Thoughts on Weak Links and Alexandrian Life in Scrum

Pamela M. Rostal
Perficient, Inc.
100 North 6th St, Suite 935C
Minneapolis, MN 55403
001-651-308-0917
pmrostal@comcast.net

ABSTRACT

This paper looks at the Scrum software development process through a lens that emphasizes small worldness, nestedness, and scale-freeness, all characteristic of networks that feature weak links between their modules. Scrum has gained popularity over the past decade as a means of delivering valuable software to its host organization on a regular basis. Since weak links characterize natural and social systems at every scale, practitioners of Scrum should be able to improve their teams' processes by applying lessons learned from studying weak links. When practitioners look for weak links directly, they may find the task daunting and ask the question: "How can I tell whether weak links are strengthening or weakening my team's Scrum process if I can't even find them?" For the answer, this paper looks to Christopher Alexander's characteristics of wholeness, integrity, or ***life*** **– *strong centers, levels of scale, echoes, alternating repetition***, and, in particular, the characteristic called ***deep interlock and ambiguity*** -- which may correlate with the presence of weak links.

Categories and Subject Descriptors

K.6.3 Software Management -- Software development, Software process. D.2.9 Management -- Programming teams. D.2.8 Metrics -- Process metrics

General Terms

Management, Measurement, Performance, Theory

Keywords

Scrum, patterns, Nature of Order, Christopher Alexander, weak links, life

PLoP'08, October 18-20, Nashville, TN, USA.

ACM 978-1-60558-151-4

1. INTRODUCTION

Imagine a world without jet engines, ecological systems, and the Internet, all casualties of eliminating the weak links [1] that characterize them. Now, imagine a world without any living creatures – since weak links are also essential to life processes such as protein folding, cell repair, and protein-protein interactions [1]. Given their importance to life on this planet, an examination of weak links in the activities, structure and artifacts of the software development process represents an exciting opportunity. This paper explores the Scrum [2, 3] software development process in terms of stabilization by weak links and suggests a relationship between them and Christopher Alexander's concept of wholeness, integrity, or ***life*** [4]. Scrum was chosen because its perceived successes have motivated over 15,000 people across the globe to seek certification as Scrum Masters (the leaders of Scrum teams) [5]. The goal of this investigation is to seek out the weak link-related factors behind Scrum's appeal and suggest new areas of exploration for process improvement.

These new areas of exploration involve small worldness, nestedness, and scale-freeness, all characteristic of networks that feature weak links (see the *Glossary in a Box* on the next page). If stabilization by weak links is at least partially responsible for Scrum's success, it should prescribe practices, structures, and artifacts that exhibit these attributes, showing that weak links have emerged to help stabilize the software development process. The attributes may show up in the way Scrum decomposes the work required to develop a system, the communication processes Scrum prescribes to keep stakeholders engaged, or the structures Scrum specifies for adapting to disruptions of the development process. Furthermore, if ***life*** indicates that weak links are stabilizing Scrum, then both its processes and their resulting products should exhibit the characteristics Christopher Alexander associates with ***life***. Efforts to improve the process should then focus on intensifying the evidence of life by altering the distribution of link lengths and strengths.

In the book *Weak Links: Stabilizers of Complex Systems from Proteins to Social Networks* [1], author Peter Csermely develops a compelling case for the view that weak links in and around complex systems exert a stabilizing effect. Numerous diverse disciplines have examined weak links, studying their role in atomic, molecular, and macromolecular interactions [6, 7], interactions in social situations [8], software system architectures [9], and links among web pages [10]. While previous researchers had observed weak links, it was Csermely who speculated on their universal role in system stabilization. Because every complex system is a network capable of weak link stabilization, software development should be no exception.

The pages that follow present key concepts common to weak links and Alexander's phenomenon of life and shows how they are exemplified in Scrum.

In very simple systems, there are no weak links because weak links between elements emerge due to interactions among interactions. Simple systems have only one set of interactions – all of them strong links, so the removal of any link will likely destroy the system. In contrast, removing a weak link often has little impact on the system in the short run because, by definition, "a link of a network is weak if its addition or removal does not change the mean value of a target measure, which is usually an emergent property of the network, in a statistically discernible way" [11]. Forgetting a birthday card can have huge repercussions if the forgetful party is a close friend or spouse, but the lack of a card from a distant relative has virtually no impact.

As an example of weak links from software development, team productivity is often measured by metrics such as lines of code, function points, or story points per iteration. Productivity, as measured by these metrics, may not suffer much if a water cooler is removed (see the Water Cooler organizational pattern [12]), but the effectiveness of the software delivered may suffer over an extended period of time because of the team's losing touch with people and information from other domains. A story reported by Gerald Weinberg [13] and cited by Alistair Cockburn [14] illustrates this principle in a parallel context:

> [At] a large university computing center ... a large common space was provided near the return window so that the students and other users could work on their programming problems. In the adjoining room, the center provided a consulting service for difficult problems, staffed by two graduate assistants. At one end of the common room was a collection of vending machines ... the noise from the revelers congregating at the machines often became more than some of the workers could bear ... [The computing center manager] went to investigate their complaint ... Without more than 15 seconds of observation, he went back to his office and inaugurated action to have the machines removed to some remote spot. The week after the machines had been removed – and signs urging quiet had been posted all around – the manager received another delegation ... They had come to complain about the lack of consulting service; and, indeed, when he went to look for himself, he saw two long lines extending out of the consulting room into the common room. He spoke to the consultants to ask them why they were suddenly so slow in servicing their clients ... For some reason, they said, there were just a lot more people needing advice than there used to be ... After some time, he discovered the source of the problem. It was the vending machines! When the vending machines had been in the common room, a large crowd always hovered around them ... they were chatting about their programs. ... Since most of the student problems were similar, the chances were very high that they could find someone who knew what was wrong with their programs right there at the vending machines. Through this informal

Glossary in a Box

NOTE: Throughout this glossary, the assumption is made that any system (i.e., anything) can be viewed as a network.

Small-worldness: a characteristic of networks whose nodes are not necessarily connected directly, but are generally no more than a few short or long hops from each other. Small worlds benefit from both clustering and long-range connections. The notion that virtually any two people in the world can be connected in no more than six hops [15, 8] is a commonly cited example of small-worldness.

Nestedness: the characteristic of a network that allows a given node to appear as a leaf from the top of the network, but as a whole network from its own perspective. The typical corporate organization chart is a perfect example of nestedness because each box can represent either a single employee or the manager of a network of subordinates, some with additional nested managers.

Scale-freeness: the characteristic of a network that allows it to have the same properties regardless of how large it grows. Many people are familiar with Pareto's 80-20 rule whereby 20% of the people in a given economy are responsible for 80% of the wealth [16]. Regardless of how large the economy grows, the percentage remains the same because the majority of the money finds its way into the hands of the wealthiest 20%. Hence, the distribution of wealth is scale-free.

Weak Links: low-intensity connections or transient higher intensity connections between network elements. In a stable network, they constitute one end of a spectrum of link strengths comprising a very few strong links and increasingly more numerous and less intense links. The social network "Linked In" is an example of weak links between professionals. Generally, a member will not correspond with all the people in his/her network, but during a period of unemployment, messages may be sent to even the most distant acquaintances.

Motif: small groups of network elements that characteristically appear together in specific linkage patterns; e.g., Alexander's architectural patterns joining context, forces, and solution, feed-forward loops, and feedback loops.

Module: a group of relatively isolated network elements that are more functionally or physically linked to each other than they are to the rest of the system; e.g., code that implements an object, a car full of people in a traffic jam, or a team on a large project.

organization, the formal consulting mechanism was shunted, and its load was reduced to a level it could reasonably handle.

In this example, if the consultants were the only source of the knowledge required to solve the students' problems, then the system would have been a simple system with a bottleneck. Because the knowledge could be constructed or obtained from others (whose identity was not known by the students but whose location might be inferred as the common area), the system is complex. Systems with only strong links are susceptible to disruption and even disintegration when disrupted by a disturbance because the perturbation cannot dissipate except by passing through the whole system or by breaking off an entire subsystem. In the example above, the long line that formed is a perturbation that slowed the students' access to the consultants. If all the students had given up on obtaining help because of the long lines, the consulting system would have disintegrated without further intervention. If the students had formed their own study groups as an alternative to the consulting service, that would have been an example of breaking off an entire subsystem.

In the grand scheme of complex systems, the student/consulting service story is a relatively simple example. In contrast, the software development process forms a highly complex system comprising enterprise and project-level goals and plans, realized by team and individual collaborations throughout the system development lifecycle. Given all of these interactions among interactions, one should be able to detect evidence of motifs, modules, small worldness, nestedness, and scale-freeness there.

Christopher Alexander was able to recognize the characteristics of weak links in spatial systems from around the world [4]. His book, *The Phenomenon of Life,* explores a theory "in which statements about relative degree of harmony, or life, or wholeness – basic aspects of order – are understood as potentially true or false." Because his fifteen basic characteristics of life relate strongly to the properties associated with weak links and because they provide a visual representation of several abstract concepts related to weak links, they are helpful in drawing the distinctions presented in the subsequent **Core Concepts** section.

Table 1 Characteristics of *Life*

Christopher Alexander's 15 characteristics of "Life" in spatial systems	
Levels of scale – the range of sizes that centers generally exhibit in a series of discrete, well-marked levels; a ratio of 20:1 between centers is too large; 2:1 or 3:1 is appropriate.	***Contrast*** – intense differentiation caused by the disparity between distinctly opposed centers whose boundary emerges as a center itself
Strong Centers – wholes, existing at different levels of a structure, that capture the observer's interest as individual focal points. They are locally symmetrical and structurally complementary in relation to the whole system	***Gradients*** – the gradual variation of a quality across space as its centers adapt to changes in the morphology of the space they inhabit
Boundaries – that which both unites the center being enclosed with the world around it and intensifies the identity of the center itself. Boundaries must be of the same order of magnitude as the centers they bound and must also interlock and connect with them.	***Roughness*** – subtle variation in a system's property due to its being created or evolved in a non-mechanical, egoless way that allows the property to optimize itself to its immediate environment
Alternating Repetition – rhythmic intensification of centers caused by wave-like repetition of two or more systems whose centers complement and enhance each other's centers as well as their own.	***Echoes*** - deep internal similarities between the small centers and angled pieces of larger centers, which tie them together to form a single unity
Positive Space – "when every bit of space swells outward, is substantial in itself and is never the leftover from an adjacent shape." [4, p. 173]	***The Void*** - "In the most profound centers which have perfect wholeness, there is at the heart a void which is like water, infinite in depth, surrounded by and contrasted with the clutter of the stuff and fabric all around it." [4, p. 222]
Good Shape – a shape that comprises recursive compact coherent centers, each exhibiting the characteristics in this list	***Simplicity and Inner Calm*** – the slowness, majesty, and quietness inherent in structures that reveal only their essence, with no unnecessary appendages
Local Symmetries – subtly interweaving small-scale symmetries that support organic, flexible adaptation to the system's context by binding its centers into a coherent whole	***Not-separateness*** – when a center having deep life evokes a feeling of connectedness to what surrounds it and is not cut off, isolated, or separated
Deep Interlock and Ambiguity – the interface zone, both system and context, where centers hook into their surroundings	

Buschmann et al. in *Pattern Oriented Software Architecture Volume 5: On Patterns and Pattern Languages* [17, p. 380] stated that the fundamental properties of life are defined in a work (the four volumes of *Nature of Order* [4], [18], [19], [20]) whose "contributions will be primarily to support deepening of our understanding of pattern concepts such as pattern sequences and the notion of 'centers' in design." While this may be true, its greater contribution may be to direct the pattern community's gaze towards areas outside the existing focus of patterns. It may be that the fundamental properties of life are actually symptoms or indicators of an order supported by weak links whose presence can be measured using methods borrowed or derived from those in other disciplines. If so, then *The Nature of Order* may have made a far more profound contribution – bringing the

process of software development into alignment with realities studied in other professions and expressed in nature.

