Clarity from Confusion: Using intended interactions to design information systems

Jeffrey V. Nickerson
Stevens Institute of Technology
jnickerson@stevens.edu

James E. Corter
Teachers College, Columbia University
jec34@columbia.edu

ABSTRACT

Two tools are described that help designers visualize the structure of a system in the requirements phase of a project. First, a matrix is constructed that represents the tendency of components to interact. The matrix is derived from sequence diagrams, which in turn are based on textual scenarios. This interaction matrix is transformed into a *structure plot* of the system, showing a graph of the essential connections between actors. Second, this same matrix is used to generate a *sequence plot*: a sequence diagram optimized for problem-solving. We illustrate the effectiveness of this approach, first with a simulation study, and later with a participant-based study of inference from diagrams. The results suggest that a similarity-based approach to information systems design can generate new testable tools. Pragmatically, the tools help novices and experts alike by automatically generating candidate system configurations in the form of structural diagrams, and by generating better sequence diagrams.

Keywords

Information systems design, insight, sequence diagrams, network scaling, visualization

INTRODUCTION

Designers seek insight while lost in a maze of problem requirements. Designers are never sure if turning the next corner will reveal the maze's center – or a dead end. Indeed, this uncertainty of progress is a characteristic of human creativity: we don't know if we are getting closer, or even if we will ever find what we seek, until the moment of inspiration (Metcalfe, 1986). Designers explore this frustrating maze by manipulating representations (Visser, 2006). Thus, insights happen in designers' interactions with visual representation (Suwa and Tversky, 1997; 2001). Can we improve design, and specifically the design of information systems, through the automated generation of diagrams that provoke insight?

Design is taught as a method, a sequence of activities. Information systems designers are expected to elicit requirements from users, then identify the main actors in the system, and finally configure the connections between actors. While there are a plethora of good techniques for accomplishing these tasks (e.g. Booch, Rumbaugh, and Jacobson, 1999; Carroll, 2000), the instructions are general, and short on advice for specific situations. Novices face a particularly daunting challenge: a system with just five components can be connected in over 1000 ways. Experts face challenges of their own: starting with ten components the potential configurations number in the trillions. Indeed, the lack of theory, methods, and tools in the design of software-intensive systems has been highlighted of late (Lee, 2000), leading to a call for the development of a design science focused on software (Hevner, March, Park and Ram, 2004). How can software design be improved?

As Schön pointed out (1983), expertise is often built through reflecting on past projects, abstracting concepts to be applied on future projects. Suwa and Tversky have shown that this reflection can take place on the spot, as designers view the sketches they just created (2001). Indeed, software design researchers have noted the need for tools to aid such reflection (Redmiles and Nakakoji, 2004).

Here we take a step toward building such tools for reflection, by developing two tools and testing their efficacy. Our approach to software design is based on methods for analyzing *similarity* relations, an idea with a long history in psychology. Psychologists have developed ways of eliciting similarity data, converting similarity into distances, and then plotting these distances in metric spaces (e.g., Shepard, 1962) and in graphs (Hutchinson, 1989). These techniques can be applied to the design of information systems in the following way.

In the initial stages of requirements, predicted interactions among system components or actors can be elicited. We use the frequency of these interactions as a form of similarity data, analogous to the way frequency of communication is used in social network analysis (e.g. Garton, Haythornthwaite, and Wellman, 1997). Specifically, the intended interactions of the system can be derived from sequence diagrams, forming an *interaction matrix*. This matrix can be used to create graphs of system components. The graphs will be generated using network scaling (Hutchinson, 1989) and other related techniques: this process we refer to as *structure plotting*. The interaction matrix can also be used to generate alternate representations of sequence diagrams based on a technique called seriation (Hubert, Arabie, and Meulman, 2001). We call this *sequence plotting*. We show through an experiment that the representations generated by this method improve design decisions.

This outcome serves as the contribution of the paper. In broad terms, it suggests that similarity-based methods can guide the creation of specific and testable information systems theories. Pragmatically, it introduce tools which may be useful to designers, both novice and expert.

We proceed by describing two examples of this proposed similarity-guided design process. First, we perform a study in which we derive interaction tendencies from sequence diagrams and use these to generate system configurations. Next, we describe a technique to optimize the representation of sequence diagrams. In a human-participant experiment, we show that the optimized representation helps designers solve problems.

