Software Engineering and Scale-Free Networks

Author

Wen, Lian, Dromey, R Geoff, Kirk, Diana

Published

2009

Journal Title

I E E E Transactions on Systems, Man and Cybernetics, Part B: Cybernetics

DOI

10.1109/TSMCB.2008.2008102

Software Engineering and Scale-Free Networks

Lian Wen, R. Geoff Dromey, *Member, IEEE*, and Diana Kirk

***Abstract*—Complex-network theory is a new approach in studying different types of large systems in both the physical and the abstract worlds. In this paper, we have studied two kinds of network from software engineering: the component dependence network and the sorting comparison network (SCN). It is found that they both show the same scale-free property under certain conditions as complex networks in other fields. These results suggest that complex-network theory can be a useful approach to the study of software systems. The special properties of SCNs provide a more repeatable and deterministic way to study the evolution and optimization of complex networks. They also suggest that the closer a sorting algorithm is to the theoretical optimal limit, the more its SCN is like a scale-free network. This may also indicate that, to store and retrieve information efficiently, a concept network might need to be scale-free.**

***Index Terms*—Concept network (CN), scale-free network, software engineering, sorting algorithm.**

I. Introduction

NEARLY ALL complex systems, which exist in both the abstract world (AW) and the physical world (PW) as described in the information-matter-energy model [28], can be abstracted as a complex network, which connects a number of subsystems and components using different types of relations [4]. Examining the structure and evolution of the underlying network provides us with a useful way of studying the system [33]. In recent years, researchers have investigated complex systems from different domains as *complex networks* [2], [32]. Networks studied include both those occurring in PW, for example, neuron activity, cellular metabolisms, protein folding, and electricity supply networks, and those in AW such as social networks and the web hyperlink network. Even though they represent totally different types of systems, they share many common properties such as power-law degree distribution [2], small-world property [33], and a high level of clustering [4], [5], which are classified in a new network model, the scale-free network [5]. The similarity between different complex networks inspires the idea that network growth mechanisms represent some basic natural tendency to create order from chaos.

Manuscript received November 30, 2007; revised March 25, 2008. This work was supported by the Australian Research Council (ARC) Centre for Complex Systems. This paper was recommended by Associate Editor Y. Wang.

L. Wen and R. G. Dromey are with the Software Quality Institute, Griffith University, Brisbane, Qld. 4111, Australia (e-mail: l.wen@griffith.edu.au; larrywen@hotmail.com; g.dromey@griffith.edu.au).

D. Kirk is with the Computing and Mathematical Sciences, Auckland University of Technology, Auckland, 1142, New Zealand (e-mail: dianakirk@acm.org).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMCB.2008.2008102

The recent progress in complex-network theory motivates us to study software systems as complex networks. A software system can be treated as a network of components connected by dependence relationships [34]. The importance of relationships are stressed by the object-attribute-relation model [29] in cognitive informatics (CI) [31], which shows that human memory and knowledge are represented by connections of synapses between neurons, rather than by the neurons themselves [31]. Network models are also addressed in concept algebra [30] that describes a general-knowledge framework as a dynamic and evolvable concept network (CN). Similarly, for software systems, some high-level functions are realized by the relationships of low-level components. In this paper, the term component indicates an abstract form of software or even hardware entities as in the behavior-tree approach [11]; similarly, the relationship between two components is abstracted as a dependence relationship. We call this network a component dependence network (CDN), and it represents a view of the architecture of a software system.

In our previous research [35], we have studied the degree distributions of the CDNs of several Java libraries and applications. We have found that they all show evidence of a power-law distribution. A similar result has been shown for applications written in C++ and Smalltalk [7]. This result also provides some evidence to support our initial conjecture that the evolution of software systems is like that of other kinds of complex systems. Not only does this result inspire some philosophical consideration of software systems but it also has some practical value. For example, it provides a measure to identify the most important components in large and complex software systems [35] based on a webmining technique [16].

In this paper, we have also studied another type of network, the sorting comparison network (SCN). Sorting algorithms has been one of the most interesting research topics since the beginning of computer science, with more than 100 different sorting algorithms invented to date [17]. Sorting is important both from a theoretical perspective and for the study of information-intensive applications such as databases [36] and CI [31]. For a comparison-based sorting algorithm, if we treat each record as a node and each comparison between two records as a directed edge between the two nodes (from smaller to larger record), the sorting process will generate a direct graph or a network, which is called the SCN. A major contribution of this paper is that, after comparing the SCNs of five different sorting algorithms, we have discovered that the closer the comparison number matches a theoretical lower bound of $n\log(n)$, the closer the SCN matches a scale-free network. This indicates that there is a possible connection between the optimization of sorting algorithms and the evolution of a scale-free network, i.e., a scale-free network may represent some kind of optimization. As

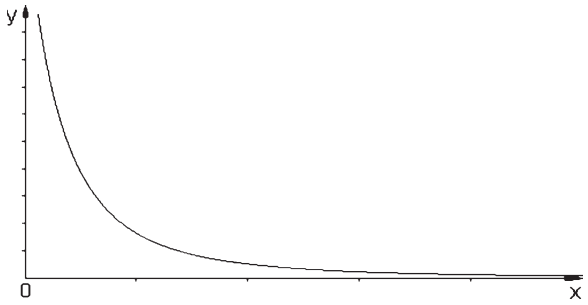


Fig. 1. Power-law distribution $y \propto x^{-\gamma}$; x is the number of connections, y is the number of nodes, and γ is a constant.

for an SCN, which rearranges abstract records so required information can be retrieved more efficiently, a CN [30] organizes different concepts into a relation network to form a complex knowledge system. We conjecture that, for a highly optimized CN, the topological structure may also be scale-free.

Most real-life large-scale networks, such as a human social network, the World Wide Web, or power-supply networks, take months, years, or even decades to evolve. It is very hard to trace all the details of their evolution history. Furthermore, due to uncontrollable factors that may affect the evolution of these networks, they sometimes appear to grow stochastically. These aspects make it difficult to use such networks as a basis for studying the evolution of scale-free networks. However, for an SCN, once the sorting algorithm and the input sequence are given, the evolution of the SCN is determined. Therefore, the SCN provides a much more repeatable and precise approach to the study of scale-free networks.

We have developed tools to test both CDN and SCN; these tools can be freely downloaded from the Internet [8], [23] and used to replicate all the results shown in this paper.

