

Systems Reengineering Patterns

Perdita Stevens

Division of Informatics
University of Edinburgh

Rob Pooley

Computing and EE
Heriot-Watt University

later joined by

Ashley Lloyd

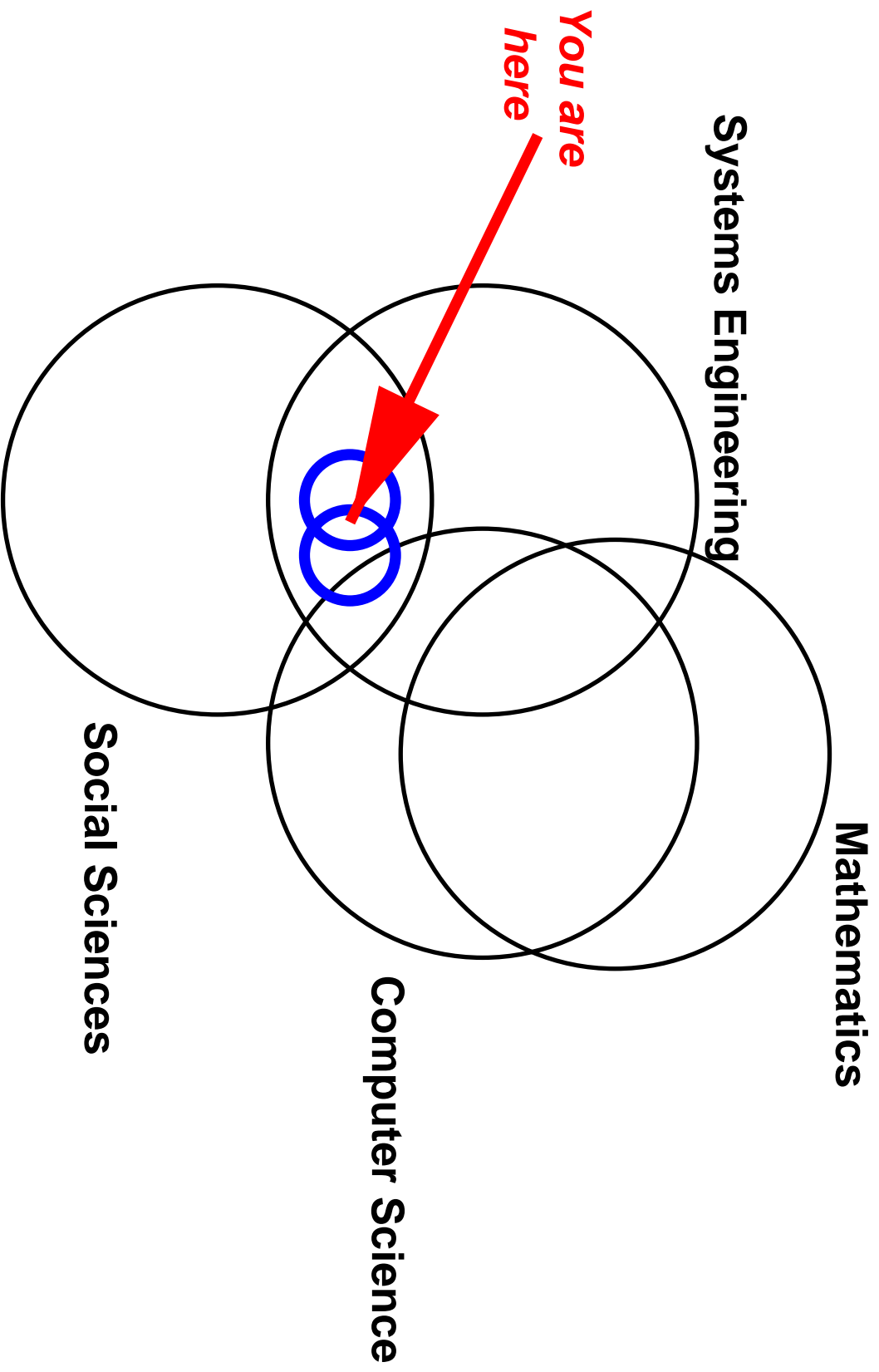
Management School
University of Edinburgh