STRUCTURE PLOTTING

Similarity can be measured through experiment. In psychophysics, we might measure the similarity of colors based on a human's ability to perceive differences, and compare this measure to another form of similarity based on the measurement of color spectra. Thus similarity can be derived from purely objective attributes of a physical system or can originate with data on human preferences.

In the case of information systems, measures of association or interaction are critically important in considering possible system designs, and such measures may be analyzed as a form of similarity. In information systems design, components of a system may be considered to be more "similar" the more they communicate with each other. To be clear, we are not saying the components themselves resemble each other, rather that because of their frequent communication they should be considered "near" to each other in some abstract space, the same way friends who don't resemble each other can be close by virtue of their communication. The motivating idea is that frequently interacting components should be clustered into tightly connected subsystems in order to modularize and thereby tame complexity. While there are many good reasons for creating such modular designs (Baldwin and Clark, 2000), the method for partitioning a system is, in practice, more art than science (Winograd, 1996). How then can we guide this art?

In most design environments, visual representation is the way for designers to communicate with each other and with themselves (Suwa and Tversky, 1997). Thus, diagrams may help us partition networks. In any communication situation, it is possible to totally connect the network, so that all components can talk to each other. This has many advantages: changing configurations of work can then be handled without reconfiguration of the network. However, by doing so we lose the benefits of modularity. For example, a local area network with too many nodes becomes difficult to repair if a node or small set of nodes begins to dominate the network traffic; also, attacks on security may be harder to defend against when all nodes are connected to all others. Anticipating such problems, designers break large networks into smaller ones, and create direct paths between closely connected nodes. At the application level, a similar logic applies: It is hard to find a bug in a system in which all modules talk to each other, but if software objects are combined into larger components, and their interactions are limited to their neighbors, bugs in one component can be more easily identified, isolated, and fixed. In both cases designers use hierarchical techniques – wiring in networks, and method/procedure calls in application software – to cluster frequently interacting objects into subsystems.

How do we acquire similarity measurements? In an existing system, we could measure similarity from the actual amount of data transmitted, or from the number of method invocations, the same way we measure similarity in social networks on the basis of the frequency of interpersonal communication (e. g. Garton et al., 1997; Granovetter, 1973). But in the early stages of the design of a new system, we can't obtain network traffic statistics or software performance metrics, because the system is still not built. All we possess are the requirements. That is, we only have descriptions of the anticipated system.

While there are many ways of eliciting requirements (e.g. Hansen, Berente, and Lyytinen, 2009; Zowghi and Coulin, 2005), two techniques from the Unified Modeling Language standard are popular at the current time (Booch et al., 1999; Fowler, 2003). *Scenarios* are collected early in the requirement phase of the project: these are succinct stories of the sequence of communication events between a set of actors (Fowler, 2003). These scenarios are then converted into *sequence diagrams*, in which a line on a diagram indicates a message between actors. While scenarios do not completely characterize the system,

they do represent the frequent and prototypical interactions of a system. We will call the measure of similarity derived from scenarios and sequence diagrams the *interaction tendency*. The measure represents the users' beliefs about the amount of interaction that will occur between components.

We suggest a simple method for computing the interaction tendency from a sequence diagram: count the number of messages depicted from an actor to another actor and treat this as a form of similarity measure. In cases where more precision is desired, users could be asked for their forecast of frequency of interaction between each pair of actors in the system.

Once all predicted interactions have been aggregated as an *interaction matrix,* how can we generate provisional designs? We can find an essential network: a set of links that, if any link were deleted, would no longer be consistent with the interaction matrix. This will uncover the skeletal structure of the system. We may later decide to embellish this skeleton with additional links to accomplish other goals (e.g. redundancy), but our aim is to automatically identify the skeletal structure as a first step to assist the information systems designer.

This problem of finding underlying structure has been studied in psychology. Indeed, well-known statistical techniques such as factor analysis and cluster analysis address this problem of finding structure in proximity data. Multidimensional scaling, less well known, is a way of finding a visualizable structure from dimensional data, and additive trees provide a topological view of features (Sattath and A. Tversky, 1977; Corter, 1998). Hutchinson (1989) suggested a way of finding essential network structures for asymmetric proximity data. Our method incorporates a technique for identifying essential network arcs taken from his network scaling procedure, called NETSCAL.