The organization of this paper is as follows. In Section II, we briefly introduce complex-network theory and scale-free networks. In Section III, we present some evidence to support the claim that CDNs are scale-free and some practical significance for this claim. In Section IV, we introduce the SCN and show the testing results, which specify the relationship between optimized sorting algorithms and scale-free networks. Some discussion is provided in Section V, and finally, a brief conclusion is presented.

II. Scale-Free Networks

For many years, networks have been treated mainly as random networks [4]. In random networks, edges linking vertices are assumed to be randomly placed, and the number of edges attached to each vertex (the degree) has been shown to follow a normal distribution.

However, the traditional model (random networks) cannot be used to explain many real large complex networks such as human social networks. Therefore, a new network model, called a scale-free network model [5], has been proposed.

The scale-free network model was introduced in the 1990s [4]. The most important feature of a scale-free network is the power-law degree distribution (see Fig. 1) on contra to the normal distribution of a random network. The direct effect of a power-law distribution is that there are a few nodes, called hubs, with a much larger number of connections as compared with the average; it seems that there is no upper limit of the possible links a hub may have as the size of the network increases and that is the reason why the property is called scale-free [5]. A scale-free network usually exhibits the following properties [4].

1) **Power-law degree distribution**: The degree distribution has a long decreasing tail, which follows the power law.
2) **Hubs**: A few nodes have a much larger number of links than most other nodes.
3) **Small world**: The average distance between nodes in a scale-free network is very small as compared to the number of nodes in the network.
4) **Clustering**: The clustering coefficient[1] of some scale-free networks is found to be much larger than that of a random network with the same number of nodes and links.
5) **Efficiency of spread**: In a scale-free network, via the hubs, information can spread through a network much more efficiently than through a random network [4].
6) **High error tolerance**: A scale-free network can tolerate a much higher error rate than a random network, if the errors happen in randomly selected nodes or links.
7) **Vulnerable to well-organized attacks**: Scale-free networks are highly tolerant to random errors but may have weak points: the hubs.

The significance of the scale-free network model is that many real-world complex networks are discovered to comply with this model. These networks include cellular metabolisms [13], chemical-reaction networks, protein regulatory networks [14], research-collaboration networks, social networks, and the World Wide Web [5]. The fact that so many different kinds of systems show strikingly similar topological properties has spawned interest across many disciplines and has caused researchers to postulate the existence of some basic organizational principle. A breakthrough in the study of complex networks may result in significant progress in the research of many different fields. In this paper, we suggest that the scale-free network property also exists in software systems. This implies that the research results relating to complex networks in other disciplines can also be relevant to software engineering.

III. CDNs

A software system can be treated as a network of components connected by certain dependence relationships. Here, the concept of a component is a high-level functional abstraction. In Java and other object-oriented systems, we treat a public class or a public interface as a component. In this section, we have tested the CDN of eight Java libraries and applications. All of them show scale-free properties. Combined with other work on C++ and Smalltalk [7], these results support our initial conjecture that a CDN is usually a scale-free network.

[1]In a network, for node i, let k_i be the number of its connected nodes, n_i be the link number between those nodes, then $C_i = 2\ n_i/k_i(k_i - 1)$. The clustering coefficient is the average of C_i over all nodes.

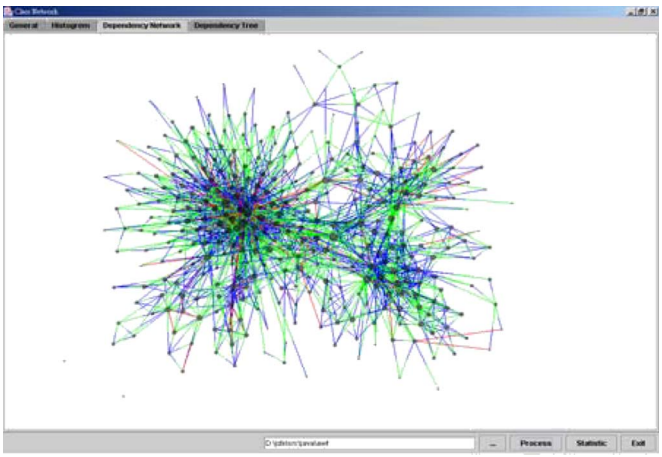


Fig. 2. CDN of Java package java.awt.

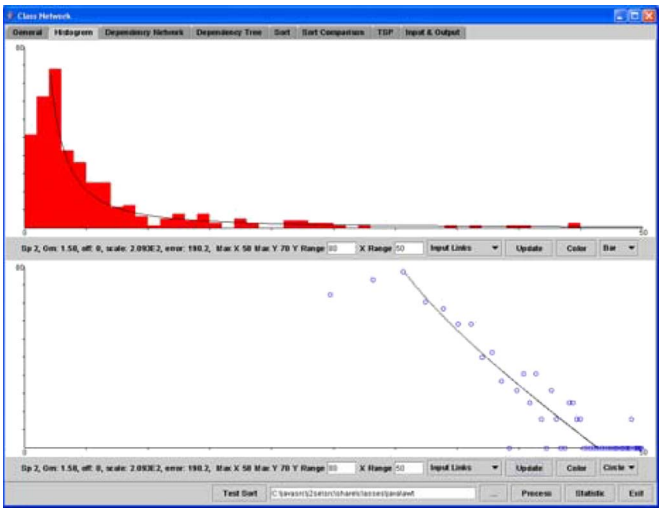


Fig. 3. Degree distribution of incoming links of package java.awt.

A. Test Results for Java Applications

We have developed a tool, which can be freely downloaded from the Internet [8], to retrieve and study the topological structure of CDNs of Java applications. Eight Java packages, including five of Sun's official Java libraries, an open-source project apache ant [3], and two small Java projects written by one of the authors of this paper, have been tested. All of them show positively the characteristics of scale-free networks. A detailed report of the tests can be found in [35]. In this section, we only highlight some of our test results.

The CDN of the Java package "java.awt" is shown in Fig. 2, where a node represents a public class or interface in the package and an edge represents a dependence relationship between two nodes. The positions of nodes in the CDN are calculated using a force-directed algorithm [9]. The size of a node is determined by the number of connections on that node.

Fig. 3 shows the incoming links degree distribution of the CDN of "java.awt". The diagram is shown in linear and logarithmic scales. The x-axis represents the number of incoming links on a node, and the y-axis represents the number of nodes. The curved line shows the function of the power-law distribution $y = A \times x^{-\gamma}$, where A and γ are constants for a given distribution.

The highlights of the test are as follows.