Rick Dewar

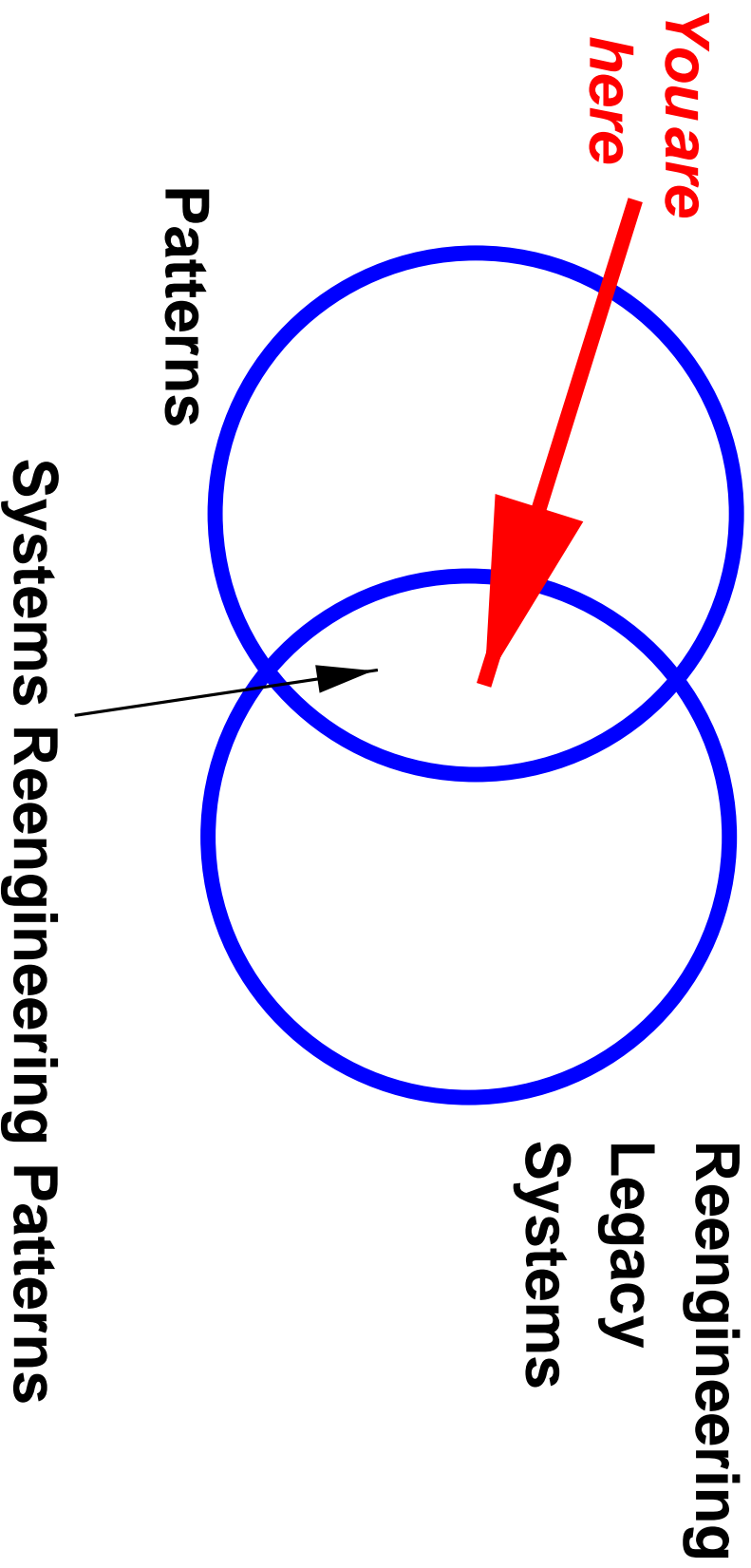
Division of Informatics
University of Edinburgh

<http://www.dcs.ed.ac.uk/home/pxs/reengineering-patterns.html>

Orientation 1:50000



Orientation 1:10000



Plan

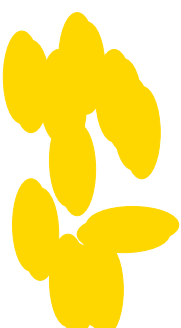
Focus on **identifying** and **communicating** expertise in systems reengineering, using a patterns approach.

- Orientation
- Reengineering Legacy Systems:
 - * What is the problem?
 - * Why is it (so) hard?
 - * What constitutes expertise?
- Patterns
 - * Why is an approach by patterns promising?
 - * Challenges and future work