Hutchinson's paper shows how essential networks can be constructed algorithmically from similarity data. Similarities can be easily transformed into distance-like (but asymmetric) measures, which we will call distances. Sequence diagrams like those shown in Table 1 will often yield sparse distance matrices: in such cases we can predict that interaction is more likely to occur between neighbors of neighbor actors than between less directly connected actors. We can model this by computing the geodesic distances, the shortest paths between all nodes (cf. Floyd, 1962). This modeling step is not necessary if the interaction matrix has been thoroughly filled-in based on, for example, the measurement of observed communication between actors in a current system. Either way, starting from a distance matrix d, an edge belongs to an essential graph G if:

$$d[x,y] \leq \min\left(\max\left(d[x,z],d[z,y]\right)\right) \forall z \neq x \vee y \quad (1)$$

In other words, an edge between two nodes is added to the graph if it is smaller than any component of any other possible paths in the graph. There is parsimony in this construction because an edge is added only when it is clearly needed. This construction can be interpreted in the context of information systems: because the network of machines and software components is often changing, new paths of direct integration should be introduced only when the advantages of the new path are substantial.

The detailed reasoning for the minmax operation of Equation 1, and a proof that the graph G is essential, are provided in Hutchinson (1989). Once a network topology has been constructed using Equation 1, there are several options for displaying the result. Hutchinson used an informal approach based on multidimensional scaling (Shepard, 1969). Instead, because we are most interested in network topology, we display the results using a spring-electrical embedding algorithm, which emphasizes the structure of a graph (Hu, 2006). The graphs are displayed without directed edges.

Thus our proposed method is:

1. Elicit an interaction matrix s (or infer s by counting edges in sequence diagrams)
2. Convert the similarities in s to distances ($d[x,y] = \max(s) - s[x,y] + \varepsilon$, the epsilon to avoid zero distances)
3. (Optional) Convert the distances to geodesic distances using the all-pairs-shortest path algorithm (cf. Floyd, 1962)
4. Build the essential graph G by applying Equation 1 to the distances (cf. Hutchinson, 1989)
5. Display G as an undirected graph using a spring-electrical embedding

We call the overall approach *structure plotting*. We think that structure plots will provide useful suggestions to designers. That is, they will reveal unseen structures and suggest credible designs. We invite the reader to participate in a simple exercise in this regard.

Consider the four different sequence diagrams in Table 1. Imagine the components *A* through *E* are software modules that need to be integrated. Inspect the sequence diagrams for the interactions between these components, and discover the configuration of components that will satisfy the general design objectives of parsimony and clarity.

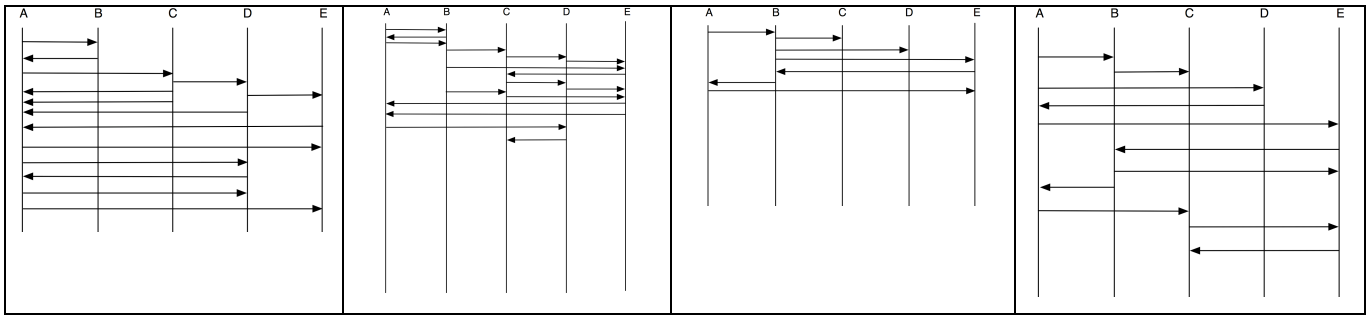


Table 1. Sequence diagrams for the exercise

The examples of Table 1 were chosen to make a point: Even simple sequence diagrams provide challenges to experienced designers. The sequence diagrams represent well the timing of events, but readers who performed the exercise will probably have discovered that sequence diagrams do not readily reveal the structure of interactions between components.