2. CORE CONCEPTS

Note: Where Alexander's characteristics appear, they are formatted in ***bold italics***.

2.1 Small-worldness

The short definition of small-worldness is "a characteristic of networks whose nodes are not necessarily connected directly, but are generally no more than a few short or long hops from each other." While this definition suffices initially, it must be qualified to exclude random graphs, which are not small worlds but have elements that can reach each other with relative ease. The differentiator between the two is clustering, which enhances local searches. Clusters are formed when neighboring nodes link not only to each other but to additional common nodes – a sort of "my friend's friend is also my friend" phenomenon.

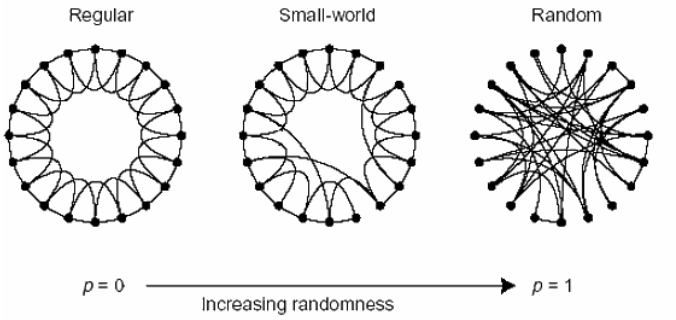

Figure 1 Small-worldness in context [21]

Figure 1, from a study by Watts and Strogatz [21], shows three types of networks: regular, small-world, and random. Note that in the regular network, where breaking and reconnecting a link would always bring it back to the same node, every node is connected to its two nearest neighbors in each direction so that it can get to any of them very efficiently, giving it a high clustering coefficient. Random networks, on the other hand, where the probability (p) of a disconnected node being reconnected somewhere else is 1, may reach nodes on the far edge of the circle quite easily, giving a short characteristic path length, but they lack the high clustering coefficient. Watts and Strogatz found that intermediate values of p characterized structures that benefit from the clustering of regular graphs and from the shortcuts of random graphs, giving them the enhanced signal-propagation speed, computational power, and synchronizability that distinguish small worlds. Just a few shortcuts in a system with a large number of nodes enables a small path length without disturbing the clustering of the pre-existing neighbors because each shortcut connects not only the affected node, but its entire neighborhood to the far side of the network.

An example in the real world is a person needing an answer to a question, as shown in Figure 2. The information might link strongly to one document because the author was a consultant (who may still have a strong link to the information, but whose access was eliminated due to contract termination – a major perturbation in the system that created the information). The questioner's path to the information might follow the paths between large boxes in Figure 2 and take precious time away from the task for which the information was required.

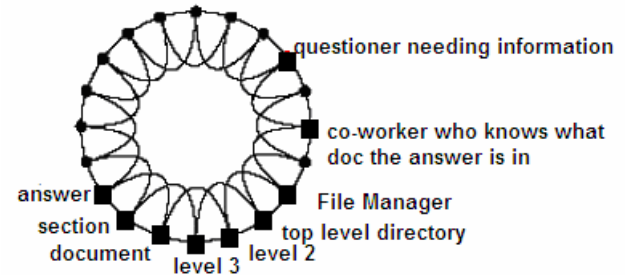

Figure 2 Information strongly linked to a single document

If, instead of the information being available only in the repository document, it was also available from a friend who was the document's author, the same questioner might get to the information in just three hops instead of the eight required above. The network in Figure 3 shows how the agile notion of osmotic communication [22] shortcuts the communication path, resulting in faster access to the information that will allow the questioner and team to continue working.

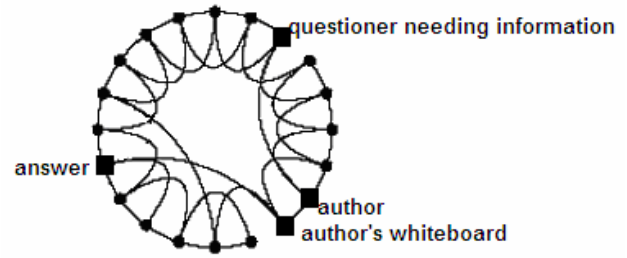

Figure 3 Information available from the author's whiteboard

Viewing the network in Figure 3 from the perspective of Alexander's life, the whiteboard would represent a ***strong center***. The system of whiteboards in people's cubes, in the hallways, and at gathering points could constitute a system of ***strong centers*** if they contained information that encouraged people to cluster around them and discuss the information displayed there. Information ***gradients*** form around these strong centers – those closest to the centers are most conversant with their contents, while those farthest away know the least. Small worldness addresses communication within a co-located community, but when people are too far apart, the shortcuts can cost too much to sustain, causing the system to break into highly clustered subnetworks that do not communicate.

Whether the reason is distributed teams or simply poor communication among stakeholders, lack of communication causes problems. Conway's Law, states "Any organization that designs a system (defined more broadly here than just information systems) will inevitably produce a design whose structure is a copy of the organization's communication structure." [23]. Only a concerted effort to support the shortcuts – e.g., communication-focused roles such as business analysts, project managers, or leadership groups supported as needed by specialized collaborative tools – can stabilize the network and allow for scale-free growth, as described in the next section.

2.2 Scale-freeness

Scale-freeness is defined as the characteristic of a network that allows it to have the same properties regardless of how large it grows. Adding more nodes to the 20 (a number too small for most research, but one that is easy to visualize) in the regular graph of Figure 1 will increase the average distance between nodes in a linear fashion. Adding nodes to the random network in Figure 1 only increases the average distance between nodes proportionally to ln(N), where N is the number of nodes. Even so, the random network does not show the benefits of a scale-free network because, as noted above, it does not have the high clustering coefficient characteristic of networks that have many nearby neighbors. The middle network of Figure 1 *does* show scale-freeness with growth because any new node leaves the clustering coefficient relatively high and the average distance between nodes relatively short due to its neighbor's shortcuts.

Examples of scale-freeness include the great cities of Europe (e.g., Budapest and Paris), whose neighborhoods are scale-free networks that developed fractally as self-organizing systems within ever larger self-similar systems. Some dimensions showing scale-free distributions are object size (from small shop windows to town squares), verticality (small homes to large buildings and sometimes to large hills in the distance), and the fractal distribution of distances that can be walked without running into an obstacle [1].

From the realm of nature, the Roystonea or Royal Palm survives hurricane-force winds [24] by shedding its fronds when the force of the storm endangers the survival of the tree. The fronds exhibit a distribution of "stickiness" to the tree such that a 40-50 mile-an-hour wind from a thunderstorm will blow down one frond, while a hurricane will remove all the fronds but leave the tree standing to grow new ones. Thus, as the author suggests, the number of fronds removed can provide an estimate of the strength of the storm.

From the realm of human design, the Alhambra tile shown in Figure 4 could scale indefinitely because enlarging it involves only adding new self-similar modules. The ratio of stars to the other elements of the design remains the same whether the

Figure 4 Alhambra tile [25]

observer looks at a small rectangle encompassing just one star and its surrounding elements or at a rectangle that encompasses the far-distant edges of the tile. The ratio of each color to each other color is also maintained as the tile grows. The stars form ***strong centers,*** while the surrounding elements form ***boundaries***, and ensure ***local symmetries***. The ***local symmetries*** combine to create coherence on a large scale because they emerge organically rather than being imposed by an overenthusiastic designer.

In contrast, Figure 5's view of the Renaissance Center in Detroit shows what happens in the absence of scale-freeness since the five large buildings emerge abruptly from the ground, without elements of intermediate heights producing scale-free verticality to bridge the stark contrast in height. Because each tower diverges so drastically from its immediate surroundings and appears so disconnected from the other towers, there is no opportunity for ***local symmetries.*** Figures 4 and 5 illustrate the coherence that emerges from small local symmetries and the lack of coherence apparent when large-scale symmetry is attempted without the ground of adjacent small ***local symmetries***.

Figure 5 Renaissance Center in Detroit [26]

The presence of local symmetries contributes to a perception of ***not-separateness*** in the Alhambra tile because it exemplifies Alexander's definition that:

> any center which has deep life is connected, in feeling, to what surrounds it, and is not cut off, isolated, or separated. In a center which is deeply coherent, there is a lack of separation – instead a profound connection—between that center and the other centers which surround it, so that the various centers melt into one another and become inseparable. It is that quality which comes about from each center, to the degree it is connected to the whole world. [4, p. 231]

Viewing the tile from the perspective of ***not-separateness***, there is a sense that removing any element of the pattern would cause a loss of balance in the relationships of its neighbors and draw the eye of the observer to the unexpected gap in that spot. This would disturb the tile's integrity; hence, each element of the tile exhibits ***not-separateness*** and contributes to the sense of wholeness of the entire tile. In contrast, the Renaissance

Center's lack of ***local symmetries*** gives the observer no support for seeing the five towers and the space between them as one single whole; hence, there is no sense of ***not-separateness***.

2.3 Nestedness

Nestedness is defined as the characteristic of a network that allows a given node to appear as a leaf from the top of the network, but as a whole network from its own perspective. In symbiosis-driven nestedness or integration [27], nestedness evolves bottom-up, from relatively stable independent networks self-organizing into larger networks until they all associate to become elements of the top network. This might happen in suburbs that merge their bus lines to support a transportation system around the city. In contrast, modularization-driven nestedness or parcellation involves segregating parts of the top network into modules, which can then become top networks themselves – e.g., top-down decomposition of software designs characteristic of structured design or cities that govern through neighborhood councils. We will examine modules in detail later in this paper.

Employees recognize nestedness in a large company's organization chart, an example of which is shown in Figure 6. Note that John Sampleboss sees the three nodes in Figure 6 that report to him as leaves, but in reality, only Sandy Sampleadmin is a leaf. James Sampleposa and Jane Samplemgr are nodes that comprise additional nodes, some of them leaves and others further decomposable nodes.