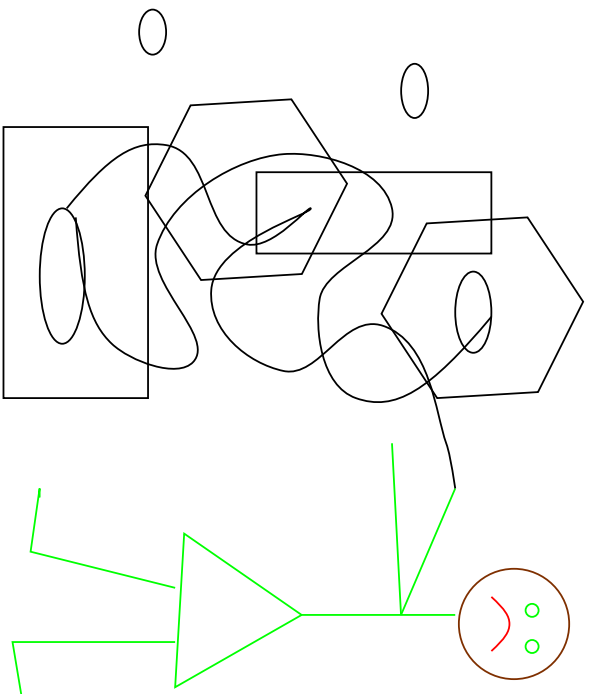
Legacy systems

A legacy system is one which has some value

\$\$\$



but significantly resists modification and evolution to meet new and constantly changing business requirements



23:59
December 31st, 1999



Diversity of legacy systems

A legacy system can have any size or function, e.g.

- Pile of spreadsheets
- Collection of FORTRAN routines for analysing results of physics experiments
- Millions of lines of COBOL, maybe no source code...
- Academic ML program

(Related issues: legacy [processes](#).)

All kinds of organisations have legacy systems.

We're still building them.

Reengineering

Reengineering offers an approach to migrating a legacy system towards an **evolvable** system in a disciplined manner. The process of reengineering may be viewed as applying **engineering principles** to an existing system in order for it **to meet new requirements**.

from Perspectives on Legacy Systems Reengineering, SEI CMU

Sounds like the kind of discipline we know how to deal with. Is there a problem?

Live problem

“Reengineering research has had notably little effect on actual software reengineering practice”

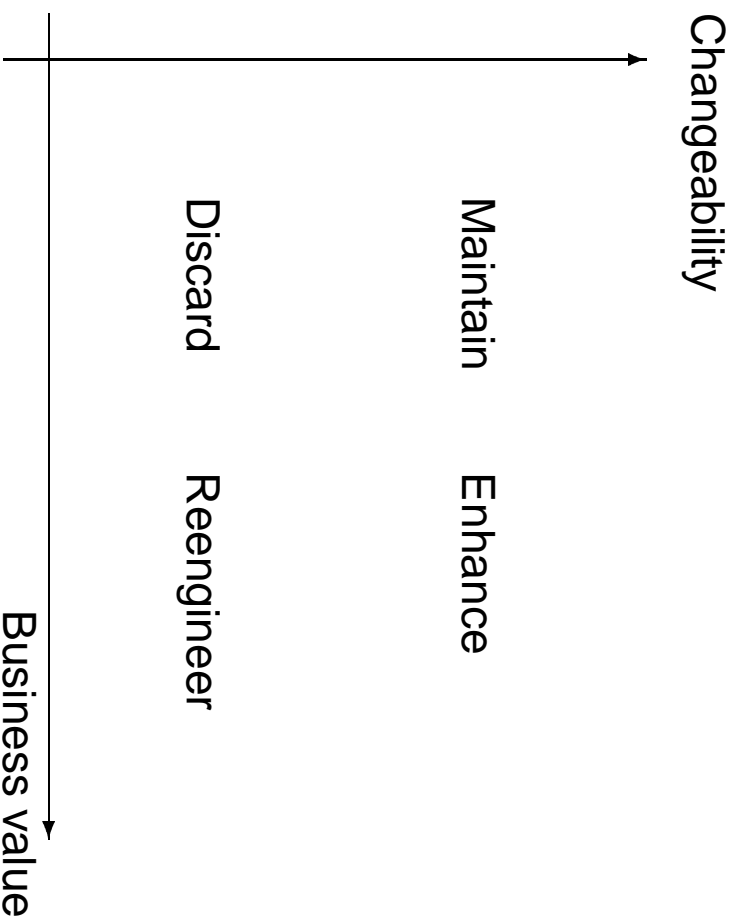
Spencer Rugaber

Linda M. Wills

WCRE 3 1996

- lack of validation?
- lack of education?
- lack of integrated understanding of complex problems?
- lack of **transferable** expertise

Which systems get reengineered?



(How, in detail, do you decide where your system falls? Very skilled task: not our focus here but there's lots of interesting work out there, e.g. US military's SRAH, Brown Morrison and Tilley at SEI, Ransom Somerville and Warren in Renaissance at Lancaster.)