We applied the structure plotting approach to all four examples. In Table 2, the sequence diagrams, the original similarity matrices, the structure plots are shown. Notice that the structure plots correspond to common network patterns: complete graphs, rings, hub-and-spokes, and complete graphs with bridges (Peterson and Davie, 2003). This exercise suggests that structure plotting from the data embedded in a sequence diagram can reveal structure that is not apparent to the designer. Next, we consider how similarity can be used to improve the sequence diagrams themselves.

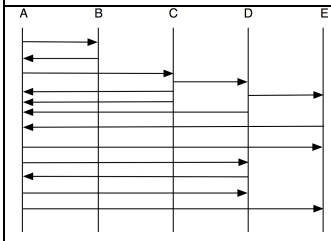
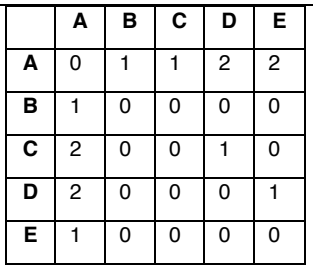
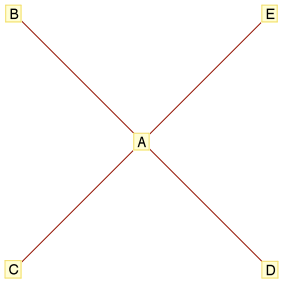
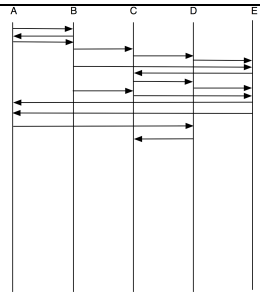
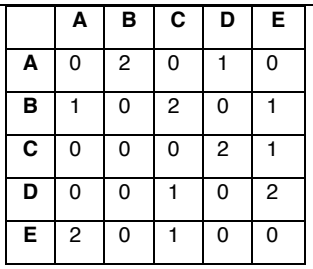
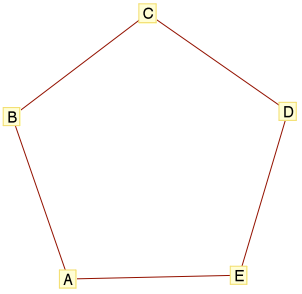
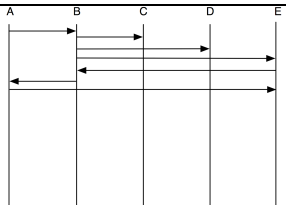
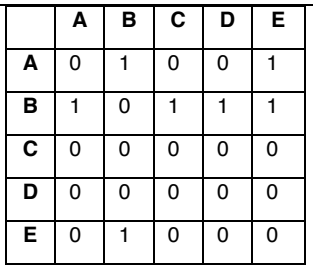
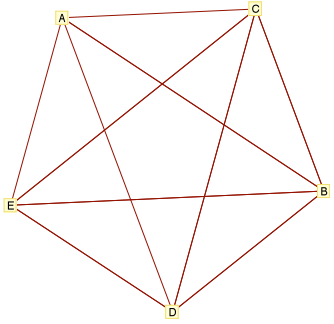
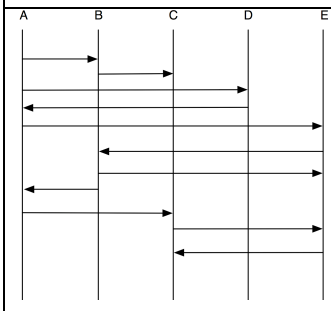
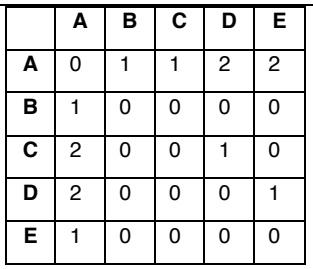
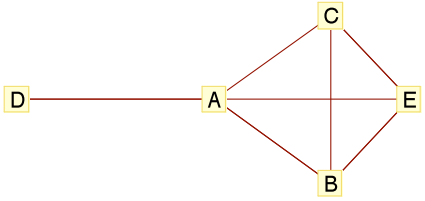
Sequence diagram	Interaction matrix derived from the sequence diagram	Structure calculated from the interaction matrix	Type
			Hub
			Ring
			Complete graph
			Complete graph with bridge

Table 2. Rows show the interaction matrix derived from the sequence diagrams of Table 1, and the result of structure plotting.

