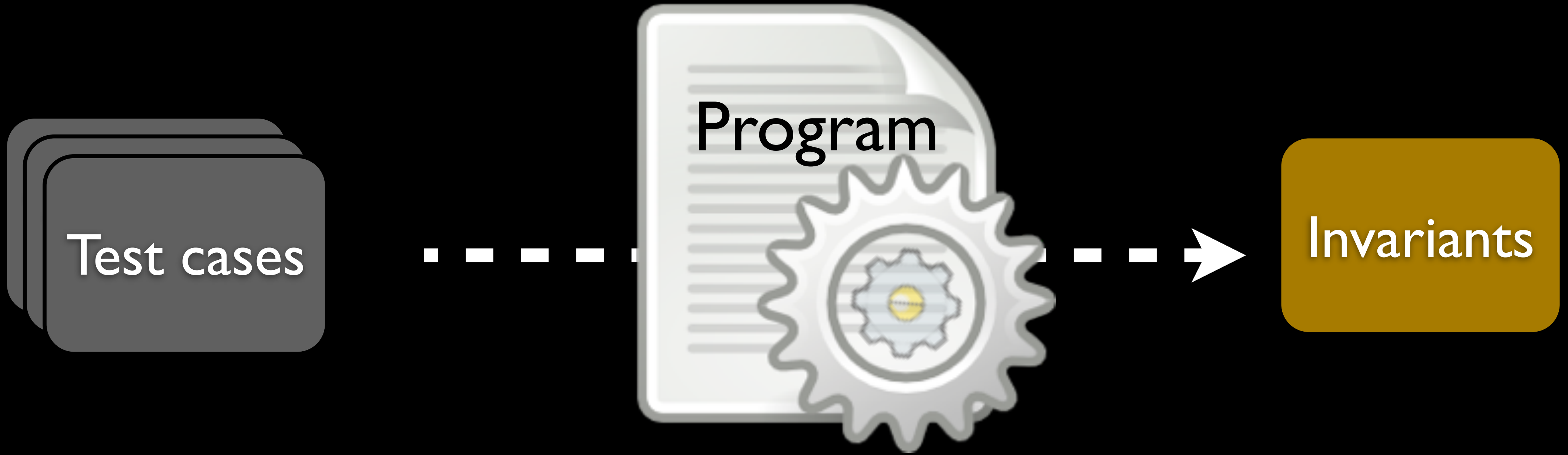
M. Gabel and Z. Su.
A study of the uniquenes

> “The space of candidate programs is far smaller than we might suppose. ”