Additional complication

The business processes underneath aren't constant.

technical changes

influence

business changes

Need to handle manageable size problem.

Could just disregard one or other set of factors, studying

- technical aspects of reengineering for well-understood stable business environments, or
- business aspects for well-understood technical environments, e.g....

OK, how do we do it?

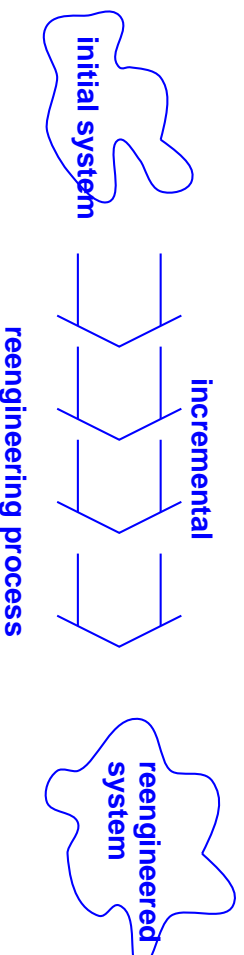
Simplest option:

throw it away and start again

This is not necessarily a bad idea, but it has obvious limitations:

- can you afford to do this?
- what makes you think you can do it better next time?
- do you really know what the current system does, let alone what you want it to do?

We're concerned with the (harder, and more common) case where evolutionary reengineering is essential.



But this is very hard

Even more than in development, there are very many **forces** to be taken into consideration:

- **technical factors:** will this plan give us a working system at every stage? Will it result in a reengineered system?
- **budgetary factors:** can we expect it to be funded at every stage?
- **social factors:** can we get people with the right skills to do it? (People don't like doing reengineering.)
- **personal factors:** can we convince all the people who could get this project cancelled, and can we keep them convinced throughout? (People don't like paying for hypothetical future benefit.)

Much better chance of success with expert involvement

So what is an expert?

Someone who gets it right!

How do people do this?

They **recognise** things about the current situation (the **context**) that are familiar.

Using their **experience** they identify problems that arise in such situations and tackle them: they draw on their past experience of **known-good solutions** and on the **merits and demerits** of each solution.

So we want “instant experts”. Dream on...

More realistically, how can people **learn** to behave more like experts **faster** than by having the experiences and making the mistakes themselves?

Patterns

A pattern is a successful, recurring solution to a common problem in a given context.

- An effectively transferable unit of expertise
- Deliberately not new: embodies “what experts all know”
- Specific to a given context
- Described in a concise, set form e.g.

Name: short, evocative, describing overall nature of the pattern.

Context: a situation giving rise to a problem.

Problem: the recurring problem arising in that context.

Solution: a proven resolution of the problem.

Consequences: notes on the merits and demerits of the resolution described, with references to other possible solutions or relevant patterns where appropriate.

Example: Window Place

Everybody loves window seats, bay windows and big windows with low sills and comfortable chairs drawn up to them...A room which does not have a place like this seldom allows you to feel comfortable or perfectly at ease...

If the room contains no window which is a “place”, a person in the room will be torn between two forces:

1. [S]he wants to sit down and be comfortable
2. [S]he is drawn toward the light.

Obviously, if the comfortable places – those places in the room where you most want to sit – are away from the windows, there is no way of overcoming this conflict...

Therefore: In every room where you spend any length of time during the day, make at least one window into a “window place”.

Alexander et al. *A Pattern Language* quoted in Buschmann et al. *A System of Patterns*