SEQUENCE PLOTTING

Problem

Similarity is useful in uncovering structures. It also is useful in refining representations. Here we show how sequence diagrams can be optimized with respect to the interaction matrix. In our previous work (Nickerson, Corter, Tversky, Zahner, and Rho, 2008), we found that designers typically generate a sequence diagram showing components from left to right in the order in which those components are mentioned in the scenario text. That is, designers proceed left to right, start to finish, in sketching out the messages that occur. But this is not always the clearest way of ordering components to represent the situation. If components are ordered inappropriately, with frequently-communicating actors far from each other, the sequence diagram becomes difficult to read. In order to calculate a rough gauge of the illegibility of a sequence diagram, we can sum the total lengths of the line segments. We predict that the longer the total length of the arcs, the more cognitive effort it takes to understand the diagram, leading to degraded performance on inference tasks.

Using the interaction matrix, we can compute the optimal linear ordering of the elements. Finding such an optimal sequence is referred to as seriation or ordination, and has been used in many fields, notably archaeology. In small examples, we can exhaustively compute the best solution by considering all permutations of the ordering of the actors, and counting the total length of the lines in each permutation. For larger examples, we can make use of standard seriation algorithms (Hubert, Arabie, and Meulman, 2001; Brusco and Stahl, 2005), which can take as input a similarity (or distance) matrix and return the optimal sequential ordering. Useful seriation techniques have been instantiated in the open source software package *R* (Hahsler, Hornik, and Buchta, 2008).

More formally, we define the preferred arrangement of a sequence diagram containing a set of actors *a* as

$$\arg\min_{p}\left(\sum_{x=1}^{|a|}\sum_{y=1}^{|a|}(|y-x|)s[x,y]\right) \tag{2}$$

where *p* ranges over all possible permutations of the set of actors, and the matrix *s* is the interaction matrix derived from a sequence diagram. In other words, we look for the permutation of the actors that minimizes the combined length of the lines that are drawn on the permutation's corresponding sequence diagram. We call this process *sequence plotting*.

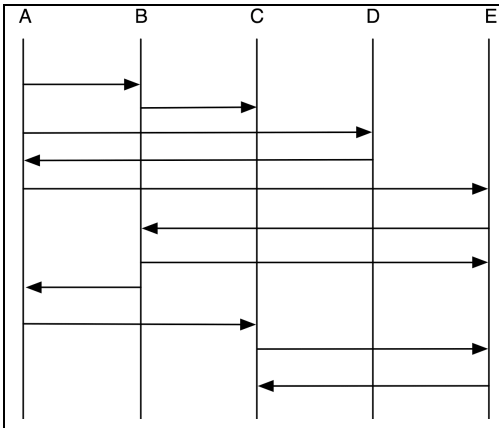


Figure 1. A sequence diagram, the last in Table 1.

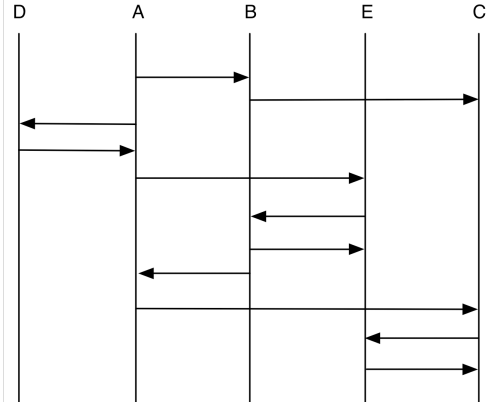


Figure 2. An equivalent sequence diagram, optimized to reduce the total line length.

The calculation inside the brackets of Equation 2 can be performed by just counting the number of units traversed in a sequence diagram. For example, in Figure 1, the first two horizontal lines are one unit long, and the third line is three units long. Continuing in this way, we find Figure 1 yields a count of 25, whereas Figure 2 yields a count of 15.

The authors believe that Figure 2 is easier to interpret with respect to how the actors are coupled to each other. This is consistent with studies of diagram cognition, in which crossing lines have been found to make diagrams harder to understand, presumably due to increased cognitive load (Purchase, Cohen and James 1997; Purchase, Colypoys, McGill, and Carrington 2002). We predict that optimizing sequence diagrams may help reveal to the designer the structure of a scenario, encouraging more accurate inferences.

We test this prediction with the following experiment.

