ABSTRACT
I argue that socio-technical systems are evolving systems. Therefore an evolutionary stance should be taken in understanding workplace studies, approaches to socio-technical systems design, and conceptualizing the CSCW research agenda. The mythology of CSCW holds that the role of workplace studies is to inform design; instead, I argue their role is more akin to reverse engineering. Systems design, in turn, is co-evolutionary forward engineering, employing evolutionary ‘good design tricks’ that Dennett terms cranes. Between reverse and forward engineering new ground opens for the conceptualization of the CSCW research agenda as an interdisciplinary journey to find, design and understand cranes.

Keywords

INTRODUCTION
Conceptualizing the design of socio-technical systems is a hard problem, and the absence of any good ways to do this hampers the growth of CSCW as a field. The mythic tradition in CSCW is for designers to look to ethnographic study to inform design – to give the technology side of the CSCW house the ‘skyhooks’ from which innovative design can proceed. Many on the ethnography side have challenged the myth, arguing that this view is unreasonable or unrealistic [24], or have noted that the process doesn’t work this way, without explicitly challenging the myth [20].

The argument in this paper is that socio-technical systems, such as CSCW systems, grow in an evolutionary way. Understanding how evolutionary processes account for ‘design’ gives a new perspective both on understanding the role of workplace studies, and the problem of designing socio-technical systems, as well as reconceptualizing the role of CSCW research.

In Darwin’s Dangerous Idea, [7] Dennett describes the concept of evolution as a ‘universal acid’, an idea that can eat “through just about every traditional concept, and leave in its wake a revolutionized world-view, with most of the old landmarks still recognizable, but transformed in fundamental ways” (p. 63). In this paper, the goal is to pour some of this universal acid on questions of CSCW research and design. At one level nothing changes – we still need to build complex socio-technical systems for use by collaborating groups. At a deeper level, our view radically shifts by taking an evolutionary perspective. In short, rather than imagining that there is a privileged designers stance or perspective, design can be seen as stepwise uncovering of evolutionary ‘good design tricks’.

I argue that complex socio-technical systems consist of co-evolving, interacting subsystems that exist in some environment, where perturbations to one subsystem or to the environment create the possibility for evolution elsewhere in the system. ‘Design’, in this model, is the series of evolutionary moves that the individual subsystems make over time. Quite frequently, ‘useful moves’ turn out either to be unanticipated a priori, or to have unexpected and unintended consequences.

This approach opens several interesting conceptual issues. From a ‘pure design’ perspective, this stepwise, evolutionary approach is an interesting candidate for solving wicked design problems [21] (the design of complex socio-technical systems is widely considered to be wicked). From a more technical perspective, this viewpoint relieves designers of the need to find ‘silver bullets’, or skyhooks – magic design perspectives that miraculously solve the problem of building complex socio-technical systems such as CSCW systems – and instead allow us to take a more emergent view of systems development. From a more abstract point of view, the model opens the possibility of replacing the ‘mythic design tradition’ in CSCW by a more grounded and appropriate role both for all sides of the CSCW house.

Evolution is a contentious topic, and therefore I begin the body of the paper by outlining what is not being argued in the hope of heading off anticipated reader concerns. Next I present a model of evolutionary processes, drawing primarily on Dennett and Kauffman [18]. I then give two examples of evolving socio-technical systems. I conclude with some more general argument about the implications of this view for CSCW research.

WHAT WE’RE NOT ARGUING
First, this approach is not ‘greedily reductionist’ [7]. I am not arguing that design of CSCW systems can be reduced...
to an underlying set of principles that come from another
discipline, e.g., biology. Rather I am arguing that there is a
more sophisticated way of conceiving of the design process
than the ad-hoc approaches currently employed. Design of
CSCW systems (and other socio-technical systems) shares
a method in common with design in other kinds of evolution-
ary systems but is not necessarily reducible to them.

Second, this approach is not determinist, predictive or pro-
gressive. I specifically do not believe that evolving sys-
tems make ‘progress’ towards a design goal (the Ultimate
Collaboration Environment, or the New American Male).
They do progress in the sense of moving through a design
space (explained below), but this is not progress towards
an end or telos. Hopefully, designs are progressively more
useful, relative to some criteria, but this is not necessarily
the case.

I am not taking a position that is insensitive to, or deni-
grates, social complexity, but rather a position that accepts
and (partially at least) accounts for it.

I do not discount the role and importance of human agency
in design. Almost all the design talked about in this paper is
design by humans. However, I do believe that the history of
any design process is evolutionary and stepwise.