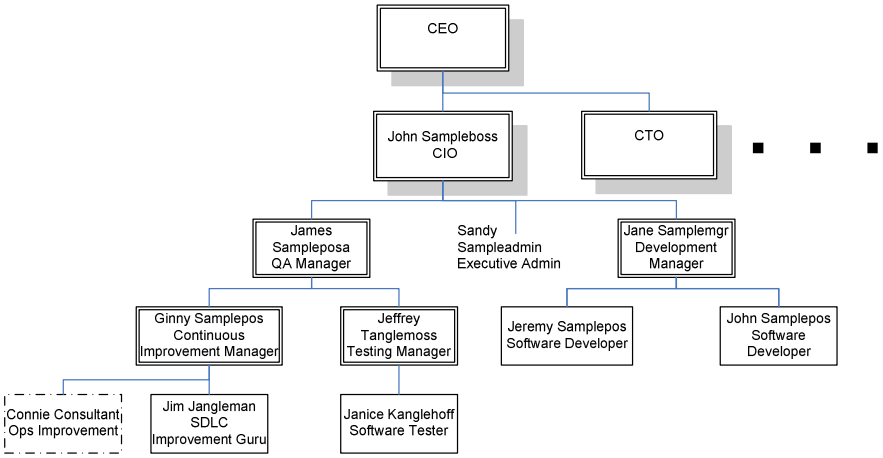

Figure 6 Nestedness in an Organization Chart

Nestedness allows organizations to scale by incorporating employees into departments (modules) exhibiting ***levels of scale, strong centers***, and ***boundaries*** that separate tightly linked employees within the department from those in other departments.

Another example of nestedness, this one from economics, is the world economy that comprises countries whose social networks are composed of people powered by networks of cells made up of protein networks that can finally be broken up into atoms. This example exhibits ***strong centers*** in its nested nodes, ***boundaries*** with ***contrast*** between the centers, ***levels of scale*** that define the top and the bottom of the network, and ***roughness*** due to uneven distribution of resources. Again, nestedness alone is not enough to guarantee life because the wholeness and integrity of the world economy, especially during the 2008 economic slowdown, appears to be questionable.

Some consequences of nestedness may not be obvious from the examples above. M. C. Escher's *Print Gallery* captures the concept of nesting visually in that the picture hanging in a print gallery appears to contain the gallery itself. Douglas Hofstadter, in his analysis of Escher's work [28, p. 714], speculates that the signature in the middle of the picture exists to cover up the fact that it would be impossible to capture the point at which the larger world intersects its nested image. This phenomenon may affect intersections other than spatial intersections and will be explored in greater detail during the discussion of Scrum planning meetings.

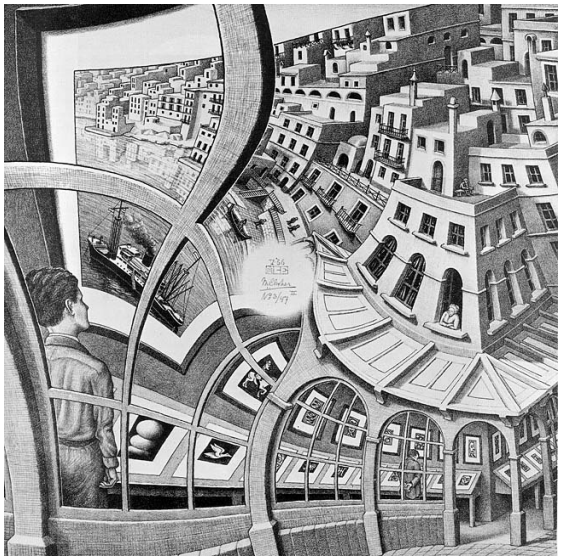

Figure 7 M. C. Escher's Print Gallery [29]

In this case, where the picture contains a replica of the gallery, which would presumably contain an even smaller replica, the

nested elements are replicas of each other, so they would be considered motifs.

2.4 Motifs

Motifs are defined as small groups of network elements with characteristic linkage patterns that occur in significant numbers within complex networks. Christopher Alexander's patterns are motifs that reflect the underlying processes of architectural design. The entire patterns community is founded on the understanding that problems in a given context, subject to the specified forces, have been successfully handled by the solution presented in the pattern, which makes the problem-context-forces-solution linkage a motif.

Motifs in software become design patterns [30], architecture patterns from the idiom level to the application architecture level [31] or enterprise application architecture patterns [32]. Motifs in implemented software become reusable components when they are extracted from existing software systems by reverse-engineering [33].

Motifs have also been found in many fields outside of architecture and software, including biochemistry, neurobiology, ecology, and engineering [34]. Research by Milo et al. has led them to believe that network motifs "can define broad classes of networks, each with specific types of elementary structures" [34]. Systems like food webs that support the flow of energy from the bottom of the food chain to the top exhibit different distributions of motifs than information processing systems. Information processing networks feature 3-element feed-forward loops as shown on the left in Figure 8, although some bi-parallel structures are also found in these systems; food webs, on the other hand, feature some feed-forward loops, but bi-parallel structures, allowing a choice of vehicles to the destination, are far more prevalent.

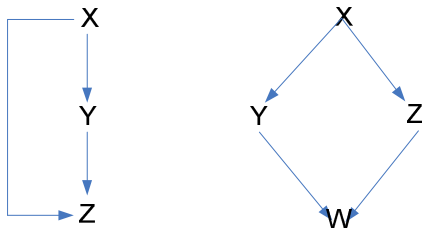

Figure 8 Three-element Feed-forward loop and Four-element Bi-parallel structure

In building architecture, motifs correspond to Alexander's ***echoes***, defined as deep internal similarities between the small centers and angled pieces of larger centers, which tie them together to form a single unity. The houses of Alberobello, Italy, shown in Figure 9, constitute an example of simple motifs because they share a family resemblance based on the similar angles of their cone-like shapes.

The Big Wild Goose Pagoda or Tower of the Wild Goose, shown in Figure 10 and built around 652 A.D. in Hunan Province, China, is another example of a structure that exhibits ***echoes*** or motifs (square structure-arched window-roof). Alexander [4] uses the tower to illustrate ***not-separateness*** in his description of the Fifteen Properties [4 p. 230], but he also presents it as an example of deep life [4 p. 11] because the

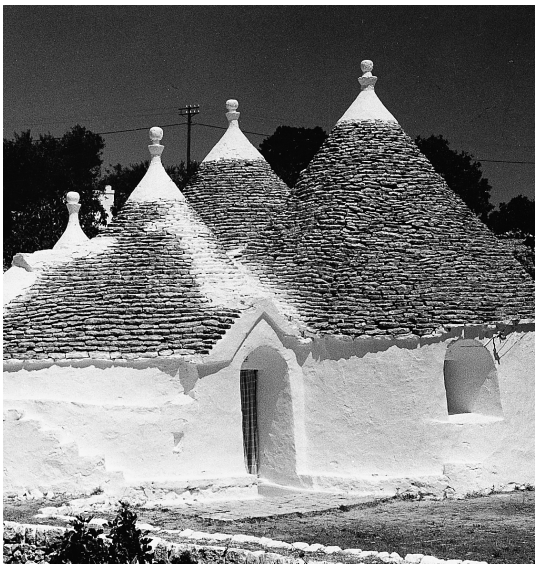

Figure 9 Conical Motifs at Alberobello, Italy [35]

collection of motifs and the supporting structure intensify the connection people feel for the tower by incorporating ***levels of scale, strong centers, boundaries, local symmetries, gradients, simplicity and inner calm,*** and ***alternating repetition*** into their arrangement.

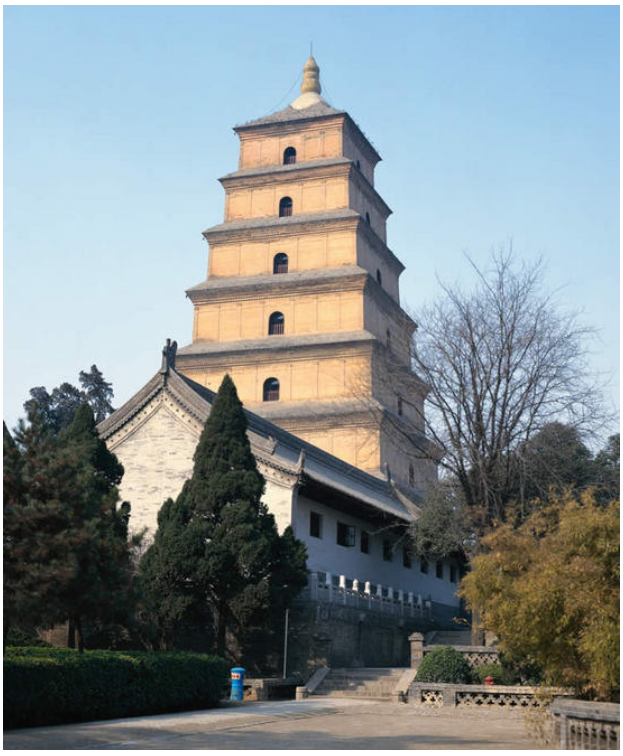

Figure 10 Tower of the Wild Goose, Hunan Province, China [36]

2.5 Modules

Modules are clusters of network elements that relate functionally or physically to each other much more strongly than they do with the other elements in the network. In Figure 10, the

tower's motifs constitute modules that exemplify parcellation – "the differential elimination of cross-interactions involving different parts of the system" [27].

Modules make scale-free networks more manageable by allowing adaptation of a given set of nodes relatively independently of the rest of the network. Modularization is a prerequisite for the adaptation of complex organisms [27] and has been implemented in the software world as object-oriented programming, as well as component-based and service-oriented architectures.

Modules are often called communities, and can be discriminated within diagrams created by network visualization tools. Figures 11 and 12 compare two types of networks, each containing 100 agents and approximately 175 links – Figure 11 is a modular network, and Figure 12 is a scale-free network without modules [37]. Notice the similarities – both show evidence of weak links in their small-world clustering, scale-free shortcuts, and nestedness. The modular network, however, contains a number of elements that are much more highly connected to elements within their own module than to elements outside their modules.

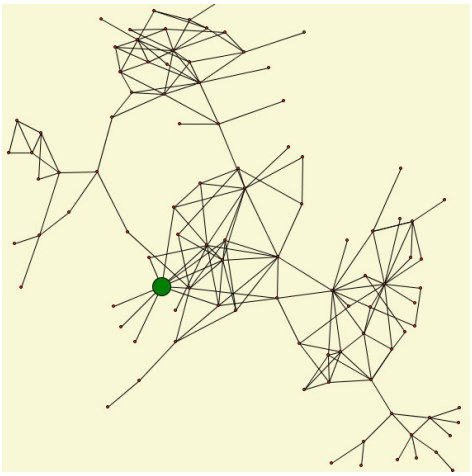

Figure 11 Modular network [37]

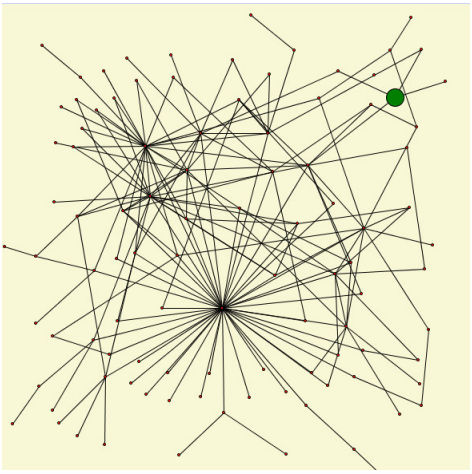

Figure 12 Small World network [37]

As noted in the section on nestedness, modularity is important in Alexandrian life because it fosters the development of ***strong centers, levels of scale***, and ***boundaries***, and supports the development of ***local symmetries, contrast***, ***gradients***, ***roughness*** and ***alternating repetition*** in some networks. One might imagine the Alhambra tile losing Alexandrian life as it scales to encompass the entire side of a building because the size of the individual **modules** would be too small relative to the size of the wall – therefore, even the best designs need to be modularized to maintain their viability as they scale.