Method

Users of an online community were asked to participate in a study in return for a modest stipend. Subjects were requested to participate only if they understood sequence diagrams, and, after completing the problem, they were asked to respond to several demographic questions, including the lines of code they had written, and their amount of experience in the design of information systems. Each subject was presented with either Figure 1 or Figure 2 at random, and given the following question:

> *Assuming that this sequence diagram represents the data traffic in a particular system, how would you partition the system?*
>
> *a) A and B connected to each other, and A fully connected to C, D, and E.*
> *b) A and C connected to each other, and A fully connected to B, D, and E.*
> *c) A and D connected to each other, and A fully connected to B, C, and E.*
> *d) B and C connected to each other, and B fully connected to A, D, and E.*

The correct answer is *C*: indeed, in the last row of Table 2, the structure map suggests a complete graph (as in a local area network or software bus) for *A, B, C*, and *E*, and a bridge connection from *A* to *D*. We predict that the correct answer *C* will be given more often in the group of subjects who received Figure 2.

Results

Fifty-three participants completed the study. These participants reported a mean age of 29.2, and 76% were male. 10% reported more than 5 years of experience in information systems design, 22% reported between 1 and 5 years, 37% reported less than one year of experience, and the rest reported no experience. 5% estimated that they had written more than 100K lines of code, 28% between 10K and 100K lines, 24% between 1K and 10K lines, 23% less than 1K lines, and 20% said that they had not written computer code. 30% had a graduate degree, 39% had a bachelor's degree, and 17% reported some college. 76% reported English as their primary language, with Hindi (6%), French (4%), and Tamil (3%) being the next most frequent.

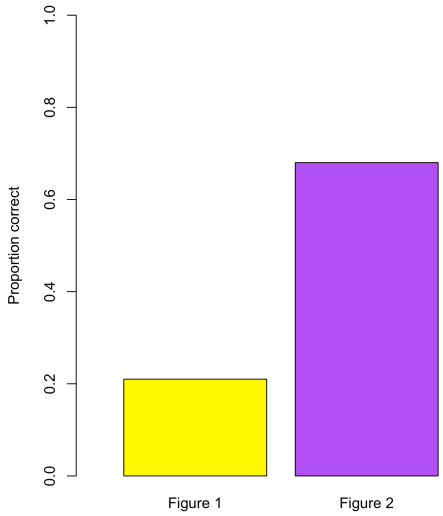


Figure 3. Proportion of correct inference for the figures seen.

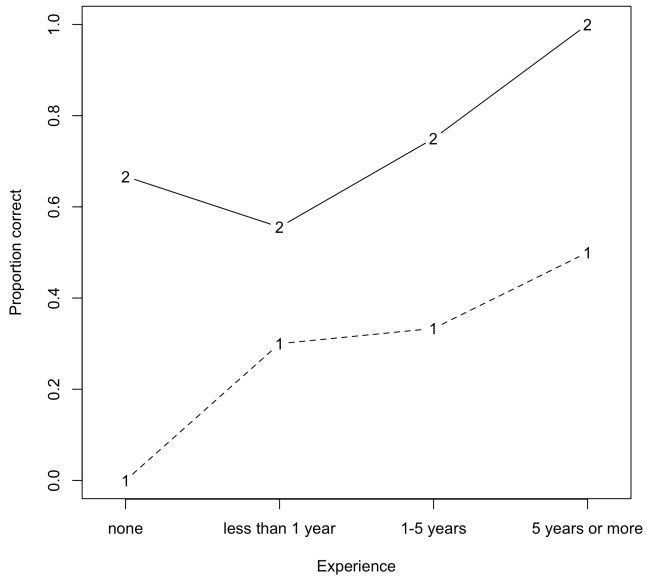


Figure 4. Proportion of correct inference as a function of figure seen (dotted line=Figure 1 and solid line=Figure 2) and self-reported work experience

Twenty-eight participants viewed Figure 1; of these, six (or 21%) correctly answered the design question. Of the 25 participants viewing the optimized Figure 2, 17 (68%) made the correct inference, as shown in Figure 3. This difference in success rate was significant by a chi-square test for homogeneity of two proportions, $\chi^2(1) = 11.662$, $p<.001$. Thus, the optimally arranged sequence diagram led to a higher rate of correct solution. To ensure that this effect was not due to differences in expertise between the two participant groups, we began by checking if the amount of previous design experience (rank-order coded) was related to performance in the inference task. Indeed, design experience was correlated .26 with choice of the correct design; this was marginally significant ($N=53$, $p<.10$). Level of experience was then used as a control variable in a logistic regression predicting correct inference from the figure viewed; in this model the figure seen still had a significant effect on inference (Wald = 10.427, $p<.01$), as did experience (Wald = 3.869, $p<.05$). Figure 4 shows the mean rate of correct solution for participants as a function of which figure they saw (optimized or non-optimized) and self-reported design experience. Figure 4 shows that on average the least experienced designers can't answer the questions correctly from the original diagram, but perform better with the optimized diagram. The most experienced designers answer correctly half the time with the original diagram, and perform perfectly with the optimized diagram. Thus, both novices and experts benefit from sequence plotting.

