INTERNATIONAL CONFERENCE ON ENGINEERING DESIGN, ICED11
15 - 18 AUGUST 2011, TECHNICAL UNIVERSITY OF DENMARK

MULTILAYER NETWORK MODEL FOR ANALYSIS AND MANAGEMENT OF CHANGE PROPAGATION

Michael C. Pasqual[1] and Olivier L. de Weck[2]

ABSTRACT

A pervasive problem for engineering change management is the phenomenon of change propagation. This paper introduces a *multilayer network model* integrating three coupled layers of product development that contribute to change propagation: namely, the *product*, *change* (process), and *social* layers. A baseline repository of tools and metrics is developed for the analysis and management of change propagation using the model. The repository includes a few novel tools and metrics, most notably the Engineer Change Propagation Index (Engineer-CPI) and Propagation Directness (PD), as well as others already existing in the literature. A case study of a large technical program is discussed to demonstrate the model's practical utility. The case study reveals a correspondence between the propagation effects of an engineer's work and factors such as his/her organizational role and the context of his/her assignments. The study also confirms the counterintuitive possibility of indirect propagation between nonadjacent product components. Lastly, the study finds that propagation was generally infrequent and always stopped after five, and rarely more than three, generations of descendants.

Keywords: Engineering change management, change propagation, multilayer network model

1 INTRODUCTION

Engineering changes are inevitable during product development [1]. Through the process of engineering change management, an organization must balance the costs, benefits, and risks of implementing design changes in light of their implications for schedule, budget, and product quality.

Among the reasons why engineering changes can be so abundant and costly is the occurrence of *change propagation*. Change propagation can be defined as the "process by which a change to one part or element of an existing system [or product] configuration or design results in one or more additional changes to the system, when those changes would not have otherwise been required" [2]. In other words, change propagation occurs when making a single change ultimately requires the implementation of multiple downstream changes in order to achieve the objective of the intended redesign. For clarity, this research has adopted the term *parent-child propagation* to refer to the act of one change (the parent) yielding an immediate descendant change (the child). Otherwise, it can be confusing whether the term "propagation" refers to a single instance or repeated instances of parent-child propagation.

Efforts to quantitatively characterize, predict, control, and prevent change propagation, though limited, have quite naturally drawn on network-based models and analyses. For instance, at the heart of most previous research on change propagation is the popular method known as the *Design Structure Matrix* (*DSM*) [3][4]. A DSM is essentially an adjacency matrix representation of a directed network. Giffin et al. [2] extend the DSM concept to create the *Component Propagation DSM* (*Component-PDSM*),[3] which is a matrix that identifies instances of change propagation from one component to another. From the Component-PDSM, one can calculate the *Change Propagation Index* (*CPI*), which quantifies a component's propagation behavior by comparing the numbers of changes that propagate in and out of that component [2][5].

Despite the progress of change propagation research to date, a new approach may be beneficial to the field. This paper introduces a *multilayer network model* integrating three layers (or domains) of product development that contribute to change propagation: namely, the *product* layer, *change*

[1] Currently working at MIT Lincoln Laboratory in Lexington, Massachusetts, U.S.A.
[2] Engineering Systems Division, Massachusetts Institute of Technology, Cambridge MA 02139, U.S.A.
[3] Giffin et al. [3] actually call this tool the *Change DSM* or *ΔDSM*, but this paper substitutes the word "propagation" for "change" to help distinguish it from a DSM used to represent a change network.

(process) *layer,* and *social* layer. To the authors' knowledge, no previous research on change propagation has, at least explicitly, taken a multilayer network approach. The hypothesis here is that a multilayer network model provides a viable framework for the analysis and management of change propagation. A baseline repository of tools and metrics is developed for use with the model. The repository includes a few novel tools and metrics, in addition to others already existing in the literature. As such, the model unifies previous research on change propagation in a comprehensive paradigm. To support the hypothesis and demonstrate the model's practical utility, this paper discusses a case study of a large technical program which managed over 41,000 change requests in eight years. Giffin et al. [2][6] performed earlier studies of the same program.