Dynamically Discovering Static Truths



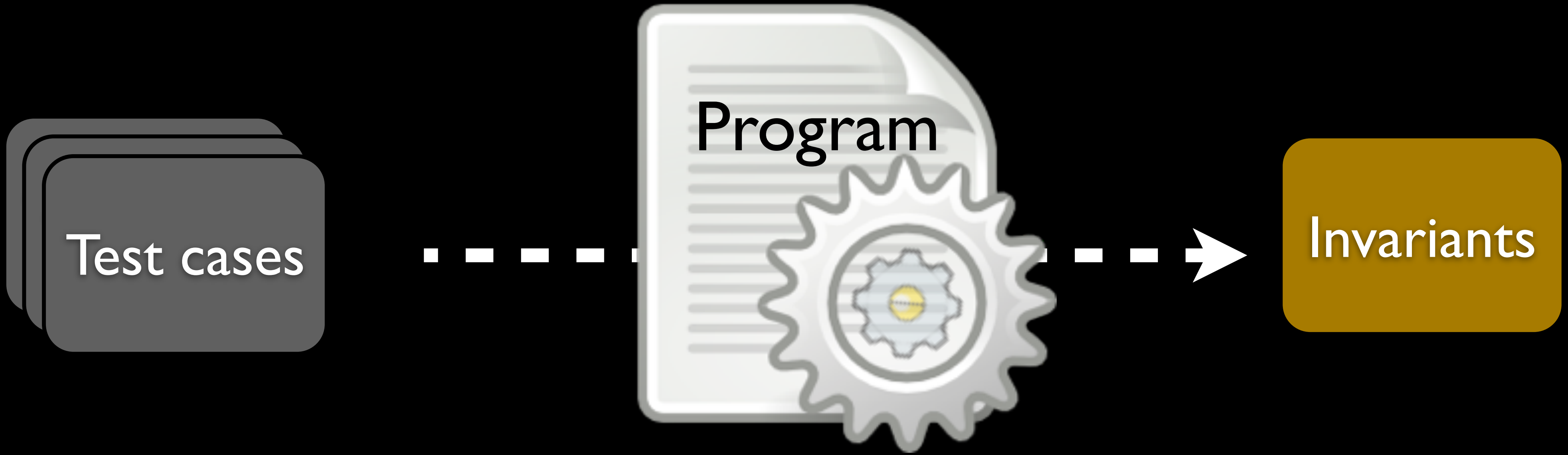
Dynamically Discovering Static Truths



DAASE

Mark Harman, CREST

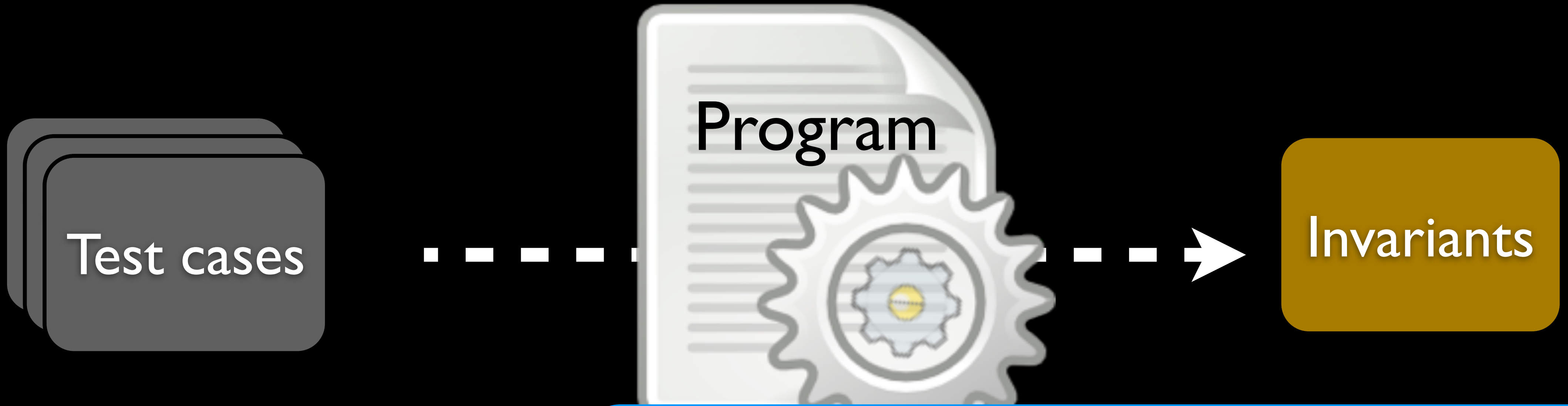
Dynamically Discovering Static Truths



M. D. Ernst. Dynamically Discovering Likely Program Invariants.
PhD Thesis, University of Washington, 2000.

M. D. Ernst, J. Cockrell, W. G. Griswold, and D. Notkin. Dynamically discovering likely program invariants to support program evolution. IEEE Transactions on Software Engineering, 27(2):1–25, Feb. 2001.

Dynamically Discovering Static Truths



“A small amount of dynamic information is sufficient to approximate (and sometimes precisely capture) static information.”

M. D. Ernst. Dynamically Disc
PhD Thesis, University of Wa

M. D. Ernst, J. Cockrell, W. G.
program invariants to suppor
Engineering, 27(2):1–25, Feb.

Latest CREST results

Latest CREST results

Bowtie2: real program of 50,000 LoC

39 files, 20,000 LoC in main code

data structures, modules, file access ...

Latest CREST results

Bowtie2: real program of 50,000 LoC

39 files, 20,000 LoC in main code

data structures, modules, file access ...

Evolved E_Bowtie2

70 times faster on average

and a modest functional improvement

Pictures used with thanks from these sources

chemical plant from http://commons.wikimedia.org/wiki/File:Chemical_Plant_Western_Reclamation.jpg

test tubes from http://commons.wikimedia.org/wiki/File:50ml_Falcon_tubes-02.jpg

ZXSpectrum16k/48k By Bill Bertram (Own work) [CC-BY-SA-2.5 (http://creativecommons.org/licenses/by-sa/2.5)], via Wikimedia Commons

Pickering's Harem: [Public domain], via Wikimedia Commons

BBC_Micro: [Public domain], via Wikimedia Commons

IBM PC: By Boffy B (Own work) [GFDL (http://www.gnu.org/copyleft/fdl.html) or CC-BY-SA-3.0-2.5-2.0-1.0 (http://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons

IMac: By Matthieu Riegler, Wikimedia Commons [CC-BY-3.0 (http://creativecommons.org/licenses/by/3.0)], via Wikimedia Commons

Programmer: undesarchiv, B 145 Bild-F031434-0006 / Gathmann, Jens / CC-BY-SA [CC-BY-SA-3.0-de (http://creativecommons.org/licenses/by-sa/3.0/de/deed.en)], via Wikimedia Commons

Clouds: By jackietran [Creative Commons Attribution-Noncommercial 3.0 Unported License]

Agile: By Devon Fyson [CC-BY-SA-3.0 (http://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons

Human and Monkey: Ekman P, Friesen WV, Hager JC. Facial Action Coding System. Salt Lake City: Research Nexus; 2002.omologous f movements in a human (Ekman et al., 2002)

jet engine from http://commons.wikimedia.org/wiki/File:Jet_engine_numbered.svg under GPL and from http://commons.wikimedia.org/wiki/File:Jet_Engine_SR-71.jpg under wikimedia commons