1) The standard deviation of the degree distribution is about three times the size of the average degree. Contrast this to a normal distribution where the standard deviation is equal to the mean.
2) The clustering coefficient is about 20 times larger than p (p is the average degree over the size of the network), whereas in a random network, the clustering coefficient is approximately equal to p [2].
3) All the degree distributions, including incoming, outgoing, and total, show long tails that follow a power law.

Because similar results were obtained from all eight packages tested, we conjecture that the CDN for most Java systems, independent of the functionality of the systems, are scale-free networks.

B. Identify Important Classes

Even though this research marks the early stages of the study of a software system as a complex network, the knowledge that a CDN is a scale-free network may have some practical value. We propose that this knowledge may help people to identify important components (classes) for legacy software systems.

Reverse engineering and software maintenance requires engineers to study the properties of code in an attempt to identify the key classes [37], the classification of subsystems [18], [26], and the location of features [12].

The CDN of a large software system may include hundreds or thousands of nodes, but as a scale-free network, it has only a small number of highly connected nodes; we propose that those nodes are more likely to be important components (classes).

A CDN is a directional network, so it can be useful to separate the incoming connections from the outgoing connections. A webmining technique [16] has been applied to obtain a more sophisticated measure. The idea of the webmining technique is not only to count the number of connections but also to evaluate the quality of the connections. In a network, each node has two associated values: the weight of hub and the weight of authority. The details of how to calculate these values can be found in [16]. A class with a high weight of authority means that it could be an important class because it is referred to by many other classes with a high weight of hub. A class with a high weight of hub means that many classes with a high weight of authority have been referred to by it. A recursive algorithm is applied to obtain a stable value of the weight of authority and the weight of hub.

Through our testing, we have discovered that, in the package of "Java", which includes more than 1000 classes and interfaces, some of most frequently used classes such as "String", "Object", "IOEception", and "System" are in the list of top five of the highest weight-of-authority classes, while "Component", "Toolkit", "Window", "Container", and "Font" are in the list of top ten of the highest weigh-of-hub classes. In the package of "java.awt", which includes 345 classes and interfaces, "Component", "Toolkit", "Point", "Rectangle", and "Event" are in the

list of top ten of the highest weight-of-authority classes, while "Component", "Toolkit", "Container", and "Window" are in the list of top five of the highest weight-of-hub classes. The results support the idea that the scale-free property of a CDN helps to identify some important classes and components (more details of the testing result can be found in [35]).

In this section, a static approach has been proposed to identify important components (classes) in a software system. This approach is based on the assumption that, for a software system, the CDN is usually a scale-free network and the degree distribution follows a power-law distribution. This feature implies that a small number of components have a much higher number of connections and, therefore, have a high weight of authority or a high weight of hub. This approach is also based on the assumption that important components do usually have a high weight of authority or a high weight of hub. The first assumption has been supported by testing results shown in Section III-A, and the second assumption has been validated by a test on package "java" and "java.awt" in this section.

The static approach for identifying important components in software systems is only one example that demonstrates the practical usage of the complex-network theory in software engineering. Some other possible practical benefits are suggested in the conclusion section of this paper.

C. Distributed Software Systems

From our previous study [35] and independent research by others [7], we argue that the CDNs of software systems are likely to be scale-free. However, all the tests are based on a software system running in one location. So far, there is no test on distributed software systems. In this paper, we will provide only some thoughts about this issue.

The overall detailed topological structure of distributed software systems is more difficult to analyze because software running in distributed locations could be written in different software languages, running on different platforms, and designed based on different architecture styles [24]. However, we believe that similar rules operating for the evolution of other large complex networks could also apply here. As we know, hierarchy is a universal structure for building large systems, that means similar structure could be applied on different hierarchical levels.

Let us consider social networks. A social network for a small group of people (e.g., a small club) is usually a complete network, which means that there is a connection between any two members in the group. However, as the number of people increases, a social network of a national or a global level is a scale-free network [4]. For large-scale social networks, if we treat a group of people as one node (for example, a country as one node, a relationship between two countries is a link), the large network is simplified to a network with a much smaller number of nodes, and it could become a complete network again.

Study of social networks suggests that a network's structure is related to its size. For a small network, it is likely to be a complete network. However, when the number of nodes increases, to reduce the complexity, the network tends to be scale-free. To explain this phenomenon, consider the two extreme models of networks. One is a tree, which has the minimum connections $(O(n))$, and the other is a complete network, which has the maximum connections $(O(n^2))$. The advantage of a tree is its simplicity. At the same time, it is fragile because only one broken connection is needed to separate a tree into two parts. Therefore, redundant connections are necessary to strengthen a network. When the network is small, the difference between $O(n^2)$ and $O(n)$ is not significant, so it is affordable to maintain a complete network as it provides the maximum flexibility and strength. However, when the size of a network increases, the cost to maintain a complete network increases much faster. A tradeoff is to have a network model in the middle of a tree and a complete network. A good candidate is a scale-free network with probably $O(n\log(n))$ connections.

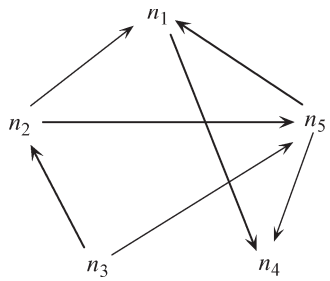


Fig. 4. Example of an SCN for a sequence of five integers.

If the evolution of the CDN of a distributed software system follows the laws that control the evolution of a social network, we may expect that the CDN of a distributed network is scale-free as the system is large. If we treat software at one distributed location as one node and the total number of distributed locations is small, then the dependence network of the distributed locations could be a complete network.

IV. SCNs

A. Definitions

For a comparison-based sorting algorithm, the sorting process is required to compare the key values of the records. Let us define each record as a node and each comparison between two records as a directed connection between the two corresponding nodes (the direction is determined by comparison result). Then, the sorting process will weave a network that is called an SCN.

For example, consider a sequence of five distinct integers $n_1n_2n_3n_4n_5$ as input data and a sorting algorithm that executes the following comparisons: $n_1 < n_2$, $n_1 < n_5$, $n_2 < n_3$, $n_4 < n_1$, $n_4 < n_5$, $n_5 < n_2$, and $n_5 < n_3$. The corresponding SCN is shown in Fig. 4.

In Fig. 4, there is a unique path $n_3 \rightarrow n_2 \rightarrow n_5 \rightarrow n_1 \rightarrow n_4$ that travels in the same direction and visits each node once. This path indicates the sorted sequence $n_3 > n_2 > n_5 > n_1 > n_4$.