Although I use the term ‘model’ in the paper in various
places, I am using the term informally. Some aspects of
what is described here may well be amenable to positivist,
formal modeling, but in general I do not believe this is so.

DARWIN’S DANGEROUS IDEA

Dennett’s book Darwin’s Dangerous Idea includes an ex-
tended argument in favor of an evolutionary approach to
understanding design in general. Dennett argues that the
‘design’ of organisms, social behaviour and technologies
all follow the same basic evolutionary principles. There is
no room in this paper to do more than sketch the argument,
which goes as follows.

Evolution is an algorithmic process. The salient features
of this algorithm are:

• It is substrate neutral. Thus it could be executed by RNA,
or in silicon, or by humans.

• Although the algorithm may be brilliant, or have brilliant
outcomes, each constituent step, as well as the transitions
between steps, is utterly simple.

• It has guaranteed behaviour: i.e., whatever it does, the
algorithm will always do it, provided it is executed with-
out misstep (Of course this behaviour may include loop-
ing forever, or producing a mess, but it will be guaranteed
to do this every time on the same input).

This idea, Dennett argues, is Darwin’s universal acid, by
which he means that while many processes may be seen to
require some form of outside agency to achieve their out-
comes, they can, in fact, be understood as the behaviour of
a reasonably simple algorithm.

What is this algorithm? To explain this, Dennett introduces
the notion of a Design Space. First, we need a basic con-
stituent element set. Examples might include all books that
could potentially be written in English, or all possible gene
combinations. Clearly, these spaces would be enormously
large and multidimensional.

Any ‘artifact’ – book or genome – would have a point in
this Design Space that represents it. And ‘near’ this point
are all the variations and mutations of this artifact.

Dennett’s evolutionary algorithm is a random walk through
this Design Space. Given a particular point in the Space,
the only possible action we can take is to move to a nearby
point in the Space. Consider a genotype representing an
organism. It is located at a particular point in Design
Space. Any evolution would move its genotype to a nearby
point in the Space. The organism that results might be ‘bet-
ter’ or ‘more fit’ and thus more able to survive and procre-
ate, or it could be less fit and perhaps die out (more on this
in the discussion of fitness landscapes below).

Design Space can be generalized to contain all possible
species (artifacts and systems) that could ever be built, of
whatever type, including all the meaningless ones. Thus,
any design can be located in Design Space. Of course it
should be remembered that this is an abstract, conceptual
Design Space – it is too large to be represented.

It’s convenient to introduce a vocabulary associated with
movement in Design Space. We say that an evolutionary
move that improves a design according to some criteria
lifts the design in Design Space. There are a number of interest-
ing concepts that follow from this. These include:

Skyhooks. A skyhook is a mythical device that can sus-
pend something ‘from nothing’ (literally, the sky). Sky-
hooks, of course, don’t exist. Dennett introduces the term
to refer to design moves that require an intelligent outside
agency. Skyhooks, from the evolutionary point of view,
are bad, as they imply some design moves simply won’t
happen without outside agency – in other words the evolu-
tionary algorithm is insufficiently powerful to ‘generate’
these moves. Skyhooks are often referred to as ‘silver bul-
ets’ in the software community. Denying skyhooks does
not deny or denigrate the possibility of human agency, but
merely moves it to the next paragraph.

Cranes. A crane, by contrast to a skyhook, is a “subprocess
or special feature of a design process that can be demon-
strated to permit the local speeding up of the basic, slow
process of natural selection, and that can be demonstrated
to be itself the predictable (or retrospectively explicable)
product of the basic process” [7 p. 76]. In other words, a
crane is an evolutionary short cut that is explicable in terms
of the evolutionary algorithm. Human agency that designs,
implements or uses cranes is quite acceptable, and usually
(always, if you believe evolutionary theory), apparent sky-
hooks can be explained away as cranes.

Good Design Tricks. A Good Design Trick is a crane that
gives a species (organism or artifact) a significant edge in
the evolutionary race. Examples of evolutionary Good
Tricks in organisms are locomotion and eyes. Examples of
Good Tricks in cars were windshield wipers and assembly line construction. Good Tricks in CSCW systems have been the HTTP (web) and SMTP (email) protocols.