3. WEAK LINKS

Now that some of the features of weak links have been described, their relationship to topological phase transitions over time can be explored. Because networks are often dynamic, their configuration can vary over time as they develop in complexity, undergo stress or experience resource replenishment [38, 39]. In a random network, generally found where complexity is lower and resources are plentiful, the network can support every node possessing close to the same number of connections, as shown in Figure 13. Increasing the amount of stress slightly causes structure to emerge so that the use of available resources can be optimized. The Alhambra tile shown in Figure 4 is one example of a network produced with plentiful resources and constrained to follow a pattern; therefore, it exemplifies a relatively simple scale-free network.

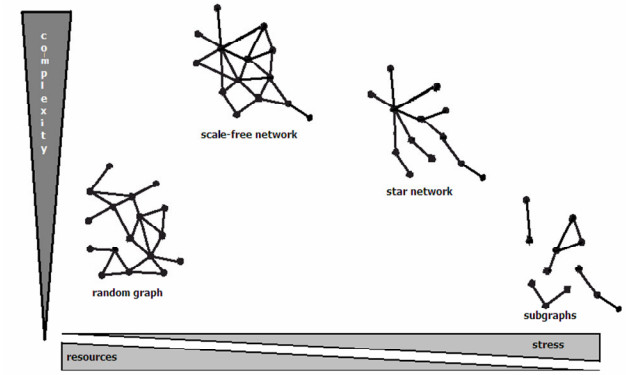

Figure 13 Network Topologies [1, p. 75]

A collection of small towns, each with its own hardware store, pharmacy, grocery, or general store is another example of a random network, as there is no advantage to traveling in one direction over another, because any direction will provide whatever services are needed. The towns, however, represent a network that can change in response to both internal and external forces. If the population shifts and some stores close, residents may choose to shop in neighboring towns, depending on distance, thus developing a scale-free network. The third topology illustrated in Figure 13 illustrates a star network, in which nearly all connections in the network pass through a central hub, which might happen when all the small stores close in response to a new super-store opening. When the star network experiences more stress, it will rip apart and no longer have a "giant" component. As an example, if oil prices make travel to the super-store too expensive, the towns in the area will no longer have their familiar old stores, but they won't be able to shop at super-store either, so new stores or online shopping may emerge as an alternative.

The reason infinite resources support a random network is that energy is required to support each of the links [40]. As stress increases and resources decrease, there is an advantage to sharing paths to the far side of the network, but there is also a benefit to the short-range clustering of the scale-free network. As resources continue to decrease, the energy required to support the extra links is not available, so a star network is formed. Finally, with even less energy, the network's giant component "dies" and decomposes into subgraphs. Thus, the giant component no longer exhibits the characteristics of Alexandrian life because its subgraphs have no relations with each other via weak links.

As the discussion above implies, weak links provide a mechanism for stabilizing a giant component as it is subjected to perturbations or disturbances due to increased complexity, decreasing resources, or increasing stress. Modules can dissipate energy from a perturbation by reorganizing their links or by detaching from the giant component. (This is also an organizational pattern [12] called "Sacrifice One Person." In this pattern, one member of a team takes on the responsibility of protecting the integrity of the rest of the group by accepting a highly disruptive task such as handling support calls or committing to a task required by the external organization but not required for the work the team is doing.)

Biological networks exhibit both redundancy (duplicate elements) and degeneracy (different elements that can reprogram themselves away from their normal function to perform the same function as a different element, depending on the context [27]) to support stabilization of their giant components. The fact that the functions performed by an element or module disabled because of a perturbation can be delegated to a different structure in the network distinguishes biological networks from technological networks, which rely almost exclusively on redundancy. This may be explained by the fact that evolutionary systems tinker with or reuse what already exists in their system to derive comparable functionality and can never be assembled by simply attaching something brand new. Enhancements to engineered or technological systems, on the other hand, can be effected by adding new and/or modifying existing components. As an example of weak links supporting biological network robustness, mutations in a single gene can often be compensated for by a different gene or set of genes that can perform the same function. However, networks of genes are subject to the same scale-free forces as other networks – i.e., mutations of some highly-connected genes, such as the p53 tumor suppressor gene that participates in programmed cell death and prevention of tumor cell proliferation, can have a profound effect (cancer), while some genes can be fully suppressed without any outward indication. [27]

Software architectures exhibit scale-freeness and small worldness when modules are integrated into the system if design principles such as maximizing cohesion, minimizing coupling, avoiding the creation of hubs with many dependencies, and minimizing development time, are applied [27]. Comparable design principles, such as optimization involving cross-network shortcuts and short paths within the neighborhood operate in the realm of cellular nets, but cellular nets are able to parlay their scale-freeness and small worldness into free homeostatis (robust stabilization mechanisms) against random failure. If researchers could discover the roots of such mechanisms, practitioners might be able to create both software development processes and software products that exhibit homeostatis.

Then, software systems may start to exhibit what Buckminster Fuller [41, p.7] calls synergy, as exemplified by chrome-nickel-steel. This alloy contradicts the belief most people have that a chain is only as strong as its weakest link – if that were true, then its tensile strength would never exceed 60,000 pounds per square inch (psi), the tensile strength of iron, which is the weakest of the three metals. Instead, the tensile strength of the alloy is 350,000 psi, which exceeds the sum of the tensile strengths of all its constituents (260,000 psi) because the structure deforms when it is stressed. Fuller's explanation is "Chains in metal do not occur as open-ended lines. In the atoms, the ends of the chains come around and fasten the ends together, endlessly, in circular actions. Because atomic circular chains are dynamic, if one link breaks, the other mends itself." The result is a metal of such strength that it can withstand temperatures generated by combustion in jet engines; hence, the goal is not just an increase in the software's robustness but enabling new kinds of software that were impossible until homeostasis emerged.

Remembering Conway's law, which states that software architecture mirrors the communication structures of its creators, if the organization's goal is to achieve what only homeostatic software can enable, the first step in this direction may be the creation of homeostatic software teams, which is essentially what Scrum can enable, with its endless circular structures around each delivery of production-ready software.

4. SCRUM

By its founders' definition, Scrum is an "iterative, incremental process for developing any product or managing any work. It produces a potentially shippable set of functionality at the end of every iteration." [3]. Scrum acts as a framework within which relevant engineering, team-building, and knowledge management principles may be applied to enable the team's meeting of its commitment to the organization. The rules are deceptively simple, just as are the design rules that produce complex software architectures, but they too are rooted in deep concepts, like weak links and complex adaptive systems, that give them great power. Disciplined attention to the application of rules and principles underlying Scrum is required to attain the benefits promised by its adherents. When that disciplined attention is applied and continually developed, both the product of the Scrum process and the process itself exhibit Alexandrian life.

The rules of Scrum involve roles, artifacts, and activities. The roles involved in Scrum are:

- ✓ Product Owner – the voice of the customer who stewards the product vision and finances, reflecting his/her constituents' needs in the prioritized product backlog
- ✓ Scrum Team – the self-organizing, cross-functional group responsible for delivering an agreed-upon segment of the highest-priority functionality within the selected time-box
- ✓ Scrum Master – the facilitator of team success responsible for focusing the team's attention on their commitments, representing management to the team and the team to management, and steward of the Scrum implementation

The artifacts of Scrum include:

- ✓ Product Backlog – the list of features, functionality, and quality attributes implicit in the product vision
- ✓ Sprint Backlog – the list of tasks required to implement the agreed-upon segment of the Product Backlog that will be delivered within the next time-box (called a Sprint)
- ✓ Sprint Burndown Chart – a graph showing the rate at which the effort required to reach the Sprint goal (which reflects the agreed-upon product increment) is declining. Time remaining must hit zero on the last day of the Sprint. Figure 14 illustrates burndown for a standard 30-day Sprint. (Note that the actual burndown does not decline linearly but varies based on new information, negotiated decisions, and revised designs.)

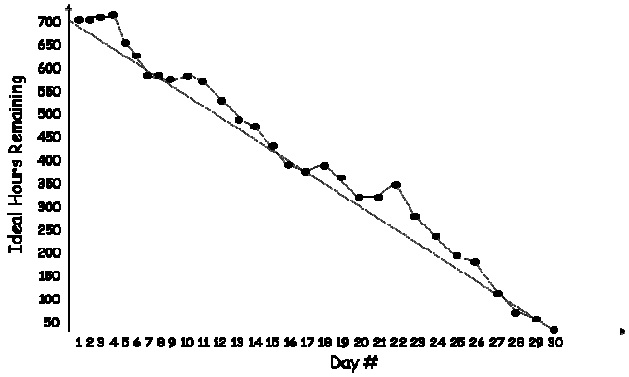

Figure 14 Sprint Burndown Chart

The activities of Scrum include:

- ✓ Sprint Planning Meeting
  - Part I - Review the prioritized product backlog, consider what work will be possible during the next Sprint, and define a Sprint goal around which to organize the work.
  - Part II - Define and estimate the tasks entailed by the work, and reconfirm its feasibility. If no longer feasible, renegotiate the work with the Product Owner.
- ✓ Sprint – both the timebox and the development activities inside the timebox that will lead to the delivery of production quality code that realizes the Sprint goal.
- ✓ Scrum Meeting – the daily 15-minute synchronization period during which team members answer the questions:
  - What did you do since the last Scrum Meeting?
  - What will you do before the next Scrum Meeting?
  - What is standing in your way?
- ✓ Sprint Review
  - Demo – the team's opportunity to tell the story of the production-quality increment constructed during the Sprint and the product owner's opportunity to review, accept, and/or adapt its functionality
  - Retrospective – a period of reflection during which the team and invited stakeholders build community, identify best practices, and suggest improvement experiments – a time to "discover, share, and pass along the learning from experience—something we also call 'wisdom'" [42].