For a comparison-based sorting algorithm, the theoretical lower bound for the number of comparisons in the worst case is $\lceil \log(n!) \rceil$ [17], which approximately equals $n\log(n)$, where n is the number of records in the input sequence. Different sorting algorithm may result in different average numbers of

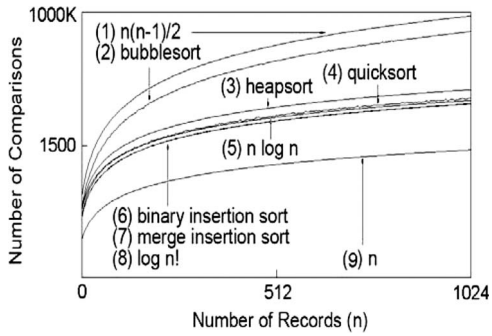


Fig. 5. Number of comparisons required by the five different sorting algorithms and four reference curves. Note: Curves are listed in decreasing order. Curves (6), (7), and (8) are too close to be distinguishable.

comparisons. In this paper, a sorting algorithm is defined as better than another algorithm if the first one requires less average number of comparisons. A sorting algorithm is *optimized* if the average number of comparisons is close to the theoretical lower bound.

B. Comparisons

We investigated the SCNs of five different sorting algorithms: Bubblesort, Heapsort, Quicksort, Binary Insertion Sort, and Merge Insertion Sort (details of these sorting algorithms can be found, e.g., in [17]). Fig. 5 shows the number of comparisons required by the five different sorting methods, which is equivalent to the number of links in the respective SCN. These numbers were obtained experimentally by averaging, for each algorithm, the number of comparisons required for ten different random input sequences. In Fig. 5, the x-axis is the number of records, and the y-axis is the number of comparisons, which is shown in logarithmic scale. Fig. 5 also includes four reference curves for comparison: n, $\log(n!)$, $n\log(n)$, and $n(n-1)/2$.

The curves in Fig. 5 confirm what one expects from the known theory of sorting methods found in [17]. The number of comparison in a Binary Insertion Sort and Merge Insertion Sort matches the theoretical optimum $\log(n!)$. Quicksort and Heapsort are close to $n\log(n)$, and Bubblesort requires many more comparisons $(O(n^2))$. Note that, in this paper, we are counting only the number of comparisons and ignoring other operations such as data movement. Otherwise, Quicksort and Heapsort would rank better than Binary Insertion Sort and Merge Insertion Sort and are preferred in practice.

Fig. 6 shows example SCNs of the five sorting algorithms; the size of a node is determined by the number of connections on that node. The length of the input sequence is 128. The graphs are layered by a force-directed algorithm [9]. Fig. 6 shows that SCNs of different sorting algorithms have very different structures. The SCN for Bubblesort is a very dense graph, with essentially a single cluster. The SCNs for Binary Insertion Sort and Quicksort show a distinct set of clusters. The SCNs for Merge Insertion Sort and Heapsort show a structure with one dense core and sparse outer layers.

In order to evaluate how close the SCNs of the sorting algorithms match the structure of scale-free networks, we may study the degree distribution of the SCNs.

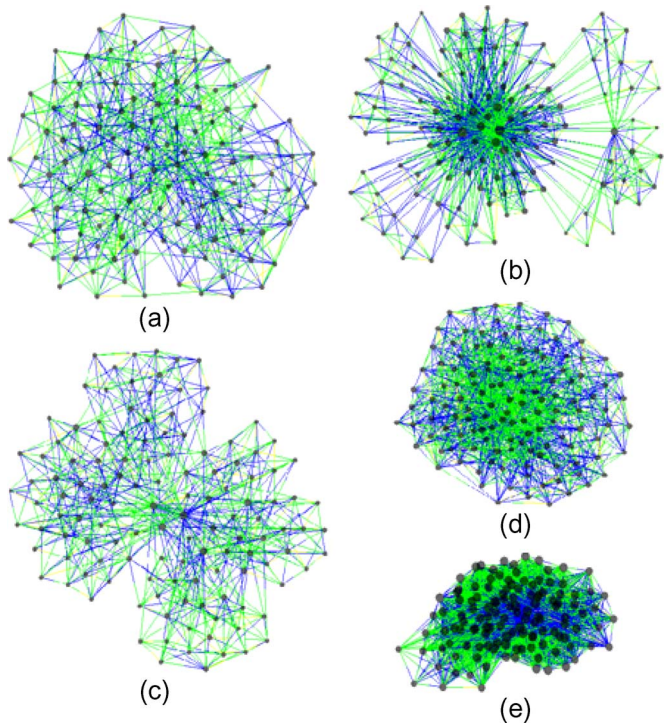


Fig. 6. SCNs of the five different sorting algorithms.

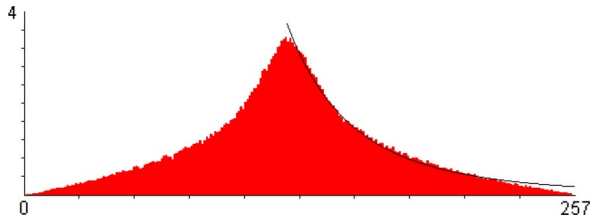


Fig. 7. Degree distribution of SCNs from Bubblesort.

Figs. 7–9 show the degree distribution of the SCNs for the five sorting algorithms. Each diagram was obtained by averaging 1000 executions of the respective sort on independent random sequences with the length of 256. The x-axis of each feature shows the range of the degree found in the SCN, and the y-axis indicates for each degree value the number of nodes in the SCN with that degree. The black curves indicated show an attempted approximation of the power-law distribution by a function $P(k) = ck^{-\gamma}$.

Among the five sorting algorithms, Bubblesort has the largest number of comparisons. The corresponding degree distribution (in Fig. 7) has an obvious bell shape that demonstrates the character of a normal distribution, which is found in a random network. Quicksort and Heapsort perform better than Bubble sort. Their corresponding SCN degree distribution (in Fig. 8) still have a bell shape, but the shape has been significantly shifted leftwards, so they have a long tail that matches the power-law distribution. The two degree distributions from Quicksort and Heapsort are very similar with the only obvious difference is that the tail of Quicksort is much longer than that of Heapsort.