Patterns in software engineering

Patterns imported from architecture to

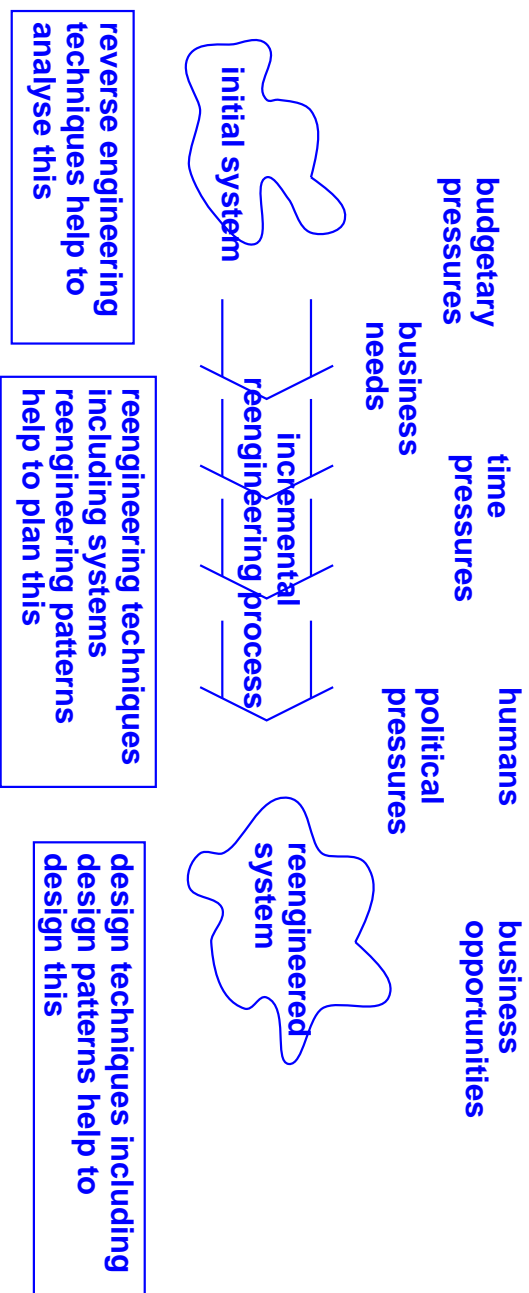
- object oriented software design, by Erich Gamma (PhD thesis 1991, then Gang of Four book)
- programming, by James Coplien (Advanced C++ Programming Styles and Idioms 1991)

Now appearing everywhere from business process reengineering to analysis.

We want to see whether they can help to solve the problem of reengineering legacy systems.

Aim to **complement** other work, e.g. reengineering methodologies (Renaissance, REFITS, Chicken Little...).

Systems reengineering patterns



2 crucial differences from design patterns:

- context much broader
- describe process, not its result

Abbreviated example: Deprecation

Context: Unsatisfactory API. Impossible to change all code that uses the API at once.

Problem: Obvious solution: release new version of API, clients must either change or keep using old version. But this may be unacceptable to clients, also want to discourage use of old versions. And we may not get the API changes right first time.

Solution: Design modification to API which is believed to be an improvement. Add new elements. Don't remove old ones: instead mark them Deprecated. In a later release, remove things previously deprecated.

Consequences: If cases emerge where it is difficult to avoid using a deprecated interface element, the element can be used and the reason for the difficulty examined. It may be that adequate replacement features are not in place. By deprecating the element rather than removing it we avoid presenting the API user with the frustrating situation in which a problem which was soluble using one version of the API becomes insoluble using a later version.

Abbreviated Candidate: Divide and Modernise

Context: legacy system whose database is obsolete and soon to be unsupported. Identifiable area of functionality, relatively well localised, of which a generalisation would be useful, but is not immediately mission critical.

Problem: Modify? Nope. Wrap? Nope, perpetuates the use of unsupported technology. Develop new system “from scratch”? Then will be expected to provide ideal functionality: will be impossible to manage expectations and the project will become huge and correspondingly **high risk** .

Solution: 1. mechanically translate (part of) old DB to new DB.

2. rewrite the relevant code without yet attempting to change its structure
3. remove the now redundant data and code from the rest of the legacy system, handling the consistency and gateway issues
4. consider the reengineering of the now-separated, manageably sized system.

Consequences: Work proceeds in distinct manageable phases. Even if “requirements explosion” does overtake the final restructuring step, the main aim, that of removing the dependency of the functionality on the obsolete technology, will have been achieved.

On the negative side, code and data is migrated before the new requirements on the system are analysed, which means that some of this effort may turn out to have been wasted.

Major outstanding questions include:

- can the problem of data dependencies between the new and the old system be solved, and how? (Consider other material on gateways and middleware...)
- will the restructuring of the new system in fact be [politically? technically?] possible and/or desirable?

Related work

Patterns and reengineering:

- Bern Software Composition Group: Richner Ducasse Nebbe
e.g. Type Check Elimination: Two Reengineering Patterns (EuroPLoP 98)
- Michael Beedle
e.g. Pattern based [business process] reengineering (Object Magazine, 97)
- Alan O'Callaghan at De Montfort University

plus too much to mention on reengineering, process patterns, design patterns, ...

Conclusions 1: promise

Systems reengineering patterns are a way of capturing manageable chunks of reengineering expertise whilst including both business and technical context.

Benefits for:

- validation
- learnability
- tailoring search for relevant material
- bridging the BPR/software engineering gap (more work required..)

Conclusions 2: ongoing work

We are actively working on:

- case studies and interviews: need more DATA!
- relation between business process change and systems change
- optimising pattern structure and content

Please help...

Mailing list, papers, links from:

<http://www.dcs.ed.ac.uk/home/pxs/reengineering-patterns.html>