When all of these elements come together, the resulting structure is often captured in a diagram like Figure 15, which identifies the relationships between the activities and the artifacts [43]. It also highlights some of the characteristics of Scrum:

- ✓ visibility into the process and the Product Owner's priorities
- ✓ feedback on at least a daily basis (more if the team is collocated and taking advantage of osmotic communication – learning by hearing and seeing ambient information)
- ✓ continuous adaptation of the work distribution and approach within the team
- ✓ continuous adaptation of the product backlog by any stakeholder (subject to the Product Owner's prioritization before the next Sprint planning meeting)

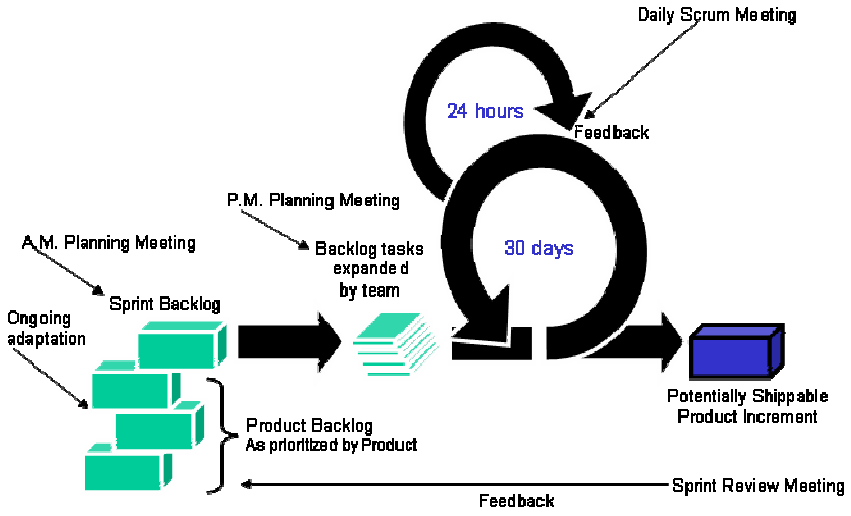

Figure 15 Scrum flow [43]

When a team first adopts Scrum, it executes the entire lifecycle of a small part of the desired system – all requirements are identified, analyzed, and reflected in the product design, and the design is reified by the team's code, supported by documentation (e.g., user manuals, help files), and tested against user acceptance criteria. Although this description sounds sequential, everything happens at once as the team optimizes itself like a surgical team to deliver working software that meets the Sprint goal within the Sprint timebox.

Team members switch roles to the extent possible in hopes of avoiding bottlenecks, but this approach to Scrum (called Scrum Type A in [44] and illustrated in Figure 16) results in "dead" time while the team reorganizes for the next Sprint. This dead time prompted an evolution into Scrum Type B. In Scrum Type B (see Figure 16), the requirements for the subsequent Sprint are developed in the current Sprint, so that only items that already have buy-in from the organization are on the product backlog at the Sprint planning meeting. This removes the development bottleneck and can result in the production of more value than the organization can handle. Finally, a single experienced Scrum team executing appropriate engineering practices can work on simultaneous Sprints of one week, one month, and three months if they are supported by a well-designed product architecture, automated testing and automated presentation of the integrated Sprint backlogs by team member. This version of Scrum is an adaptation of Takeuchi and Nonaka's original "overlapping development phases" [45] in that multiple smaller Sprints can be completed before their containing Sprint is completed. This approach resulted in 45 releases per year for PatientKeeper, an organization that currently employs it [3].

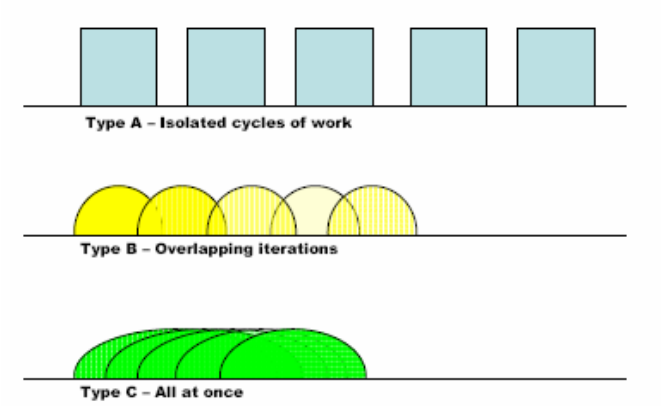

Figure 16 Three implementation styles of Scrum [44]

In order to support Scrum Type C, the product backlog must be prioritized and available to the team at very short intervals. This means that the Product Owner continually manages the list as it adapts to changes in business conditions, team capabilities, technological factors and the capability context resulting from the most recent product releases. To maintain the priorities of the product backlog, the Product Owner must ensure that the organization's executives implement a process analogous to the Scrum team's process, but with different inputs and outputs, as shown in Figure 17, the diagram of Meta-Scrum. Note that the team's flow takes the output of the management flow as its input and returns the team's new functionality to the executive team as revised process / technology needs and context.

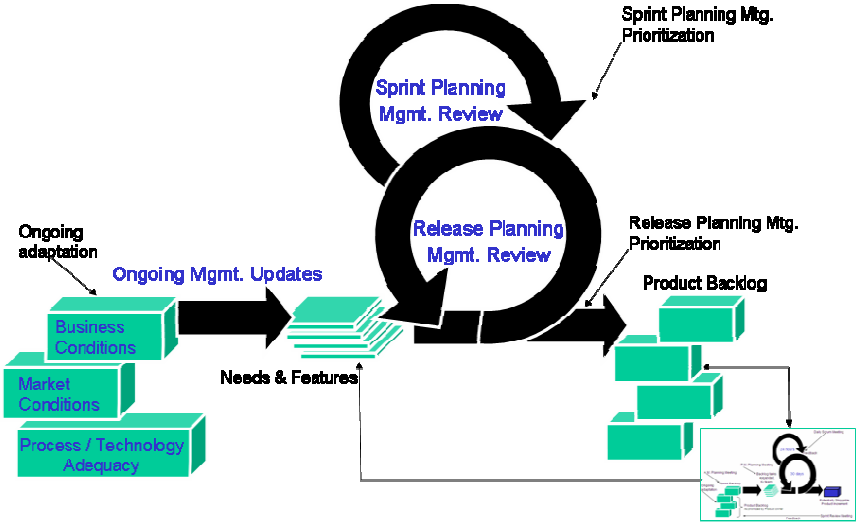

Figure 17 Meta-Scrum for the Executive Team

Given the size of the smallest Sprint in Scrum Type C, the Meta-Scrum meeting must occur weekly in order to ensure that the next week's work will reflect the organization's highest priorities and to accommodate perturbations such as a customer offering to buy $1 million worth of product if a certain feature can be delivered in a short timeframe. If that particular feature had been a low-priority feature or did not exist on the product backlog, it would have to replace a feature that had been high-priority. Therefore, the product of a Meta-Scrum meeting may be a list of actions that need to be taken in order to take advantage of the reported opportunity. The resulting tasks may include notifying customers who were expecting the high-priority feature in a certain release of the new timetable, estimating the tasks required, extracting related items from the product backlog at the next planning meeting, and possibly starting requirements work immediately, thus causing premature

termination for some teams. Premature termination involves breaking links that had previously formed in favor of forming new and potentially more valuable links related to the new information, one of the many parallels between Scrum, Alexandrian life and weak links to be explored in the next section.

5. WEAK LINKS AND ALEXANDRIAN LIFE IN SCRUM

Note: Where weak link concepts appear, they are formatted in **bold**.

This section looks at Scrum from a weak links perspective, seeking evidence of Alexandrian life and the stabilizing characteristics of small worldness, nestedness, and scale-freeness. It identifies and considers the purpose of motifs and modules in Scrum's practices, structures, and artifacts, examining each in terms of its contribution to the stability of a highly functional Scrum implementation.

5.1 Sprints

Scrum defines its dynamic structure in terms of the Sprint, which opens with planning and estimating meetings, closes with a demo and retrospective, and allows execution to take place within the ***boundaries*** these meetings impose. Each Sprint that reprises this structure over an equivalent time scale is a temporal **module** with a ***strong center*** that ***echoes*** its neighbors with ***alternating repetition***. The strength of the center should increase as the team matures in terms of its ability to self-organize around the Sprint goal, deliver a production-quality increment as the Sprint's output, and improve its performance in the next sprint.

Figure 18 illustrates a **motif** echoed by every Sprint. The execution node is represented in the diagram by the star (with its points representing closely collaborating team members and its center capturing their shared links to information and tasks). The left side of the star denotes input, with one element representing what is taken from the product backlog and the other signifying the estimates of how much time that work will take. The right side denotes output from the Sprint, with one element symbolizing what was put onto the product backlog as a result of the demo and retrospective and the other corresponding to the realization of the Sprint goal. The top elements represent each team's Scrum Master, who acts as the ***contrasting boundary*** between the team and the organization, removing impediments, bringing external resources close enough that the teams can take advantage of those resources, and channeling information from outside the team as required.

Figure 18 Sprint Motif for 3 teams using Scrum Type A

This figure looks very much like Scrum Type A in Figure 16. There are spaces between the Sprints during which teams prepare for the next Sprint. Alexander identifies ***not-separateness*** and ***good shape*** as characteristics of life, which suggests that Scrum Type A is not the final goal of organizations adopting Scrum. It seems to be a stop-gap implementation that allows the team to discover how they can reinforce each Sprint's ***strong center*** by adjusting their commitments to the Sprint length, adapt to its ***boundaries*** by strengthening their community's communication and feedback skills, and acclimatize to the cyclical rhythm Alexander calls ***alternating repetition***. To the extent that the organization has adapted to Scrum, they will develop a greater sense of wholeness by filling the gaps and overlapping the Sprints in such a way that the ***levels of scale*** corresponding to Scrum Type C emerge, as illustrated in Figure 19. Note that even the smallest Sprints exhibit the same Sprint **motif**.

Figure 19 Scrum Type C Motifs and Levels of Scale in a 3-month release, 1-month patch, and 1-week bug fix

When the increment to be delivered requires a number of teams to work on the same product simultaneously, Scrum recommends a Scrum of Scrums approach, whereby each Scrum Master meets with the other teams' Scrum Masters to synchronize their teams' efforts. From these meetings, each Scrum Master can gain information that affects the team they serve, report it to the team, and determine with the team what impact that information will have on their ability to meet the Sprint goal. Based on their decision, the Scrum Master may call a meeting with the Product Owner to terminate the Sprint, or the team may redesign the work to meet the Sprint goal.

The Scrum of Scrums might be represesnted as shown in Figure 20. Note that its purpose is to provide the backbone for the integrated teams or **modules** to communicate with each other. Note also that the Scrum of Scrums does not effect top-down **parcellation** to modularize the teams for control, but is **integrative**, bringing each team's progress into the **top-level network** for coordination through dissemination of information across the whole system.

Figure 20 Scrum of Scrums with Scrum Teams

Because the team can self-organize according to its distribution of skills, the center of each star can adapt to the resources available, the complexity of the problem, and the level of stress

as needed. Figure 13 is reproduced in Figure 21 to remind the reader that in the absence of stress, with a full complement of resources and a relatively simple problem, a **random network** organization will prevail. This might correspond to a team with highly overlapping skillsets such that any team member has the skills and information to undertake any task, thus avoiding all bottlenecks. It might also correspond to a new team within which the team members do not know the distribution of skills. As specialization within the team or complexity of the tasks increases stress, or if the team becomes smaller, **scale-freeness** and **small worldness** might emerge to take advantage of those in the group with specialized skills required by multiple tasks. Because of clustering, this arrangement may allow several concurrent tasks to take advantage of the same resource so that the team becomes more productive. With greater stress, **modularity** helps to make a large, complex problem more manageable and still takes advantage of the signal-propagation speed and synchronizability benefits of **scale-freeness**. As Sprint goals change across time, Scrum teams may modify their communication patterns to optimize the resources available against the work required, causing phase changes among the network topologies.