The best sorting algorithms in regard to the number of comparisons in the five algorithms are Merge Insertion Sort and Binary Insertion Sort. The average numbers of comparisons of

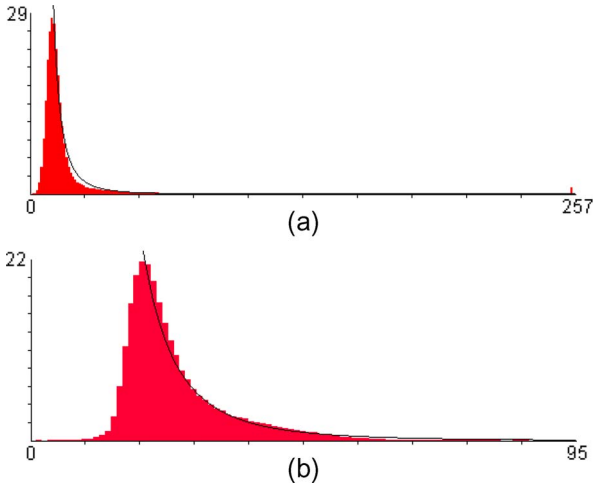


Fig. 8. Degree distribution of SCNs from Quicksort and Heapsort.

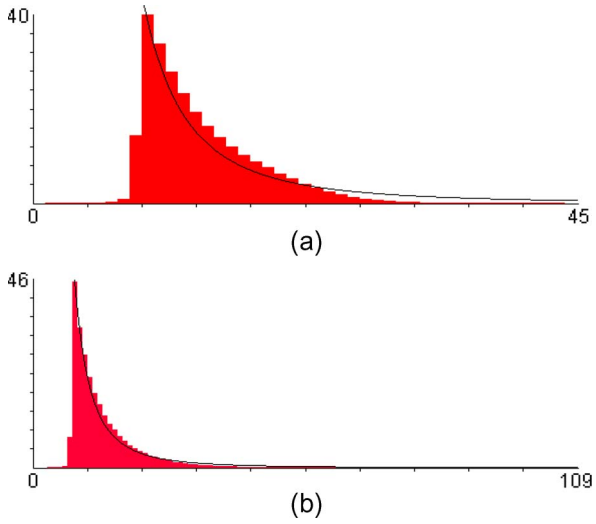


Fig. 9. Degree distribution of SCNs from Merge Insertion Sorting and Binary Insertion Sorting Algorithms.

both these algorithms are very close to the theoretical lower bound, and the corresponding degree distributions (shown in Fig. 9) are perfect scale-free networks.

The experimental results from Figs. 7–9, in conjunction with Fig. 5, support our conjecture that the closer the number of comparisons in a sorting algorithm is to the lower bound $\log(n!)$, the closer the respective SCN is to a scale-free network.

The test results in this section imply that the scale-free network structure provides an indication that the corresponding sorting algorithm is an optimal one in relation to the number of comparisons. This raises interesting questions about the possibility that scale-free networks in other disciplines also represent some kind of optimization.

C. Formalization of the Observations

In the previous section, we have compared the degree distribution of the SCNs of five different sorting algorithms. Based on visual judgement, we conclude that the SCN of a more optimized sorting algorithm is more like a scale-free network. In this section, we will compare some mathematical attributes of the degree distributions to formalize this observation. The results are listed in Table I. It includes SCNs of the five sorting algorithms and also the CDNs of three Java packages.

TABLE I
STATISTIC ATTRIBUTES OF DIFFERENT NETWORKS

	n	l	Rt	Tnr	Tlr	Twr
Bubble sort	256	15956	63.3	1.03	1.08	1.10
Heap-sort	256	3140	1.53	2.90	3.83	10.6
Quick-sort	256	2126	1.04	1.94	27.2	10.1
Binary insertion	256	1798	0.89	26.1	17.2	166
Merge insertion	256	1692	0.83	12.7	4.25	59.6
java.awt	345	1724	0.60	6.97	35.3	49.3
java	1077	8819	0.81	5.61	157	55.0
ant	743	3682	0.52	2.77	113	15.2

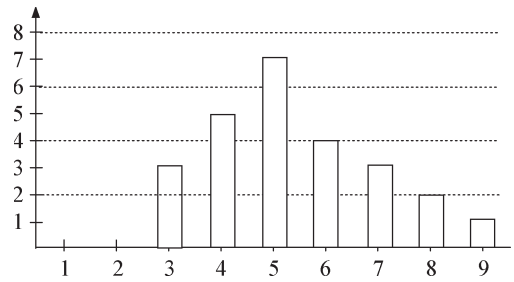


Fig. 10. Simple example distribution.

For a scale-free network, the interesting part is the long tail. Here, we define the middle point as the point where the distribution reaches its maximum value; the distribution after the middle point is called the tail; the distribution before the middle point is called the head. In Table I, "n" is the number of nodes in the network; "l" is the total number of links; "Rt" is the link number over $n\log(n)(Rt = l/n\log_2(n))$; "$Tnr$" (tail–number ratio) is the ratio of node number in the tail over the node number in the head; "Tlr" (tail-length ratio) is the ratio of the length of tail over the length of the head; "Twr" (tail weighted ratio) is ratio of the weighted length. To help to understand these concepts, consider the simple example distribution shown in Fig. 10.

In Fig. 10, the middle point is five, and the other parameters are calculated as follows:

$$Tnr = (4+3+2+1)/(3+5) = 1.25$$
$$Tlr = (9-5)/(5-3) = 2$$
$$Twr = (4\times 1+3\times 2+2\times 3+1\times 4)/(5\times 1+3\times 2) = 1.82.$$

Fig. 11. Typical railway network and a typical airline network.

Because a normal distribution is symmetric, it is obvious that all three tail ratios should be one for a restrict normal distribution. For a scale-free network, due to its long tail, we expect that the tail ratios should be larger than one. From Table I, we can observe the following phenomena.

1) The tail ratios for Bubblesort are close to one.
2) As an index of the scale-free property, the tail weighted ratio (Twr) is more consistent to the visual judgment.
3) When the Twr is about 50, the corresponding network has a very good scale-free property.
4) For the CDNs, the Twrs for both "java.awt" and "java" package are close to 50, but the Twr for package "ant" is much lower. A possible reason is that, as a fundamental Java library, package "java" and package "java.awt" are better maintained in the structure than normal applications software.
5) Rt is in the range of 0.52–1.53 for all the tested networks which are scale-free. It matches our expectation that the number of connections of a scale-free network is $O(n\log(n))$.