**Fitness Landscapes.** To illustrate this idea, consider a technical artifact (like a Palm Pilot). Now consider a three-dimensional ‘mountainous’ landscape, like the Alps. Altitude in this landscape stands for ‘fitness’ (for example acceptability in the market), and latitude/longitude for some design factors (such cost, user interface characteristics, wireless network access, or available memory). Your Palm Pilot will ‘place’ itself somewhere on this Palm Pilot landscape (depending on the design factors of your particular model, and any options you might have chosen), and have a certain fitness (altitude) as a result. Now, evolve the Pilot in some way (add memory, remove the color screen, add a wireless networking module). The ‘new’ Pilot will be placed in a different position on the fitness landscape. The ‘altitude’ of this position will be higher or lower than the altitude of the previous position: the new Pilot will be more or less ‘fit’. Fitness landscapes, relate directly to Good Design Tricks: a Good Trick should cause one to ‘lift’ in design space, in other words to move to a higher altitude on the Fitness Landscape.

There is, however, is a crucial point: the fitness landscape is not static. Changes to the environment, for example increasing acceptability of personal information managers as a ‘necessary work tool’, or changes to other, related artifacts, can cause the fitness landscape to deform. If, for example, a new competing product appears on the market, the perceived cost/benefit can change, and the Pilot might become too expensive (less fit) or more acceptable (more fit). Thus a Good Design Trick may cease to be good, over time, and artifacts that appear to be less fit may suddenly become more so.

**KAUFFMAN’S PATCHES**

Stuart Kauffman [18] has been investigating the relationship between chaos theory and evolution (the point being that evolution requires the right conditions, and these emerge on the boundary between order and chaos). Like Dennett, he argues that the evolutionary model applies as much to culture, economies or artifacts as to biological organisms. He argues that species do not simply evolve; they co-evolve, i.e., the evolution of any species is dependent on the evolution of other species with which it is interdependent, as well as responding to changing stimuli from the environment. Changes to one species will affect the evolution of others, in the following way:

- Any species will attempt to maximize the chances of its own survival (in effect by evolving to a locally optimal ‘fitness peak’ on its fitness landscape).
- This will cause the fitness landscapes for all interdependent species to deform. It is thus possible that these species, instead of sitting on a locally optimal peak in their fitness landscapes, will find themselves ‘shifted’ up or down as the landscape deforms, and will then evolve further, or go extinct, in response.

- Indeed, this deformation is a crucial part of ongoing evolution: because the fitness landscape changes, further evolution then becomes possible. Without these deformations, evolution would become ‘stuck’.

To explain this, Kauffman constructs a thought experiment: Imagine a complex system in which many parts interact. For our purposes, we are particularly interested in socio-technical systems, so let’s choose one as an example: a multi-group software development organization. Divide this system into a number of parts called patches. For example, patches could be groups in the organization or development teams. Now allow each patch to locally optimize itself (evolve), if necessary at the expense of other patches. When any patch optimizes itself, for example by changing a software development practice, a hiring practice, or an interface, or introducing a new functionality to a part of the system, this causes the fitness landscapes of all patches to deform, which allows the other patches (groups) to re-optimize themselves, and so on. Once a patch has evolved to a local fitness peak, it can’t evolve any further; it is stuck until its fitness landscape deforms.

This implements a process of co-evolution: each patch or part of the problem is trying to achieve the best possible local situation. And if this impacts negatively on other parts of the ‘system’, too bad. Senge gives this phenomenon a name: the ‘beer game’ [27].

While this might seem harsh or unrealistic, it appears to be borne out in practice; in field work we have undertaken, we have come across examples of this kind of co-evolution in action, in which different parts of an organizational system locally optimize in ways that are often detrimental to other parts of the organization, or the system as a whole.

Thus, in one real-world situation we have studied the support part of an organization used to hire junior software developers to man the help desk. As these people became more experienced with the systems, they migrated to maintenance and then development groups. The helpdesk manager chose to stop hiring software developers (because of problems he was having with turnover and training), and instead promote administrative staff within the company to the helpdesk, as these people were seen to have a more limited career path and thus more likely to stay in place. This in turn has had an impact elsewhere in the company.

What is happening here is that each subgroup represents a patch or species, changes in hiring practice are local optimizations on the deforming fitness landscapes of the patches, and the entire organizational ‘system’ is co-evolving (in this case quite possibly to the detriment of the organization as a whole).

We’ve also seen situations where it’s blindingly clear to the outside observer of a system that some kind of change is needed (a different work practice, for example), but the

---

1 This study is as yet unpublished; but the insights we draw from it here are hardly novel.
‘system’ seems incapable of adapting. Why? I suggest it is because the system has somehow locally optimized itself, is thus sitting at a ‘fitness peak’ and it cannot optimize further until its fitness landscape deforms (for example the organization experiences economic problems or gets taken over).