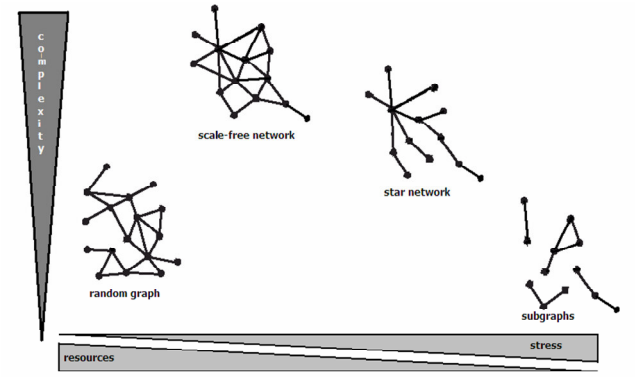

Figure 21 Network Topologies [1, p. 75]

In those methodologies where unrealistic dates are imposed on the team, the higher level of stress might cause the team to adopt a star configuration, possibly calling for a "hero" to emerge and save the day. If the hero fails, then the project might decompose into subunits, resulting in the finger-pointing that often accompanies failed projects.

To avoid this situation, the team must understand the Product Owner's expectations well enough to make a valid commitment leading to ***positive space*** and ***good shape***. The abililty of the team to commit to a reasonable workload during the course of a Sprint is dependent on the quality of the product backlog. If the product backlog has only items that exceed the team's ability to deliver within a single Sprint or Priority 1 items, they will fall into the same trap as non-Scrum teams. Thus, the quality of the planning that goes into the product backlog and the distribution of attributes (***levels of scale*** and **scale-freeness**) across the product backlog constrain the Alexandrian life of the Sprint.

5.2 Product Backlog and Planning

The product backlog captures the work required to realize the Product Owner's vision of the product. Thus, it contains large-grained items that have been envisioned but are too far in the future to describe in detail, medium-grained items that should be delivered within the next 6 months to a year, and smaller items that are visible in high relief due to the immediacy of the need for them or their importance to a smooth-functioning organization. These levels of detail correspond to the **nestedness** of the product, its releases, and the increments delivered in each Sprint, thereby contributing to the **modularity** of both the representation of the work in the product backlog and the work itself as implemented in the Sprints.

Scrum software development teams within traditional organizations often suffer due to the standard budgeting cycle, which depends on commitments to highly speculative estimates. Therefore, Scrum teams function best in an environment that integrates input from all perspectives, including marketing, sales, IT, and corporate executives into the planning process. In this type of an organization, the executive team operates on the Sprint schedule as well, as described by the Meta-Scrum flow in Figure 17. The executive team members accept emerging business and market conditions, along with process and technical competency as input into their Meta-Scrum meetings, and update the product backlog by adding new items, eliminating obsolete items, consolidating some items, and breaking others into smaller elements. The revised product backlog produced as output from these meetings is presented to the teams at the next Sprint planning meeting, and the teams commit to the highest priority items they can deliver. Note that in Figure 22, the Scrum Master facilitates the Sprint Planning meeting but does not interpret the product backlog for the team. Thus, the **modularity** supported by the Scrum Master interface that facilitates execution during the actual Sprint turns into **small worldness** between the team and the organization again during Sprint planning and Sprint review, reflecting ***alternating repetition*** and another example of ***echoes***.

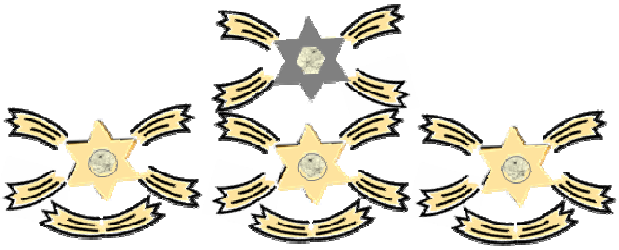

Figure 22 Sprint Planning Meeting with Product Backlog Updated by Meta-Scrum

As illustrated in Figure 15, Scrum requires a planning meeting before the start of each iteration so that the team can commit to delivering the highest priority items from the product backlog. These meetings may serve the same function as Escher's signature in that they allow the ***boundary*** between the product backlog items and the team's fine-grained tasks for realizing those items to be negotiated rather than assumed. Planning meetings provide a venue for links to information, activities, people and goals to be created, shuffled, or modified in response to forces that have arisen since the last planning meeting or in response to improved understanding of the organization's needs. These links are the source of ***deep interlock and ambiguity*** across both time and space. In the space dimension, the links serve as shortcuts that bind the team **module** and the

organizational **modules** the stakeholders represent. The **nesting** between the team and the rest of the organization will be discussed further in the Scrum Structure section. In the time dimension, the links are the long-range relationships between the vision captured in the product backlog and the tasks the team will perform during the next Sprint. The **nesting** between the product backlog and Sprint backlog fosters ***levels of scale*** across the team's delivery horizon, the immediacy of the organization's need, and the clarity of the organization's perception of its needs.

In addition to selecting items from the product backlog into the Sprint backlog at the planning meeting, the team and product owner select a Sprint goal that reflects the reason for building the chosen work. This goal represents the meaning or significance to the organization of the production-ready software that the team has committed to delivering during the Sprint. The Sprint goal is one of several sub-goals that must be met in order to deliver the value reflected in the overarching release goal. The release goal itself reflects larger product goals, which in turn reflect business strategies for the entire organization. Thus, the goals developed at planning meetings provide a further reflection of **nestedness** and multiple ***levels of scale***. They also allow the team to commit at the goal level, which avoids premature commitment to specific deliverables and allows the team to redesign as needed during the sprint. **Weak**ly **link**ing deliverables and goals is desirable because the same goal can be reached through multiple design paths, and the team and Product Owner should be free to negotiate the most appropriate path, given the priorities, time and skillsets available within the Sprint. This ability to redesign or renegotiate what a goal entails dissipates perturbations due to new knowledge, missed dependencies, misunderstandings, etc.

The finest-grained planning takes place within the daily Scrum meetings, within which each team member commits to the day's work. Once again, the team members can reshuffle the strength of their links so that someone who was working independently may volunteer to form a stronger link with someone who needs help. An observer who expects rigid adherence to the plan may interpret this "deviation" as ***roughness***, but it is evidence of judgment being applied to maximize value delivery in the context of .Scrum Structure.

5.3 Scrum Structure

Within the team, members pass information and make decisions with a high degree of clustering, while the Scrum Master removes barriers that keep them from accessing sources of information external to the team. James Coplien [46] studied communication patterns across many projects in large telecommunications organizations and found several characteristic patterns. These patterns bear a striking resemblance to those that show small worlds and random networks, as illustrated in Figure 23 and 24. Cain and Coplien used these words to describe their findings [46]:

> In a mature process, one that has evolved over a decade in a relatively constant domain, load balancing takes place. Information flow is broadly distributed across process roles (Figure 23). The network of communication between roles is not fully distributed, but the total distance between two roles is rarely more than two hops, and there is a good mapping between direct, strong coupling and ''need-to-know'' relationships. We also know that these processes have largely distributed decision-making loci and are not centrally orchestrated. In fact, the Manager role in these processes is usually at the outer edge, with tighter coupling to resource organizations (e.g., the computer center) than to the principals in the process! There is likewise no centralized decision making in a ''mess,'' and managers are likewise at the periphery of the process. However, a ''mess'' process tends to be a more fully connected graph than a ''mature'' process, indicating that the communication overhead on each role is unfocused and intense. (Figure 24)

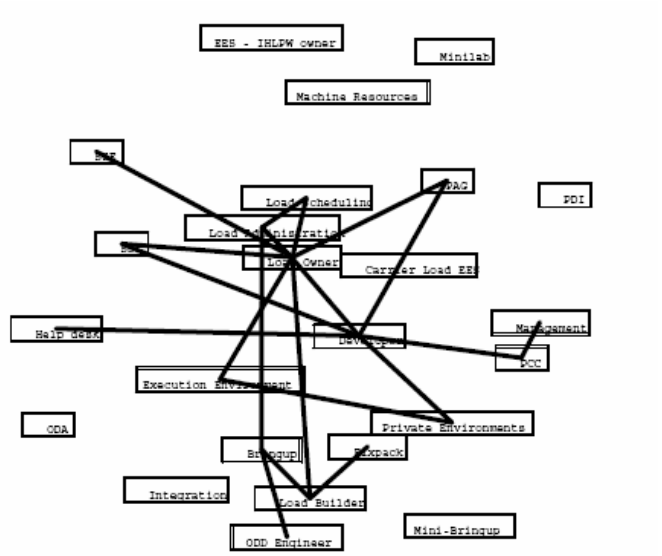

Figure 23 A Mature Process

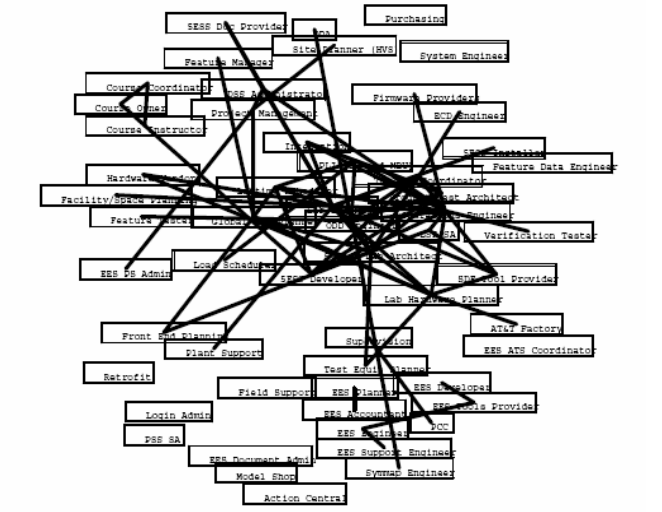

Figure 24 A Messy Process

Note that the mature process in Figure 23, which is found in product-oriented groups, closely resembles Figure 21's **scale-free** network topology, while Figure 24, which is found in more service-oriented groups, resembles its **random** network with unfocused and intense communications requiring a great expenditure of energy and resources. This is consistent with findings that the highest and lowest socioeconomic classes, and social networks under stress do not exhibit scale-free relationships – they tend to interact only with those like themselves. [47, 48] These are very small networks, so these ideas are not conclusive, but the phenomenon warrants attention to determine whether or not weak links are at work here.