2 MULTILAYER NETWORK MODEL OF CHANGE PROPAGATION

This section introduces a *multilayer network model* of change propagation. The model is composed of *three* layers that contribute to change propagation: the *product layer, change layer*, and *social layer*. As illustrated in Figure 1, the multilayer network model provides an intuitive and insightful representation of change propagation and the overall engineering change management process. That is, *engineers* in the *social layer* work on *changes* in the *change layer* that affect *components* in the *product layer*.

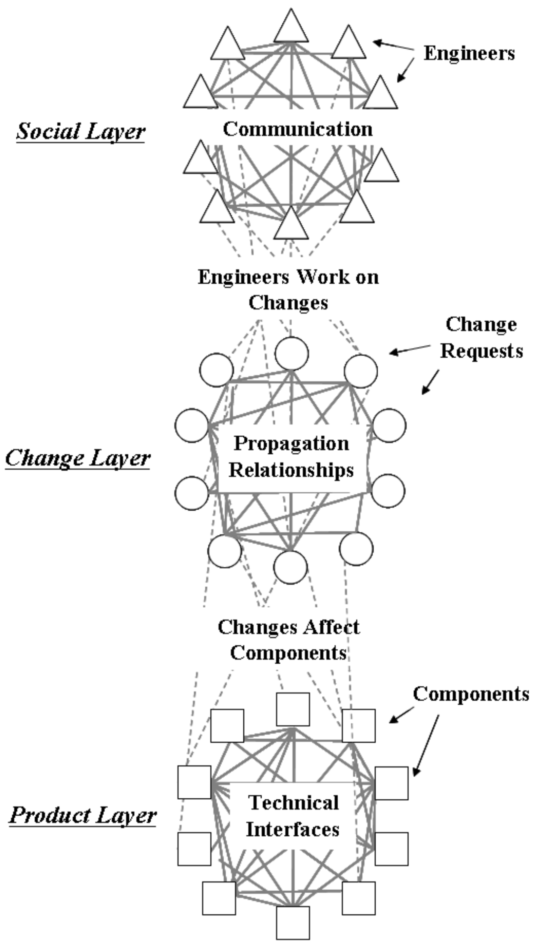


Figure 1. Multilayer network model

The multilayer network model captures the interactions within and across the product layer, change layer, and social layer. Each layer of the model consists of a distinct, directed network composed of *nodes* connected by *intra-layer edges*:

- The *product layer* is a network representation of the product or system being designed. The nodes of the network represent hardware components, software components, and associated documentation (e.g., requirements, specifications, and drawings). The (intra-layer) edges of the network represent technical interfaces among the components (or subsystems). The interfaces can be physical connections (e.g., by bolts or welding) or channels for the flow of energy (e.g., electrical power and heat), mass (e.g., fuel), and information (e.g., software inputs/outputs and control signals such as actuator commands) [5]. If a technical interface has direction (e.g., in the case of a flow channel), the edges can be directed. An edge might also identify a functional dependency that relates design variables to a desired performance level.

- The c*hange* (or process) *layer* is a network representation of change propagation. The nodes of the network represent individual changes or change requests (CRs). The (intra-layer) edges of the network represent propagation relationships among the changes. As in [6], directed edges can identify parent-child relationships, while bi-directional edges can identify sibling relationships between children of the same parent, or two changes related in a significant way.
- The *social layer* is a network representation of the organization. The nodes of the network represent people, e.g., teams, sub-teams, or individual engineers or employees. The (intra-layer) edges of the network represent various relationships among individuals and groups. For example, the edges might correspond to theoretical or actual communication links [8]. The edges might also reflect an organization's hierarchical structure or chain of command.

To complete the multilayer network model, the product, change, and social layers are linked together through *inter-layer edges*.