DISCUSSION

The results of this simple experiment demonstrate that well or poorly chosen arrangements of components in a sequence diagram can have strong effects on the ability of system designers to recognize the correct system topology. Only 21% of respondents found the correct topology when the sequence diagram was not optimized, while over two-thirds of respondents found the optional system topology when the sequence diagram was optimally ordered.

In previous research we have shown that non-optimal diagrams are often constructed by novice system designers and that even expert designers tend to generate sequence diagrams in the order of actor appearance in a textual description (Nickerson et al., 2008). Thus, the general inability of participants to understand a non-optimal diagram should be addressed. We demonstrated the remedy: sequence plotting provides a better diagram that aids understanding.

With respect to research on diagram understanding, our finding shows that an aspect of sequence diagrams that has been treated as unimportant or even trivial, the serial ordering of components, has a large influence on the quality of viewers' understanding.

While the sequence plotting experiment shows that humans do respond well to better diagrams, the study on structure plotting shows that automated tools may be able to not only produce better diagrams, but also suggest better structures.

We have been cautious not to make claims about the optimality of the networks generated by structure plotting: network optimization is often intractable. If enough realistic constraints are added to a problem, all possible graphs need to be evaluated (Garey and Johnson, 1979). But the designer often has to act regardless of the large number of possible configurations. The approaches presented here assist designers in visualizing alternatives, by generating a small set of candidate designs based on similarity measures. Comparing these automatically generated solutions is tractable. That is, designers can use tools to generate a small set of alternatives, and then run these alternatives against a richer model of payoffs and penalties.

CONCLUSION

The results of early requirements gathering can be used to estimate the extent to which actors will communicate with each other when a system is built. From these predicted interactions, the structure of a system – the essential graph – can be plotted. These interactions can also be used to improve sequence diagrams, which in turn will improve inferences about the design of the system. The improved diagrams aid novices and experts alike.

From a theoretical standpoint, this work suggests that just as psychology has found that the principle of similarity can guide the generation of theories of human cognition, so might information systems find that similarity can guide the generation of theories of human-computer and computer-computer interaction.

The aids presented here may be of particular interest to educators of designers. Novices are less likely to correctly chunk information than experts (cf. Larkin et al., 1980), and these aids may assist them. Experts have better intuitions about how to chunk data, but are more likely to confront large systems that defy the human capacity to recognize patterns, and so they also may find these aids useful. Finally, systems troubleshooters are often confronted by a system that is misbehaving, and have to discover the cause. In order to do so, they could make use of these tools to quickly understand the overall structure of an unfamiliar system.

More generally, this approach suggests that clarity can emerge as a result of the automated manipulation of representations. That is, insight may be the result of a reorganization of representation. Tools help us navigate the maze of design activity so that we can turn the next corner – and suddenly see the center.

ACKNOWLEDGMENT

The authors gratefully acknowledge the support of the National Science Foundation under award IIS-0725223.

REFERENCES

1. Baldwin, C. Y. and Clark, K. B. (2000) *Design rules*, MIT Press, Cambridge, MA.
2. Booch, R., Rumbaugh, J, and Jacobson, I. (1999) *The Unified Modeling Language User Guide*, Massachusetts: Addison Wesley.
3. Carroll, J. M. (2000) Making Use: Scenario-Based Design of Human Computer Interactions, MIT Press, Cambridge, MA.
4. Corter, J. E. (1998). An efficient metric combinatorial algorithm for fitting additive trees, *Multivariate Behavioral Research*, 33, 2, 249-272.
5. Floyd, R. W. (1962) Algorithm 97: Shortest Path, *Communications of the ACM* 5, 6 pp. 345.
6. Fowler, M. (2003) UML distilled: A Brief Guide to the Standard Object Modeling Language, Addison-Wesley, MA.
7. Garey, M. R., and Johnson, D. S. (1979). *Computers and Intractability*. Freeman, New York.
8. Garton, L., Haythornthwaite, C., and Wellman, B. (1997) Studying online Social Networks, *Journal of Computer-Mediated Communication*, 3, 1
9. Granovetter, M. (1973) The Strength of Weak Ties, *American Journal of Sociology*, 78, 6, pp. 1360-1380.
10. Hansen, S., Berente, N., and Lyytinen, K. (2009). Requirements in the 21st Century: Current Practice and Emerging Trends. In Lyytinen, K., Loucopoulos, P., Mylopoulos, J., and Robinson, W. *Design Requirements Engineering: A Multi-disciplinary Perspective for the Next Decade*. Springer-Verlag, Berlin.