Scrum capitalizes on both of Coplien's structures as it moves from Scrum A to Scrum B to Scrum C. When Scrum A is implemented in an organization, the team self-organizes around tasks from all the disciplines involved in creating an increment of production-quality software. Scrum B separates out the requirements-gathering tasks into their own sprint or module, as preparation for the development tasks that dominate the implementation sprint – analysts participate in the latter, but their activities dominate in the preparatory sprint. This type of modularization resembles parcellation or top-down modularization. In Scrum C, the number of simultaneous sprints supported by a single individual increases, and **modules** corresponding to groups of tasks from each of the simultaneous sprints integrate into each individual's daily schedule, reflecting the weekly, monthly, and quarterly sprint **modules**. This becomes visible as ***levels of scale*** that ***echo*** each other in the backlogs and each team member's daily schedule.

Regardless of the organization's implementation of Scrum, deformations appear within all Scrum teams as team members encounter tasks that require skills beyond their own or schedules that preclude a particular design decision. Under these conditions, Scrum encourages disclosure of the need at the team's daily Scrum meetings so that multi-disciplinary members can volunteer assistance or information to each other and avoid bottlenecks. When the team unexpectedly encounters a task that requires skill beyond all of its members, the Scrum Master's job is to find someone with that skill to train or mentor the team. This extends the team's access to skills and solutions via the Scrum Master's shortcut. In nature, these frequent short searches interspersed with less frequent intermediate searches and rare searches of distant locations are called Levy flights. The behavior is characteristic of ants, bumble bees, deers, jackals, and monkeys looking for food because it balances cost-efficiency with novel search locations that ensure energy is not spent duplicating previous searches [1].

If no solution can be found within the team, the Scrum Master notifies the Product Owner of the problem, and the stress is removed by redesign, postponement of the item to a later sprint, or premature termination of the sprint. Though termination is extreme, it dissipates stress on the team, allowing it to regain its former state and regroup for the next Sprint.

Because the Scrum Master acts as a buffer between the team and the rest of the organization, dissipating the impact of external perturbations, the role often entails managing promises made by the Product Owner and the rest of the organization. As such, the Scrum Master and Product Owner roles provide ***contrast*** and ***boundaries*** around their constituents. They also support ***deep interlock and ambiguity*** but this feature will be discussed in greater detail below because it is so profoundly embedded in Scrum's practices, artifacts, and structure.

5.4 Deliverables

Within each Sprint, the team produces an increment of production-quality code having strong links inside the increment and weaker links between preceding and succeeding increments. It generally takes several Sprints before the increments are released to production, which leads to the **nesting** of several increments inside each release, and multiple releases constituting the complete product. In terms of Alexandrian life, products built by Scrum teams exhibit ***levels of scale*** (in-process work within the Sprint, Sprint increment, release, product), strong centers that include working code and whatever else is required to meet the team's doneness criteria, and ***gradients*** of product completion that expand over time in the production environment. Because Scrum does not specify engineering practices, it is difficult to predict which characteristics of weak links or Alexandrian life will be exhibited by its output, but one might expect the product to mirror the process that produced it.

Following Conway's law, a highly functional Scrum team applying the agile values of communication and feedback should produce high-quality software for deployment into the production environment. Applying the simplicity value of extreme programming to an object-oriented paradigm should produce objects with ***strong centers*** and ***boundaries*** that ***echo*** the team's accepted class hierarchy or patterns of composition. Simplicity requires producing only the objects or components needed to meet the customer's needs as expressed in their acceptance tests, meaning that the software, its users, and its environment should exhibit ***positive space*** and ***not-separateness*** when all increments have been delivered. Interfaces are always a challenge when multiple teams must connect their work products to each other or to existing systems, forcing the team to validate assumptions as tests that connect to test systems or simulated systems with analogous interfaces. Once built, the adapters or components that allow these tests to pass exhibit ***contrast*** such that the boundary or interface between the systems on either side becomes a ***strong center*** in itself, suggesting that creating interface subteams or cross-team pairs would be an appropriate practice at the product ***boundaries***. While time constraints may sometimes force the team to accept ***roughness*** due to technical debt, refactoring should produce ***good shape*** before the end-of-Sprint demo.

5.5 Demo and Retrospective

Like its planning meeting, Scrum's defined closure activities provide opportunities to break links between iterations deliberately, thereby dissipating perturbations due to forces such as schedule slippage, revised priorities, or externally imposed constraints. Also like the planning meeting, the demo and retrospective provide an opportunity to connect with stakeholders, goals, and relevant information from outside the team. While the demo examines the product from the perspective of the stakeholders' needs, the retrospective examines the process with the goal of providing the team with the opportunity to continuously improve their ability to deliver.

Once the team demonstrates its "production-ready" software, the customer can choose to break with the release schedule in order to take advantage of unanticipated value inherent in the features already created by the team. **Weak links** between value and schedule allow the organization to prioritize pending work based on the anticipated release schedule, but capitalize on emergent utility when an unscheduled release needs to be arranged. This may lead to ***roughness*** when the release landscape is observed, but the newly generated value may lead to a greater harmony or ***not-separateness*** among the organization, the team, and the delivered system.

Within the team, the retrospective provides an opportunity to link the execution and outcome of the Sprint with lessons to be applied in future Sprints. Regular examination enables the identification of **motifs** or ***echoes*** that support success or failure. The latter can be used as warning signs that trouble may be ahead, prompting vigilance even if the connection cannot be articulated explicitly. As mentioned in Section 5.1, regular retrospectives provide ***alternating repetition*** of experiences designed to strengthen the team members' links to each other by encouraging team-building, validating each other's perceptions, and acknowledging contributions made by team members who play a supporting but critical role in the team's success. Like the watercarrier, who did not hold a position of designated power but delivered the water essential to the tribe's survival, these team members are cited for exemplifying "the essential nature of all jobs, our interdependence, the identity of ownership and participation, the servanthood of leadership, the authenticity of each individual" [49, p. 62]. Their example teaches new members about Scrum's culture by demonstrating awareness, listening, healing, empathy, and commitment to the growth of others. Retrospectives also provide a time to celebrate successes, grieve fatal perturbations that destroy the team's giant component, and recognize members' skill advancement. These ***contrasting boundaries*** become ***strong centers*** themselves as they evoke ***not-separateness*** between Sprints by interlocking past experience with commitments to the future and generating new insights that open pathways to greater capability – ***deep interlock and ambiguity.***

6. DEEP INTERLOCK AND AMBIGUITY

Deep interlock and ambiguity abound in Scrum to an extent that is unique among software development methodologies. The weak link characteristics of **nestedness**, **motifs**, and **modularity** are part of many software development methodologies – consider the standard phases of requirements gathering, analysis, design, coding, testing, and operations, where each provides input to the next, and every project replicates the motif. What makes Scrum stand out is the frequency of opportunities to create weak links, the variability of the link lengths created during those opportunities, and the breadth of dimensions across which those links extend. ***Deep interlock and ambiguity*** in the boundaries between ***strong centers*** or **modules** comprising goals, information, activities, and groups of people with different perspectives at different levels of the hierarchy give highly functional Scrum implementations a sense of ***not-separateness*** from their organizational context, their history, their output, and their core attributes.

Among Scrum's core attributes are visibility, inspection and adaptation, all of which foster the development of shortcuts between the business and the Scrum team. These core attributes find expression in Scrum's frequent opportunities for creating links or restructuring existing links of varying strengths, lengths, and duration, including:

- ✓ planning meetings where the Product Owner presents stakeholders' priorities in the form of the product backlog to the team
  - o links with business and external stakeholders to negotiate the highest value that can be delivered in the next Sprint
  - o links to the nested business goals reflected in the product backlog from which the Sprint goal will be drawn
  - o strong links to Sprint goal negotiated with stakeholders
  - o parcellation of product backlog items into Sprint backlog items and tasks (modularization)
  - o links to history or analogous projects to determine estimates for tasks
  - o selective formation of links between members and tasks via self-organization
- ✓ daily Scrum meetings where the team synchronizes its members' activities in the presence of all interested stakeholders
  - o shortcuts to the Sprint backlog when reporting progress and committing to next day's activities
  - o shortcuts from outside the team to the team's work for monitoring progress
  - o emergent links to teammates whose impediments can be removed by team members
  - o commissioning of Scrum Master to remove newly-identified obstacles by linking to non-team members
  - o emergent links from team to Product Owner via Scrum Master when the Sprint goal is in jeopardy so that the team's options can be discussed
- ✓ demos where the team re-engages with its stakeholders to connect its current increment to the future they envision
  - o shortcuts to business and external stakeholders when determining the correspondence betwee the delivered increment and the Sprint and release goals
  - o breaking of links to requested features that do not represent progress toward product vision
  - o formation of links to new features suggested by the product's current state
  - o destruction of links between product backlog and schedule when accrued value warrants disruption of the schedule
- ✓ retrospectives where team members recommit to continuous improvement and to the team as a community
  - o reflective links with the activities and events of the past Sprint with the purpose of evaluating their value and impact
  - o feed-forward loops with experiments designed by the team to improve its process
  - o rites and ritual activities that link each Sprint to its predecessors and provide closure to the team
- ✓ Scrum Master's transient strong links to stakeholders who can remove team's impediments by providing needed resources, information, privileges, etc.

- ✓ Product Owner's spectrum of link strengths and lengths to other stakeholders whose needs must be reflected in the product backlog
- ✓ information radiators that foster osmotic communication with anyone passing by, regardless of their involvement with the project
- ✓ engineering practices featuring
  - small releases – 2-6 weeks to production-ready code to accommodate varying link strengths to item priority
  - simple design – no premature generalization to avoid strong links to an over-engineered solution
  - refactoring – evolutionary tinkering to break links with worthless code and add or abstract what's needed
  - continuous integration – surfacing integration issues immediately to allow creation, repair and dissolution of links between components
  - customer acceptance tests – test-driven requirements to create strong links between strong business centers and the boundaries of components that support them
  - test-driven development – tests written before code to allow link evolution (ambiguity) inside the boundary defined by the tests but remain connected to the original purpose (deep interlock) on the outside
  - coding standards – team-defined standards for naming, configuration control, design, etc. to support weakly linked motifs/echoes in both process and deliverables from past, current, and future releases
  - whole team – customer, QA, architecture, developers, BAs, as needed to support real-time formation and dissolution of information and interpersonal links
  - sustainable pace – workweek defined by the team to meet corporate standards and foster viable links to co-workers, information, and output, as well as to family and community

7. IMPLICATIONS

Because Scrum offers so many opportunities for deep interlock and ambiguity, unsuccessful Scrum teams may find themselves starting their search for process improvement at these boundaries. Investigation may identify lack of stakeholder participation from a scale-free distribution of distances up the organizational hierarchy (one executive sponsor and many subject matter experts), or periods of stakeholder engagement with the team (many short engagements with a few long engagements). Although support for executive involvement has always been identified as a success factor, the analogies with weak links may provide new models for justifying that support and possibly new ways to compensate for a lack thereof. The improvement of success rates in the Standish CHAOS reports from 1994 to 2004 were partially attributed to agile requirements process [50], which may be more rigorously defined using notions of scale-free distributions, small worlds, and modularity. The emphasis on clean interfaces between modules may be re-examined in light of Alexander's notion of contrast and boundaries appropriate to the size of the strong centers they encapsulate, possibly causing an evolution to "smart boundaries" for both teams and the software they build. The purpose of this paper was to raise awareness of the relationships among weak links, Alexandrian life and Scrum so that interested parties could construct new ways of improving their Scrum implementations and share their results with the community. As an example, process improvements may be more effective if they enhance ***deep interlock and ambiguity*** where ambiguity refers to augmenting the organization or team's choices relative to optimizing its resources for the current context. The enhancement mechanisms may become the process design patterns of future Scrum teams. Enhancing deep interlock may provide scale-free access to all affected parties so that the entire system reacts synchronously, giving rise to a whole new branch of organizational patterns or a deeper understanding of existing ones. These are just two ideas, meant to seed the field of possibilities for a new community of discourse around these topics.