V. Discussion

A. Optimized Architecture

Without a clearly defined criterion, there is no possible way to discuss the optimization of a system. A software system, just like any other kind of organized system, includes two opposite elements: freedom and restriction or, alternatively, chaos and order. For a dependence network, there are two extreme situations: One is with the maximum number of links, and the other is with the minimum number of links. The first is actually a complete network, which means that for any two arbitrary nodes in the network, there is a link between them. The other form is a tree, which means that between any two nodes in the network, there is a unique path between them.[2] When the architecture of a software system becomes too complex, we need to perform some antiregressive activities [19], [20] to reduce the complexity of the system.

In a network, when a message is transferred from one node to another node, it can go through one of many possible paths. The number of possible paths can be defined as the freedom within the network. In a complete network,[3] the number of possible paths reaches the maximum or has the maximum freedom. In a tree, because there is only one path between any two nodes, the freedom comes to the minimum. In a network with less freedom, there are fewer choices for the message passing between nodes, i.e., it is less ambiguous or easier to manage. However, too few links may cause some links or nodes to be overloaded and, in certain cases, may cause some paths to be too long, eventually affecting the efficiency. An optimized architecture of a dependence network needs to balance these two factors.

The association between scale-free networks and optimized sorting algorithms suggests that, for complex systems, an optimized architecture may have the scale-free property that lies somewhere between a complete network and a tree.

B. Why Some Complex Networks Are Not Scale-Free

Many complex networks from different disciplines are discovered to be scale-free, but there are still large complex networks that are not scale-free. A good example is railway networks (Fig. 11).

Fig. 11 shows a typical railway network and a typical airline network. It is obvious that the structures of these two networks are very different. The airline network has a few hubs, which is the most significant feature of a scale-free network, but the railway network does not have this feature. Consequently, we may draw the conclusion that large airline networks are likely to be scale-free networks but large railway networks are not. Barabási and Bonabeau [5] claim that the U.S. highway system is a random network but the U.S. airline system is a scale-free network. This leads to a very interesting question. Both are transport networks, but why does one show scale-free features and the other not?

Our answer to this question is that, unlike an airline network, a railway network is a planar graph [10] built on a 2-D surface. Checking the railway network in Fig. 11, we discover that, for

[2]A path is a sequence of connected links with no node occurring more than once on it.

[3]In a network, if for every pair of nodes there is a link between them, this network is called a complete network.

a railway network, even though the intersections of links are not totally forbidden, it is very limited for links to intersect in the 2-D network. From this, we have a conjecture, *if a network is built on a 2-D surface and the intersection of links is prohibited, then this network will not be able to evolve into a scale-free network*. It could be difficult to prove this conjecture in the general case, because there is no general mathematical model for a scale-free network yet. However, based on the observation, a common property of scale-free networks is a high clustering coefficient [4], which means that small groups of nodes may form complete graphs where every pair of nodes in a group are directly connected. However, based on a corollary of Euler's formula, it is known that a complete graph of five, which is denoted as K^5, cannot be contained in a planar graph [10]. It is therefore reasonable to conjecture that a scale-free network cannot be drawn as a planar graph.

The limitations of a scale-free network on a 2-D surface inspire us to think about another interesting mathematical problem: the four-color mapping problem. The original problem is that, for any map drawn on a 2-D surface, we need at most four different colors to draw each area and guarantee that there exists a drawing method so that no two neighboring areas (that means there is a piece of mutual boundary between the two areas) are drawn in the same color. This problem can be transferred into a network problem. Each area can be treated as a node, and a link between two nodes indicates that the two represented areas are neighbors. The delicate part is that the network is built on a 2-D surface, and no two links can be intersected.

If we shift the four-color mapping problem into a 3-D space, what is the minimum number of colors that is sufficient to make sure every two connected nodes are drawn with different colors? The answer is obvious: No fixed number of colors can guarantee that any connected nodes can be colored differently for arbitrary networks built in a 3-D space. When the number of nodes and the number of connections are increased, the minimum number of colors required will be increased without any limitation. This result indicates that a 3-D space is fundamentally different from a 2-D space in regard to the ability to host complex networks. The four-color mapping problem addresses a crucial restriction for the complexity of networks that can be evolved in a 2-D space. Circuit-layout problems [27] also reflect a similar limitation in a 2-D space. However, when we consider a 3-D space, none of the discussed restrictions exist. Networks of any topological structures can be built in a 3-D space. Then, we come back to the original question in this section, why are some large networks such as a railway network not scale-free? The answer is that scale-free networks can probably only be developed in a space of more than two dimensions.

Many large scale-free networks are built in virtual space where the dimension is hard to determine, such as the human-relationship networks. However, physical complex networks such as a brain, which is a network of nerve cells, are built in a 3-D space. We may conjecture that, due to the limitations for forming complex networks, objects of certain complexity such as a life form can only be created (or evolve) in a universe of at least three dimensions.

C. Three Searching Methods

Large networks evolve by incrementing of their size. During the growth process, there are two essential operations. The first is to add new nodes, and the second to create new connections in the existing network. Here, we discuss only the creation of new links.

After a node is created, it needs to be connected to other nodes. The problem is how the new node can search for the most suitable target node for it to connect with. Generally, there are three types of searching methods.

1) **Random searching**: Where the new node tries to connect to other nodes randomly and there is a chance that it may find a suitable target. The chance increases if the new node keeps trying. This method mostly relies on luck, and may be applied in certain circumstances (perhaps Edison adopted this methodology when he was trying to find a suitable material for the thread in a light bulb). Usually, it is not a good strategy and may take a long time to find a suitable target and establish a useful connection.
2) **External diverted searching**: Here, a new node discovers the best possible connection from an external information provider. This searching method is based on the assumption that there exists a most suitable target node for the new node, and this knowledge is known by someone. If the source node can acquire this knowledge, it can directly make the best connection. For a small system designed by an individual, that person has an overall view of the system and total control over the system; thus, the designer can be an external information provider. However, for many real-world large systems, these preconditions do not apply; the external information provider either does not exist or is unapproachable. For example, in a social network, if a person is looking for a job, there may be a best job opportunity for him but the problem is how he could find it.
3) **Self-adjusting searching**: Here, at the beginning, the new node tries to determine a possible best node based on its own knowledge and makes a testing connection (first, a few test connections can be based on a random searching method). After one or a few testing connections, the new node will be able to collect some feedback from the connected nodes. This feedback may help the new node to narrow the search scope and find better testing nodes. After trying the new testing nodes and getting new information, the new node will eventually find a satisfied target node and makes a link. This search method is obviously a practical one and is adopted by most real networks. In the job-search example, a possible scenario is that the person may contact his most knowledgeable friend first; his friend may advise him to try a job agent; the job agent may recommend a potential employer, etc. Generally, this searching method is a learning process. The more the new node learns from the existing connections, the quicker it is able to reach a suitable target node.