Patch size (or in other words, what we consider to be a species) turns out to be very important. If the size of the patch matches the size of the system, the system has very poor optimization characteristics – a ‘local peak’ on the fitness landscape is reached quite quickly, and the system ‘sticks’ there. It cannot move because there can be no deformations (as the entire system is one ‘species’), unless these come from outside. When deformations do occur, the re-optimization is very complex because of its sheer size.

On the other hand, if the size of each patch is very small, the system will likely enter a chaotic state. With the right patch size, the various patches are capable of evolving quickly in response to changes while still maintaining a certain stability. Intuitively, this seems reasonable: an organization that manages everything centrally is usually inefficient and unwieldy; an organization that gives everyone too much autonomy is chaotic; but with the right group size and interdependencies, decentralized organizations can operate reasonably well.

Thus for organizational evolution, there are good intuitive ‘sizes’ for patches, even though they might differ from situation to situation: group sizes within organizations are a good starting point (of course not all patches need be the same size).

What about technology? If we consider any fairly complex system the same extremes seem to apply. Large-scale pieces of technology (for example corporate information systems) are notoriously unable to adapt. Many small pieces of technology that can change independently lead to chaos. Between these extremes lie interesting patch sizes.

The ‘patch model’ seems to imply species stability – the set of species that co-evolve is somehow fixed. This is not the case. Any evolutionary move could potentially drive whole classes of species to extinction, and create the opportunity for new species to emerge. For example, the advent of MSDOS drove many competing PC-type operating systems to extinction. The advent of the PC opened up the possibility for many new application types, such as the spreadsheet. Initially there was one spreadsheet (VisiCalc), then a number of competing spreadsheet products, which eventually settled down to a couple that now co-evolve (features added to one quickly appear in the other, and so forth). A similar argument can be mounted for Web browsers. Generally, therefore, we can conceive of the family of ‘patches’ or species and their interdependencies changing over time.

**BRAND LAYERS: PATCHES IN ACTION**

To give a feel for the patch argument, consider Stewart Brand’s work on understanding what seems on the surface to be an unlikely candidate for a evolving system: buildings [4]. He argues that buildings evolve in time, and do so by ‘layering’ themselves into a number of subsystems, each of which can then evolve reasonably independently, as shown in Figure 1.

At the top of the diagram, the ‘stuff’ layer evolves most quickly as furniture, chairs, books etc all constantly move around. The second (space) layer evolves as the space plan for the building (rooms, offices, etc) changes, and so on. Note how ‘space’ can evolve independently of ‘structure’ (the structural aspects of the building), although some space modifications will have structural implications (moving a load-bearing wall, for example). All these changes are results of various environmental pressures (such as adding staff) and/or evolutions to other layers.

I suggest that, for buildings, the Brand layers constitute a useful first approximation of Kauffman patches, and that any evolution of one layer will eventually lead to evolution of (some of) the other layers. So this is one kind of conceptualization for technological patches. Assuming that ubiquitous computing or smart rooms become commonplace, it’s likely both that the family of layers will change (new species introduced) and the ways in which they impact on one another will alter.

**EXAMPLE: EVENT NOTIFICATION SYSTEMS**

The major example of a co-evolving system I will consider in this paper is the event-based notification work in which we have been involved for almost a decade. The purpose here is not to describe in detail the ways in which these tools have been used, or the impact (social and technical) they have had. For this information, see [9, 12]. This example is chosen because it is ‘at hand’ – I have enough familiarity with it to draw upon its history. This work is not part of the trajectory of development and analysis around the Elvin project. Rather, the point is to demonstrate (informally) that the mechanisms used for the growth of this system appear to follow the processes outlined by Dennett and Kauffman. In other words, the evolutionary design process can account for design and evolution of socio-technical (and particularly CSCW) systems.

<table>
<thead>
<tr>
<th>Stuff</th>
<th>Space</th>
<th>Services</th>
<th>Structure</th>
<th>Skin</th>
<th>Site</th>
</tr>
</thead>
</table>

Figure 1. Brand Layers for Buildings
A Publish/Subscribe notification service **decouples** producers of events from consumers via a notification server. Producers create notifications and send them to a notification server. Consumers subscribe to certain patterns of notifications by sending subscription expressions to the server, and when matching notifications arrive at the server, they are sent to the appropriate consumers. Figure 2 illustrates the general architecture. In general, producers have no idea which (if any) consumers will pick up their notifications, and consumers have no idea (unless it is encoded in the notification) which producers generated a notification. Thus producers and consumers are strongly decoupled.