11. Hevner, A.R., March, S.T., Park, J., and Ram, S. (2004) "Design Science in Information Research", *MIS Quarterly* 28, pp. 75-105.
12. Hu, Y. F., (2006) "Efficient, High-Quality Force-Directed Graph Drawing," *The Mathematica Journal*, 10, 1, pp.37-71
13. Hutchinson, J. (1989), Netscal: A network scaling algorithm for nonsymmetric proximity data. *Psychometrika*, 1989, 54, 1, pp. 25-51.
14. Larkin, J.H., McDermott, J., Simon, D.P, and Simon, H.A. (1980) "Expert and Novice Performance in Solving Physics Problems", *Science* (208), pp. 1335-1342.
15. Lee, A. (2000) Systems Thinking, Design Science and Paradigms: Heeding Three Lessons from the Past to Resolve Three Dilemmas in the Present to Direct a Trajectory for Future Research in the Information Systems Field. *11th International Conference on Information Management*, http://www.people.vcu.edu/~aslee/ICIM-keynote-2000
16. Metcalfe, J. (1986) Premonitions of insight predict impending error, *Journal of Experimental Psychology: Learning, Memory, and Cognition,* 12, 623-634.
17. Nickerson, J. V., Corter, J. E., Tversky, B., Zahner, D., and Rho, Y. (2008). The Spatial Nature of Thought: Understanding information systems design through diagrams, in Boland, R., Limayem, M., and Pentland B., (eds), *Proceedings of the 29th International Conference on Information Systems*.
18. Peterson, L. L. and Davie, B. S., (2003) *Computer Networks: A Systems Approach,* 3rd edition, Morgan Kaufmann, Boston.
19. Purchase, H. C., Cohen, R. F., and James, M. I., (1997) An Experimental Study of the Basis for Graph Drawing Algorithms, *Journal of Experimental Algorithmics*, 2, 4, pp. 1-17.
20. Purchase, H. C., Colypoys, L., McGill, M., and Carrington, D. (2002) UML Collaboration Diagram Syntax: An Empirical Study of Comprehension*, Proceedings of the First International Workshop on Visualizing Software for Understanding Analysis*, 2002.
21. Redmiles, D., and Nakakoji, K. (2004) Supporting Reflective Practitioners, *Proceedings of the 26th International Conference on Software Engineering (ICSE'04)* pp. 688 – 690.
22. Sattath, S., and Tversky, A. (1977). Additive similarity trees. *Psychometrika*, 42, 319-345.
23. Schön, D.A. (1983) The Reflective Practitioner: How Professionals Think in Action, Basic Books, Jackson, TN.
24. Shepard, R. N. (1962). The analysis of proximities: Multidimensional scaling with an unknown distance function, *Psychometrika*, 27, 125-140.
25. Suwa, M., and Tversky, B. (1997) What Do Architects and Students Perceive in Their Design Sketches? A Protocol Analysis, *Design Studies,s* 18, pp. 385-403.
26. Suwa, M., and Tversky, B. (2001) Constructive Perception in Design, *Computational and Cognitive Models of Creative Design V*, Gero, J.S., and Maher, M.L. (Eds.), University of Sydney, Australia, pp. 227-239.
27. Visser, W. (2006) *The Cognitive Artifacts of Designing,* Lawrence Erlbaum, Mahwah, NJ.
28. Winograd, T. (1996) *Bringing Design to Software,* Addison-Wesley, Boston, MA.
29. Zowghi, D., and Coulin, C. (2005). Requirements Elicitation: A Survey of Techniques, Approaches, and Tools. In Aurum, A., and Wohlin, C. (Eds.) *Engineering and Managing Software Requirements*, Springer-Verlag, Berlin.