Here, we have discussed three different searching methods for a new node to discover the most suitable target node. The

topological structure of the network will be different for the different searching methods.

If the random searching method is applied, the growing network may have the structure of a random network. If the external diverted searching method is applicable, the generated network will not have redundant connections and it can be in its simplest form, which is a tree.

In general, if the self-adjust searching method is adopted with an optimized algorithm to use the information collected from the testing connections, then we conjecture that the network will be woven as a scale-free network. The study of the SCNs in this paper supports this conjecture. A sorting algorithm uses the self-adjust searching method to construct the SCN. After each comparison, the new comparison will be selected based on the previous comparison results. A more efficient sorting algorithm simply means that it makes better use of the information collected from the testing connections. An optimized sorting algorithm can nearly maximize the usage of the information, and in this situation, we know that the SCNs are scale-free.

For large software systems, since they are usually "grown" rather than built [6], the information about the best connection points for new components is usually not clear. The lack of information leads to more connections than necessary, and the growing process is similar to a network developed on a self-adjusting searching method, and it is not surprising that the final structure is usually a scale-free network.

D. Theoretical and Practical Values

Based on our test results and independent results from other researchers [7], we conjecture that the CDNs for most software systems may be scale-free networks. The result may have following theoretical and practical values.

1) It suggests that the same laws are the foundation for the evolution of software systems and other complex networks, such as social and biological networks.
2) Based on the scale-free property, important components can be identified in large software systems.
3) If the degree distribution of a software system's CDN drifts from a power law to normal, it may indicate that the structure of the system has been overdeveloped, and a major cleanup or antiregressive [19] process is required.
4) For a scale-free network, random errors may not cause major crashes of the whole system. This may be why many large software systems can function properly most of the time even with software defects. However, if errors happen at some critical points in a scale-free network, minor problems can cause the whole system to fail. Therefore, to secure a large software system, extra precautions are recommended for the relatively small number of key components.

The discovery of the relationship between scale-free networks and optimized sorting algorithms provides an efficient approach to study the evolution of large complex networks.

Many real-life large networks such as human social networks take many years or decades to evolve, and the evolutionary process is affected by so many uncontrollable factors that the evolution seems to be stochastic. Therefore, it is very hard to trace all the details of the whole evolution and do some experiments that can be easily repeated. However, for an SCN, once the sorting algorithm and the input sequence are given, the evolution and the final structure of the SCN is determined. Therefore, an SCN provides a more repeatable approach in studying the nature of scale-free networks.

In CI, a CN shows how natural intelligence [31] stores and retrieves knowledge. Besides the hierarchical, dynamic, and evolvable nature [30] of the structure of a CN, the topological details of a CN are still unknown. Because of the similarity between a CN and an SCN (both of them are networks that arrange records in a form so that the required information can be retrieved efficiently), we conjecture that a CN could also be a scale-free network.

VI. Conclusion

In this paper, we have studied the topological structures of eight Java packages' CDNs and five different sorting algorithms' SCNs in regard to a new network model, the scale-free network [5]. Through this paper, we have made two discoveries.

The first discovery is that all the Java packages' CDNs clearly show scale-free properties. The second discovery is the relationship between scale-free networks and optimized sorting algorithms. These discoveries are interesting, because they show the similarity of the structures of networks from different domains.

Parallel studies between different disciplines frequently inspire new ideas. In recent years, people have started to research the commonalities between software and biological evolution [25], the similarity between a virtual society and real society [1]. A number of interesting discoveries have been made. The similarity between the topological structure of a CDN, an SCN, and other types of complex networks implies that the laws, which work behind the evolution of software systems, could be the same as those working behind the evolution of other complex systems such as human society and biological systems. Continuing studies may reveal more commonalities among the structure and evolution of those different complex systems and, therefore, benefit complex system theory in general.

Acknowledgment

The authors would like to thank their colleagues for the discussions, Prof. F. Dehne for his feedback on some of the work in this paper, and the anonymous reviewers for their comments.

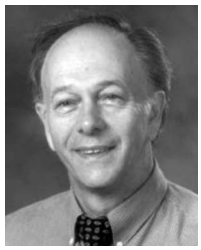
References

[1] R. Alberich, J. Miro-Julia, and F. Rosselló, *Marvel Universe Looks Almost Like a Real Social Network*, vol. 1, Feb. 11, 2002. oai:arXiv: cond-mat/0202174.
[2] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks," *Rev. Mod. Phys.*, vol. 74, no. 1, pp. 47–97, Jan. 2002.
[3] *Apache Ant*, 2007. [Online]. Available: http://ant.apache.org/
[4] A.-L. Barabási, *Linked*. Cambridge, MA: Perseus Publishing, 2002.
[5] A.-L. Barabási and E. Bonabeau, "Scale-free networks," *Sci. Amer.*, vol. 288, pp. 60–69, May 2003.
[6] F. P. Brooks, "No silver bullet: Essence and accidents of software engineering," *Computer*, vol. 20, no. 4, pp. 10–19, Apr. 1987.