The first event notification service we built was the UIUC Message Bus [5, 16, 17]. This system used a publish/subscribe architecture and **semi-structured** messages (analogous to email, with headers). Subscription expressions were simple regular expressions over the headers. The message bus was used to integrate a number of different pieces of technology, notably the various parts of the ConversationBuilder workflow system [17]. Choosing to use the message bus as our integration substrate led to the development of a strongly-decoupled architecture for ConversationBuilder, including a hypertext front-end with embedded links and graphics. As an interesting example of co-evolution in action, an undergraduate working on the user interface parts of ConversationBuilder went on to build NCSA Mosaic – an illustration of how slowly evolving work can cause a (completely unexpected) explosion of new species.

The UIUC message bus was not used for human-human interaction, but for tool integration. The system was limited in the kinds of subscriptions and events it could support, because of the semi-structured nature of the messages, the kinds of programming interfaces it provided, and performance limitations of the server.

Elvin [3, 12, 26] is an evolution of the message bus model. In Elvin, notifications describe events using a set of named attributes of simple data types and consumers subscribe to a ‘class’ of events using a boolean subscription expression. When a notification is received from a producer, it is compared to the consumers’ registered subscription expressions and forwarded to those whose expressions it satisfies. An example of this is given in Figure 3. There is no notion of ‘header’ fields in Elvin, so it is a content-based publish and subscribe service. The content-based approach was initially chosen to allow more flexible kinds of systems integration and network management applications. But – as ever with evolution – this change to the ‘fitness’ of event processing caused a co-evolution elsewhere.

It turned out to be simple to plug many different kinds of information sources into Elvin, such as usenet news, because there are no centralized configurations or channels. This led to the development of a notification display. Elegantly, the developers chose a scrolling one-line marquee or *ticker* (see Figure 4). Messages scroll across this marquee and fade according to user-specified parameters.

For debugging purposes, it made sense to be able to generate events easily, so a dialogue window was added to the ticker to allow events to be sent. This rapidly evolved into a simple chat interface, and from this grew the notion of a chat group, where the ‘group name’ is simply a distin-
guished field in a notification. Many users could thus subscribe to the same chat groups: a practice of using the ticker to chat amongst group members, or across the DSTC as a whole, began to grow. Interestingly, the social practice of chat was one in which ‘being overhead’ was considered usual, and serendipitous interaction and tangential discussion is welcomed.

The number of chat groups rapidly grew, as did use of the ticker itself: a subsidiary editor to manage group subscriptions was added to the ticker. Many users implies interest in a wider range of information: new producers were added, for example ‘scraping’ news off major news carrier sites.

One-line chat is all very well, but more information is sometimes needed: the ticker messages developed an ‘attachment’ field that could be used for any MIME-typed information (such as email messages, file paths or URLs).

Several competing subspecies of ticker evolved (different versions each in Java, Python, and C, and on both Windows and Unix/X) and competed for a while. Two survive.

Other clients, such as a coffeebiff, that could be used to notify users about who was in the coffee room, appeared and evolved through several versions quite quickly before settling down into a stable niche that it has occupied, unaltered, for almost two years. Users became tired of ‘missing’ notifications on the ticker; so alternate ‘chatbox’ style interfaces appeared which did not delete information (several of these competed for a while, but only one survives).

Information repositories were instrumented. Thus, for example, whenever a programmer checks a piece of code into our CVS repository, the other programmers in the group get a notification. This has made it much easier for programmers to keep abreast of each other’s work, and has encouraged interaction and awareness.

This also points to a more profound change that was occurring. We were beginning to re-visible work and expertise. Generally, computer systems are very good at making work invisible – if you look around, all you can see are workers typing and mousing, but not what they’re actually doing. Operating systems are designed to give the illusion that one is the only user of a system, so it’s difficult to get a sense of what others are doing [23]. Elvin was beginning to place notifications in places that helped us to ‘see people working’ once again. Similarly, the ticker allows one to (politey) ‘shout out’ for help without disturbing others, and if someone in the organisation with the appropriate expertise sees the ticker message, you get help.

There are several changes in social behaviour and work practice that followed from the adoption of ticker technologies: work-at-a-distance has been eased within the DSTC, temporally limited emails have been reduced (moved to ticker, where messages expire automatically), use for game playing and chitchat has grown. The social changes implied here are complex and subtle. We have been analysing implications for work practice of changing use of Elvin and its clients, and will report these results elsewhere.

Technical co-evolution of Elvin has continued in parallel with this social co-evolution. Once producers are freed of the responsibility to direct notifications, the determination of the significance of a state change becomes less important: they can promiscuously notify any potentially interesting information, and rely on the notification service to discard notifications of no (current) interest to consumers.