8. CONCLUSIONS

This paper has illustrated key aspects of weak links, examining Scrum in terms of those aspects and suggesting that they express themselves as characteristics of Christopher Alexander's life. This section is not intended to be a conclusion but an invitation to further investigations in this area, with the ultimate goal of redefining what makes an effective software development process in the context of the socio-economic system it supports. Scrum supporters have said for years that their process has its practical basis in empirical process control but is informed by such abstract theoretical disciplines as artificial life and complex adaptive systems. Its success may rest, however, not in its similarity to these areas, but in a shared ancestor grounded in weak links. Mining other disciplines for diagnostic and measurement techniques may provide agile software development, and Scrum in particular, with a completely new set of metrics that generate far greater insight into the connections between IT, management, and business than anything currently defined.

9. ACKNOWLEDGMENTS

This paper would not have been possible without the steadfast encouragement and judicious counsel of my shepherd and mentor, Linda Rising. The valuable criticism of Takashi Abi, Miguel Carvalhais, Christian Crumlish, Christian Kohls, Kathy Larson, Ricardo Lopez, Mary Lynn Manns,Lubor Sesera, Fran Trees, Steve Wingo, all reviewers at PLoP'08 in Nashville, TN, reshaped the paper and led to valuable insights that had previously escaped the author. Finally, I owe a great debt of gratitude to Curt McNamara and his IEEE systems book club in the Twin Cities, which led to my discovery of weak links in the first place. Because the entire paper was rewritten based on PLoP feedback, any errors or omissions are solely the responsibility of the author.

10. REFERENCES

[1] Csermely, P. 2006. *Weak Links: Stabilizers of Complex Systems from Proteins to Social Networks*. Springer Frontiers Collection, New York, NY.

[2] Beedle, M. and Schwaber, K. 2001. *Agile Software Development with Scrum*. Prentice Hall, Upper Saddle River, NJ.

[3] Sutherland, J. Agile Project Management With SCRUM: Theory and Practice. Presented at OTUG Twin Cities on June 3, 2005. Also at: http://jeffsutherland.com/papers/OTUG2003/Scrum_Theory_files/v3_document.htm. Retrieved 7/19/2008.

[4] Alexander, C. 2001. *The Phenomenon of Life: The Nature of Order, Book 1.* Oxford University Press. New York, NY.

[5] Trifork A/S. http://www.trifork.com/default.asp?Action=Details&Item=17. Retrieved 12/15/2008

[6] Smith, M., Kulkarni, S. and Pawson, T. FF domains of CA150 bind transcription and splicing factors through multiple weak interactions. *Molecular and Cellular Biology*, 24, (Nov 2004), 9274-9285.

[7] Swanson, K., Kang, R., Stamenova, S., Hicke, L. and Radhakrishnan, I. Solution structure of Vps27 UIM-ubiquitin complex important for endosomal sorting and receptor downregulation. *EMBO Journal*, 22 (Sep. 15 2003), 4597-4606.

[8] Granovetter, M. The Strength of Weak Ties. *American Journal of Sociology,* 78, 6, (May 1973), 1360-1380.

[9] Sole, R., Ferrer Cancho, R., Montoya, J. and Valverde, S. Selection, tinkering and emergence in complex networks. *Complexity*, 8, (Jan 21 2003), 20-33.

[10] Barabási, A. and Albert, R. Emergence of scaling in random networks, *Science,* 286, (Oct. 15 1999). 509-512.

[11] Berlow, E. L. Strong Effects of Weak Interactions in Ecological Communities in Nature. 422, (Mar 29 1999), 633-637.

[12] Coplien, J. and Harrison, N. 2004. *Organizational Patterns of Agile Software Development*), Addison-Wesley, Boston, MA.

[13] Weinberg, G. 1998. The Psychology of Computer Programming. Dorset House, New York, NY.

[14] Cockburn, A. Good Old Advice, *CrossTalk: The Journal of Defense Software Engineering*. 21, 8, (Aug 2008), 7-10.

[15] Guare, J. 19 90.*Six Degrees of Separation: A Play* Vintage Books, New York, NY.

[16] Pareto, V. The New Theories of Economics*, Journal of Political Economics*, vol. 5, (1897), 485-502.

[17] Buschmann, F., Henney, K., and Schmidt, D. 2007. *Pattern Oriented Software Architecture Volume 5: On Patterns and Pattern Languages*. John Wiley & Sons, Hoboken, NJ.

[18] Alexander, C. 2004. *Process of Creating Life: The Nature of Order, Book 2.* Oxford University Press. New York, NY.

[19] Alexander, C. 2005. *Vision of a Living World: The Nature of Order, Book 3.* Oxford University Press. New York, NY.

[20] Alexander, C. 2004. *The Luminous Ground: The Nature of Order, Book 4.* Oxford University Press. New York, NY.

[21] Watts, D.; Strogatz, S. Collective dynamics of 'small-world' networks. *Nature,* 393, 6684 (June 4 1998), 409–10. Also at http://www.sociology.columbia.edu/pdf-files/watts08.pdf Retrieved 7/14/2008.

[22] Cockburn, A. 2001. *Agile Software Development* (The Agile Software Development Series) Addison-Wesley Professional. Boston, MA.

[23] Conway, M. How do Committees Invent, *Datamation*, **14**, 5, (April, 1968), 28-31. Also at: http://www.melconway.com/research/committees.html. Retrieved 11/30/2008.

[24] Whitinger, D. Survivors! (June 18, 2008) Trees and plants that will remain after the storm passes. At: http://davesgarden.com/guides/articles/view/1268/. Retrieved 11/30/2008.

[25] Sanderson, B. A pattern seen at Alhambra, Spain. in Dror Bar-Natan's Image Gallery at http://www.math.toronto.edu/~drorbn/Gallery/Symmetry/Tilings/333/Alhambra.html. Retrieved 8/11/2008.

[26] Google Maps. Satellite view of the Renaissance Center in Detroit. http://maps.google.com/maps?hl=en&tab=wl, Renaissance Center, Detroit, MI. Retrieved 8/11/2008.

[27] Sole, R., Ferrer Cancho, R., Montoya, J. and Valverde, S. Selection, tinkering and emergence in complex networks. *Complexity*, 8, 1, (Jan 21 2003) 20-33.

[28] Hofstadter, D. 1979. *Gödel, Escher and Bach: an Eternal Golden Braid*, Vintage Books, New York: Random House, Inc.

[29] Gerten-Jackson, C. http://sunsite.icm.edu.pl/cjackson/escher/p-escher13.htm. Retrieved 12/15/2008.

[30] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software* (Addison-Wesley Professional Computing Series), Addison-Wesley, Boston, MA.

[31] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., and Stal, M. 1996. *Pattern-Oriented Software Architecture Volume 1: A System of Patterns.* John Wiley & Sons, Hoboken, NJ.

[32] Fowler, M. 2002. *Patterns of Enterprise Application Architecture.* Addison-Wesley Professional Series. Boston, MA.

[33] Myers, C. 2003. Software systems as complex networks: Structure, function, and evolvability of software collaboration graphs. *Physical Review E,* 68, 046116

[34] Milo, R., Shen-Orr, S., Itzkovitz, S., Kashtan, N., Chklovskii, D., Alon, U. Network Motifs: Simple Building Blocks of Complex Networks. Science, 298, 5594, (October 25 2002) 824 – 827.

[35] Alexander, C., Alexander, M., Schmidt, R., Littlestone, N. Behrman, B., and Davis, H. Alberobello, Italy. http://www.livingneighborhoods.org/ht-0/fifteen.htm. courtesy of natureoforder.com. Retrieved 11/30/2008.

[36] Brad2512. Tower of the Wild Goose, Hunan Province, China. Posted by Brad2512 to the Photo section of http://www.travelpod.com. http://images.travelpod.com/users/brad2512/1.1210868160.big-wild-goose-pagoda.jpg

[37] Center for Models of Life, Neils Bohr Institute. Self-Assembling of Information in Networks. Created using demo at http://cmol.nbi.dk/models/infoflow/infoflow.html . Retrieved 12/1/2008.

[38] Derenyi, I., Farkas, I., Palla, G. and Viscek, T. Topological phase transitions of random networks. *Physica A.* 334, 3, (Mar 15 2004), 583-590.

[39] Palla, G., Derenyi, I., Farkas, T. and Vicsek, T. Statistical mechanics of topological phase transitions in networks. *Physics Review E.* 69, 4 (Part 2), (April 2004), 046117. DOI: 10.1103/PhysRevE.69.046117.

[40] Latora, V. and Marchiori, M. Economic small-world behavior in weighted networks. *The European Physical Journal B – Condensed Matter and Complex Systems.* 32, 2, (March 2003), 249-263.

[41] Fuller, B. 1975. *Synergetics: Explorations in the Geometry of Thinking*. Scribner, New York, NY.

[42] Kerth, N. Retrospectives.com at: http://www.retrospectives.com/. Retrieved 12/1/2008.

[43] Schwaber, K. Scrum Flow, www.controlchaos.com/images/diagram/flow.gif . Retrieved 12/1/2008.

[44] Sutherland, J. Future of Scrum: Support for Parallel Pipelining of Sprints in Complex Projects. In *Agile 2005 Proceedings* and at: http://jeffsutherland.com/Scrum/Sutherland2005FutureofScrum20050603.pdf. Retrieved 7/19/2008.

[45] Takeuchi, H. and Nonaka, I. The New New Product Development Game, *Harvard Business Review,* 64, 1, (Jan-Feb 1986), 137-146

[46] Cain, B. and Coplien, J. 1993. A Role-Based Empirical Process Modeling Environment, *Second International Conference on the Software Process: Continuous Software Process Improvement.* 125-133.

[47] Granovetter, M. 1983. The Strength of Weak Ties: A Network Theory Revisited. *Sociological Theory*, 1 202-233.

[48] Killworth, P. and Bernard, R. 1978. The Reversal of the Small-World Experiment? *Social Networks,*1, 159-192.

[49] DePree, M. 1992 *Leadership Jazz.* Dell Publishing, New York, NY.

[50] The Standish Group International. 2003. Chaos Chronicles. http://www.standishgroup.com/chaos/introduction.pdf. Retrieved 4/30/2004.