[7] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Trans. Softw. Eng.*, vol. 20, no. 6, pp. 476–493, Jun. 1994.
[8] *Classnet, The Tool to Explore the CDN of a Java System*, 2007. [Online]. Available: http://www.sqi.gu.edu.au/gse/tools/classnet.html
[9] I. F. Cruz and R. Tamassia, *Graph Drawing Tutorial*, 1998. [Online]. Available: http://www.cs.brown.edu/people/rt/papers/gd-tutorial/gd-constraints.pdf
[10] R. Diestel, *Graph Theory*, 2nd ed. New York: Springer-Verlag, 1999.
[11] R. G. Dromey, "From requirements to design: Formalizing the key steps," in *Proc. IEEE Int. Conf. SEFM*, Brisbane, Australia, Sep. 2003, pp. 2–11. (Invited Keynote Address).
[12] T. Eisenbarth, R. Koschke, and D. Simon, "Locating features in source code," *IEEE Trans. Softw. Eng.*, vol. 29, no. 3, pp. 210–224, Mar. 2003.
[13] H. Jeong, B. Tombor, R. Albert, Z. N. Oltvai, and A.-L. Barabási, "The large-scale organization of metabolic networks," *Nature*, vol. 407, no. 6804, pp. 651–654, Oct. 2000.
[14] H. Jeong, S. P. Mason, and A.-L. Barabási, "Lethality and centrality in protein networks," *Nature*, vol. 411, no. 6833, pp. 41–42, May 2001.
[15] B. J. Kim, A. Trusina, P. Minnhagen, and K. Sneppen, "Self organized scale-free networks from merging and regeneration," *Eur. Phys. J. B*, vol. 43, no. 3, pp. 369–372, Feb. 2005. [Online]. Available: http://arxiv.org/abs/nlin/0403006
[16] J. M. Kleinberg, "Authoritative sources in a hyperlinked environment," *J. ACM*, vol. 46, no. 5, pp. 604–632, Sep. 1999.
[17] D. E. Knuth, *The Art of Computer Programming, Sorting and Searching*, 2nd ed, vol. 3. Reading, MA: Addison-Wesley, 1997.
[18] A. Lakhotia, "A unified framework for expressing software subsystem classification techniques," *J. Syst. Softw.*, vol. 36, no. 3, pp. 211–231, Mar. 1997.
[19] M. M. Lehman, "Programs, cities, students, limits to growth?" *Inaugural Lecture*, 1974.
[20] M. M. Lehman, "FEAST/2 final report—Grant number GR/M44101," Dept. Comput., Imperial College, London, U.K., 2001.
[21] K. Park and Y.-C. Lai, "Self-organized scale-free networks," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 72, no. 2, p. 026 131, Aug. 2005.
[22] R. Pastor-Satorras and A. Vespignani, "Epidemic spreading in scale-free networks," *Phys. Rev. Lett.*, vol. 86, no. 14, pp. 3200–3203, Apr. 2001.
[23] *Sorting Comparison Network Explorer*, 2007. [Online]. Available: http://www.scs.carleton.ca/~cgm/scale-free/
[24] J. A. Stafford and A. L. Wolf, "Software architecture," in *Component-Based Software Engineering, Putting the Pieces Together*. Boston, MA: Addison-Wesley, 2001, ch. 20.
[25] D. Svetinovic and M. Godfrey, "Software and biological evolution: Some common principles, mechanisms, and a definition," in *Proc. 8th IWPSE*, Lisbon, Portugal, 2005.
[26] V. Tzerpos and R. C. Holt, "ACDC: An algorithm for comprehension-driven clustering," in *Proc. 7th Work. Conf. Reverse Eng.*, 2000, pp. 258–267.
[27] W. M. Vancleemput and J. G. Linders, "An improved graph-theoretic model for the circuit layout problem," in *Proc. 11th Workshop Des. Autom.*, 1974, pp. 82–90.
[28] Y. Wang, "On cognitive informatics," *Brain Mind: A Transdisciplinary J. Neurosci. Neurophilosophy*, vol. 4, no. 2, pp. 151–167, 2003.
[29] Y. Wang, "The OAR model for knowledge representation," in *Proc. 19th IEEE Can. Conf. Elect. Comput. Eng.*, 2006, pp. 1696–1699.
[30] Y. Wang, "On concept algebra and knowledge representation," in *Proc. 5th IEEE Int. Conf. Cognitive Informat.*, 2006, pp. 320–331.
[31] Y. Wang, "The theoretical framework of cognitive informatics," *Int. J. Cognitive Informat. Natural Intell.*, vol. 1, no. 1, pp. 1–27, Jan. 2007.
[32] D. J. Watts and S. H. Strogatz, "Collective dynamics of small-world networks," *Nature*, vol. 393, no. 6684, pp. 400–442, Jun. 1998.
[33] D. J. Watts, *Six Degrees: The Science of a Connected Age*. London, U.K.: Heinemann, 2003.
[34] L. Wen and R. G. Dromey, "Architecture normalization for component-based systems," *Electron. Notes Theoretical Comput. Sci.*, vol. 160, pp. 335–348, 2006.
[35] L. Wen, D. Kirk, and R. G. Dromey, "Software systems as complex networks," in *Proc. 6th IEEE Int. Conf. Cognitive Informat.*, 2007, pp. 106–115.
[36] *Wolfram Math World*, 2007. [Online]. Available: http://mathworld.wolfram.com/Sorting.html
[37] A. Zaidman, T. Calders, S. Demeyer, and J. Paredaens, "Applying web-mining techniques to execution traces to support the program comprehension process," in *Proc. 9th Eur. Conf. Softw. Maintenance Reengineering*, 2005, pp. 134–142.

Lian (Larry) Wen received the B.S. degree in mathematics from Peking University, Beijing, China, the M.S. degree in electrical engineering from the Chinese Academy of Space Technology, Beijing, and the Ph.D. degree in software engineering from Griffith University, Brisbane, Australia, in 2007.

He has worked as a Software Engineer and a Project Manager for different IT companies. He is currently a Research Fellow with the Software Quality Institute, Griffith University. His current research interest includes behavior engineering, complex systems, scale-free networks, software change, and evolution and quantum computing.



R. Geoff Dromey (M'08) received the Ph.D. degree in chemical physics from La Trobe University, Melbourne, Australia, in 1973.

He has worked with Stanford University, Stanford, CA, The Australian National University, Canberra, and Wollongong University, Wollongong, N.S.W., Australia. He has also spent sabbaticals at Oxford University, Oxford, U.K., Stanford University, and Strathclyde University, Glasgow, U.K. He is the Foundation Professor of software engineering with the School of Information and Communication Technology, Griffith University, Brisbane, Australia, where he is the Founder and the Director of the Software Quality Institute. He is also a cofounder of a successful software company, Calytrix Technologies, Perth, Australia, which works in the Defence Sector and has offices in Australia and the USA. His current research interests include development methods that control complexity in the analysis and design of large-scale software integrated systems. He has authored two books in leading international computer science series (Prentice–Hall and Addison-Wesley) and authored/coauthored more than 50 research journal papers. He serves on the editorial boards of three international journals.

Diana Kirk received the M.S. degree in computer science from the University of Edinburgh, Edinburgh, U.K., in 1985 and the Ph.D. degree from the University of Auckland, Auckland, New Zealand, in 2007.

She is currently with the Computing and Mathematical Sciences, Auckland University of Technology, Auckland. Her career in the software industry spans 20 years and has taken her on a path through technical programming, codirecting an emerging company, project management, software quality management, and consulting. Her experiences have resulted in a deep appreciation of the issues, both technical and human, that affect software project outcomes. Her main interests relate to maximizing effectiveness in the processes applied during software projects.