While useful from a user's perspective, large volumes of unused notifications are costly in terms of computational resources. To overcome this problem, Elvin has been developing a quenching mechanism that discards unneeded notifications without consuming resources, allowing producers freedom in determining event significance.

Several groups outside the DSTC now use Elvin (as well as a number of users worldwide), and this has introduced new challenges. We would not, in general, want all notifications generated within DSTC to go to another institution, such as an undergraduate dorm (or vice versa). Nor would we want to overload our server with their traffic. But some notifications can usefully flow between these sites. When Elvin was first developed, a simple federator for interconnecting Elvin servers was developed but had been discarded because it served no useful purpose (one server could handle all internal traffic). This federator was revived and extended to allow controlled sharing of notifications across multiple sites. This in turn has caused a change to what information flows in what groups, so as to ensure appropriate spread of traffic.

An increasing user community sending many messages implies that, sooner or later, privacy concerns are going to become critical. Earlier versions of Elvin did not have any kind of security systems. Now mechanisms are being built that allow for secure, private messaging and notification. This in turn will cause changes to use, as many possible uses of the system are simply avoided by users because they know that certain of our colleagues engage in snooping as standard practice.

Figure 4 shows screen dumps of a number of ‘ticker instances’. The first illustrates tickertape being used to track down an expert; the second various kinds of automatically generated notifications (from the calendar system and from Usenet news. The third illustrates a tickerchat interface (from Xtickertape): here the discussion is triggered by an automatic notification (from a CVS checkin event) and moves on to uncovering and repairing (not shown) a bug in the X-windows version of tickertape. More detail on these stories can be found in [9].

It is possible to use Elvin and the tickers to convey a certain amount of awareness information, but in general specific user action is required. For example, we could easily imagine a ‘who’s around’ status board, but users would have to manually indicate their status (except in some special cases, like inference from machine idle time). This is problematic as people forget to update the board (this is a problem with uptake of the CoffeeBiff, for example). It would be much better if the board could update automatically.
We’re therefore investigating inputs from the ‘physical world’, such as RFID trackers attached to users and mobile equipment. This will probably cause large volumes of ‘low-level’ events, so we’re also looking at event correlation – determining higher-order events from lower-order ones, for example a series of events that indicate that Simon is in his office and his lights have been turned on and his computer booted up all probably indicate that Simon is in and starting to work.

**THIS IS A STORY OF CO-EVOLUTION IN ACTION**

The Elvin story illustrates the ways in which technological and social practice appear to co-evolve following the mechanisms outlined by Kauffman and Dennett.

While the developers of Elvin would argue that they have ‘designed’ the system (in a traditional sense), I suggest that in fact this has been an evolutionary process that closely follows the model of co-evolution on deforming fitness landscapes suggested by Kauffman.

We have many patches in this story:

- The notification server
- Other pieces of infrastructure such as the federator
- Client species such as Tickertape (various subspecies), Tickerchat (various subspecies), CoffeeBiff
- Input species such as the Ticker chat messaging subcomponent.
- Producers, such as the Usenet producer, Email producer, CVS, Calendar and other repository producers.
- Users, with varying degrees of technical skill and interest.
- The Elvin developer community

The impact of changes in one ‘species’ here seems readily traceable to effects in others. For example, in the beginning, with low traffic volumes, the tickertape interface proved to be very useful – it was fit, or well suited to its purpose. However, increasing volumes of messages and more ‘useful’ discussions caused frustration on the part of users – they were missing information and were left feeling frustrated. The fitness landscape for tickers had been deformed by the evolution of the producer and chat patches. The result was a series of evolutions of tickers (in Python and Visual Basic) and a set of chat-window extensions to the Xtickertape, as illustrated in Figure 4. There has also been evolution of use – many PC users now run tickertape to ‘keep an eye on things’ and tickerchat to refer back to if they miss information. Xtickertape users can do the same thing with one tool. I predict that a similar combined tool will soon emerge on PCs.

The addition of new, external Elvin user communities created a need for the federator. This is a wonderful example of co-evolution: a piece of technology which looked like a crane, turned out to be unfit and fell into disuse, later turning out to be a Good Design Trick after all and returning to useful life. Adding the federator has expanded the use of ticker to many new sites, and has increased the need for security and privacy facilities, which are now being added.
To keep the Elvin story simple, environmental pressures were omitted above, but it is useful to consider them now. One of these pressures has been resource limitations: at any time there are several competing demands for development, and limited resources available. At each point, a decision has to be made as to what shall be allowed to evolve. That decision is determined by what is needed most, what seems possible with available resources, and what seems to best fit the ‘longer term strategy’ of the project.

Making these decisions (i.e., choosing what comprises ‘needed most’, ‘seems possible’, ‘best fit’) looks like a sky-hook but is in fact a crane: these choices could have been made by ‘natural selection’, but the crane (human intervention) speeds up the choice. Importantly, the choices could be wrong. We don’t know that the changes are facilitating progress towards a goal. The more general lesson here is that ‘human agency’ has no need to be a skyhook and can quite happily be a crane, and that ‘needed most’, ‘seems possible’, and so forth are simply positions on the fitness landscape.

Other environmental pressures have included the pressure to commercialize Elvin, which has impacted on security and is driving a move to internationalize the next version.

This history closely tracks Dennett’s evolutionary Design Space model. We started with a very simple publish-and-subscribe notification model, and came across some extremely useful cranes that were Good Design Tricks, of which the most important are probably content-based subscription, the ticker interface, and federation. These then shaped and drove the use of the system as a whole in unexpected but useful directions (for example, social use of Elvin, for communications and chat).

Other techniques that look like Good Design Tricks are security, quenching, capturing events from the physical world by instrumenting doors, attaching RFID tags, etc., and event correlation. However, we don’t yet know if these ideas and their implementation will prove to be well-adapted for actual need.

Although the language of evolution that we use here might seem unusual to those with a computing background, the underlying ideas have shown up in the systems development literature before. Richard Gabriel articulates a development philosophy that he calls ‘worse is better’ [13]. The essence of this philosophy is to build small, incomplete, useful implementations of systems quickly, and release these to the market, and then adapt them in response to user needs and the market. In effect, this is a co-evolutionary strategy, and Gabriel argues that the software products that are generally seen as ‘successful’ (e.g., Unix or most Microsoft products) have followed this development path.

THE EVOLUTIONARY PATH IS NOT ALWAYS SMOOTH
Not all technological innovation will necessarily cause social co-evolution, or vice versa. The QWERTY keyboard is a nice example of this. Other, ‘better’ design alternatives have been proposed and have gone extinct (or nearly extinct) because the ‘social patch’ simply failed to evolve in response to the technical change. Sometimes (often?) co-evolutionary effects are completely unanticipated. ConversationBuilder was a project about workflows and collaboration, but the technologies that have proven to be useful and have survived have been an editor (Xemacs), and influences on the concepts of web browsers and event notification systems, none of which were in the ‘conscious’ design path of our research. Indeed, we failed to find a skyhook – the ultimate CSCW framework – and instead found some very interesting Good Design Trick cranes along the way.

THE EVOLUTIONARY STANCE AND THE CSCW RESEARCH AGENDA
Besides being a useful way of understanding how systems ‘grow’, Darwin’s universal acid has an impact on the CSCW research agenda. I suggest that the research agenda of CSCW might be reconceptualized in evolutionary terms, and that this reconceptualization covers both sides of the ‘CSCW house’. While this will probably make many readers uncomfortable, well, universal acid is just like that. In particular I want to address three questions:

• What is the role of workplace studies in CSCW?
• How should we conceptualize the CSCW research agenda?
• What is the implication of the evolutionary stance for the builders of CSCW tools and environments?

The Role of Workplace Study
Dennett argues that one of the interesting things one can do with evolutionary systems is to engage in reverse engineering: trying to uncover answers to ‘what’, ‘how’ and ‘why’ questions. Reverse engineering is the practice of taking a system and trying to understand what it is for, what it does, how it does it, and why it works like it does. For our purposes I’m interested in a very rich conception of ‘system’ here, namely complex socio-technical systems.

I suggest that reverse engineering of complex socio-technical systems is exactly what workplace studies are for. There are marvelous examples of this in the literature, for example Heath and Luff’s London Underground study [15] or the Lancaster ATC studies [14]. Schmidt argues “those workplace studies that have had the strongest influence on CSCW research have been studies which did not aim at arriving at specific design recommendations … but instead tried to uncover … the ways in which social order is produced in cooperative work settings, whatever the design implications of the findings may be” [24]. In other words, those studies that have engaged in reverse engineering.

While this might seem like a strange formulation, in fact it helps to clear up a profound misunderstanding about the role of workplace study. The mythology of CSCW states that the purpose of workplace study is to help with design (i.e., forward engineering) of CSCW systems. But in fact the role that workplace study actually plays is to perform
reversed engineering.

The CSCW Research Agenda

In contrast to reverse engineering, forward engineering is the conscious application of cranes to force moves in Design Space. This begs the question of what cranes we are after, how we might find the cranes, and whose job it is to do so. It is these questions that opens up ground between reverse and forward engineering – new ground for both the social and technical sides of the CSCW house. I suggest that it is finding cranes that is the joint research agenda of the CSCW community. Gale Moore put it better: “it is finding, designing and understanding cranes that situates the CSCW research agenda” [Moore, personal communication]. Indeed, if all we do is workplace studies or systems building, we may as well be ethnographers of computer scientists. It is the work we jointly do as crane uncoverers that is the essence of interdisciplinary CSCW research.

The CSCW community has already uncovered many successful cranes, some more social, some more technical, some straddling the gap, and usually the result of interdisciplinary collaboration. Examples include articulation work, awareness [22], situated action [29], the Locales Framework [8], chat, email, some aspects of CSCW systems architectures [25] such as MUD-type environments [6, 19] and so on.

I cannot suggest a ‘method’ for crane uncovering here – rather, I believe that cranes emerge (or co-evolve) over time. By actually paying attention to these as moves within the design space rather than as technical or even socio-technical problems or solutions our engagement with Design Space is more open and dynamic.

One crane that might prove to be useful, and should be investigated more directly, would be to use Alexander’s Pattern Languages [1, 2] as a rich mechanism for capturing complex socio-technical constraints and design alternatives. This role as repository of deep insights grounded in the social is precisely the way in which Alexander used Pattern Languages in the built environment, and they read like poetry as a result. Each pattern is a crane for designing part of a building, neighbourhood, city or town. Perhaps the same ideas could be applied in the socio-technical domain as well. Interestingly, Alexander takes an explicitly evolutionary view of pattern languages, and, while he does not use the language, clearly sees that the role of patterns is that of cranes.

Part of the issue when trying to write pattern cranes will be the question of how to decompose the systems under study. One possible solution is decomposition by aspect. Brand broke down the ‘structure’ of buildings by aspects that split in time. Fitzpatrick views workplace study through aspect prisms (civic structures, presence/awareness, trajectory, and so forth: the elements of the Locales Framework [8, 10, 11]). This may prove to be a useful way of decomposing problems for patterning.

Evolutionary Systems Building

Socio-technical systems, according to the argument we have developed above, evolve. There are no skyhooks or silver bullets, and any piece of (new) technology or social practice will always be an evolutionary step away from prior technology – and we will get there with a crane.

Horst Rittel [21] introduced the powerful notion of wicked problems. Many (if not all) interesting design problems (especially in CSCW) are wicked. For a wicked problem:

- It is not possible to articulate a problem independent of a solution. In other words, the problem statement and solution co-evolve.
- There are no ‘stopping rules’, in other words one stops when a ‘satisficing’ [28] solution has been found as opposed to when an ‘optimal’ solution has been identified or a solution has been ‘proven’. (From Kauffman’s co-evolutionary viewpoint, eventually perturbations to one patch, or subsystem, will be less likely to cause perturbations in another).
- Solutions are ‘good’ or ‘bad’ (relative to some usually subjective criteria) as opposed to ‘right’ or ‘wrong’.

Returning to the Elvin story for a moment, we primarily use Elvin these days for social and awareness notification of interesting events – news stories, changes to software repositories, chat. Designing a system to support this mix of interactions is a wicked problem. The co-evolutionary process ‘found a solution’ for us.

I suggest that CSCW problems in general are wicked, and a co-evolutionary strategy is an efficient and useful way of finding solutions. The strategy can be enhanced by explicitly engaging in forward engineering of the systems – application of cranes to speed the movement of the system through Design Space. Although radically different to the traditional view, it perhaps more useful.

CONCLUSIONS

I have argued in this paper that complex socio-technical systems should be conceived as co-evolving. Taking the evolutionary stance opens up new ways of conceptualizing the role of workplace study in CSCW, the research agenda of CSCW, and approaches to systems building, as well as uncovering a common linking that runs through all of these. No doubt this paper will generate a lot of controversy; but that controversy will only help, in the best evolutionary tradition, to shape and drive the evolution of the ideas themselves.

ACKNOWLEDGMENTS

Thanks to Gale Moore for her engagement with these ideas, without which this paper would never have (co-)evolved to its present form; to my colleagues Bill Segall, David Arnold, Ted Phelps, Julian Boot and Blaize Rhodes for their work on the Elvin system and its clients; and to Geraldine Fitzpatrick, Kate Horsey, Michael Docherty and Tim Mansfield for discussions that helped to shape these ideas. Geraldine, Kate and Michael also participated with Simon in the field study mentioned in the paper. Since the material in this paper is probably highly contentious, it should
be pointed out that the author alone is responsible for its content.

REFERENCES