- *Social-to-change edges* relate the social layer to the change layer, depending on which engineers work on (e.g. propose, evaluate, approve, or implement) each change. If engineer *m* works on change *n*, then an inter-layer edge would connect node *m* in the social layer and node *n* in the change layer.
- *Change-to-product edges* relate the change layer to the product layer, depending on which component is affected by each change. If change *n* involves a redesign of component *k*, then an inter-layer edge would connect node *n* in the change layer and node *k* in the product layer.
- *Product-to-social edges* relate the product layer to the social layer, depending on which engineers are in charge of designing, redesigning, or sourcing each component. If engineer *m* is assigned to component *k*, then an inter-layer edge would connect node *m* in the social layer and node *k* in the product layer. Figure 1 does not illustrate the product-to-social edges, but they could be included, if desired.

3 HYPOTHETICAL APPLICATION

A hypothetical application should illustrate how to construct the multilayer network model for a given design scenario.

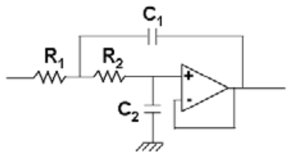


Figure 2. Sallen-Key low-pass filter to be designed

Suppose three engineers, John, Susan, and David, are designing a Sallen-Key low-pass filter (as in Figure 2) using two resistors (R_1 and R_2), two capacitors (C_1 and C_2), and a unity-gain amplifier. The amplifier has already been purchased. John is in charge of choosing the resistors, Susan is in charge of choosing the capacitors, and David is in charge of setting performance requirements, i.e., cutoff frequency (ω_c) and quality factor (Q). The resistors (in ohms) and capacitors (in farads) determine the low-pass filter's performance (ω_c in Hz and Q unitless) according to the following formulae.

$$\omega_c = \frac{1}{2\pi\sqrt{R_1R_2C_1C_2}} \tag{1}$$

$$Q = \frac{\sqrt{R_1R_2C_1C_2}}{C_2(R_1+R_2)} \tag{2}$$

Suppose the low-pass filter has been designed to have a baseline cutoff frequency of ω_c = 10 kHz, and quality factor Q = 0.5 (i.e., critically damped), per David's initial performance requirements. However, some changes inevitably become necessary. David decides that the cutoff frequency should now be 5 kHz instead of 10 kHz (change #1), but that the quality factor should remain at Q = 0.5. To facilitate this requirements change, the team initially plans for John to change R_1 (change #2), but that change is rejected because the resistors have already been ordered. Consequently, Susan must reselect the capacitors. Susan realizes that she cannot simply change one of the capacitors to accomplish the task of reducing ω_c while maintaining the same Q. In effect, the change to one capacitor will propagate, causing the other capacitor to change as well. Susan ultimately decides to double the original capacitances of C_1 (change #3) and C_2 (change #4) to reduce ω_c by a factor of two (10 kHz to 5 kHz), while maintaining Q = 0.5.

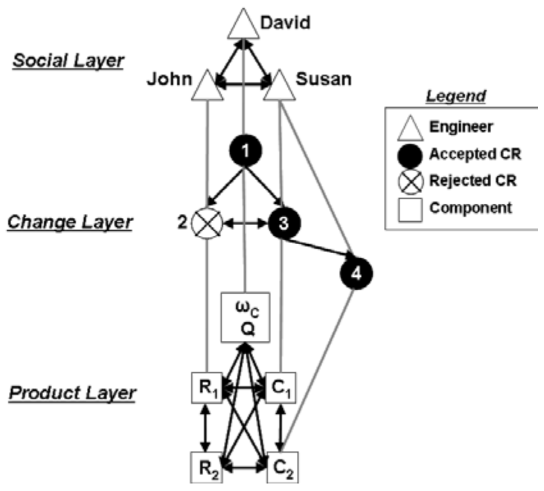


Figure 3. Multilayer network model of the hypothetical application

Figure 3 shows the multilayer network model corresponding to this hypothetical application. The model captures all the elements of the change activity that occurred. The social layer contains nodes for John, Susan, and David, with edges representing their communication links. The change layer contains nodes for changes #1-4 with edges representing propagation relationships. The edges indicate that change #1 had two children: change #2 (rejected) and change #3 (accepted) which also had a child in change #4. The product layer contains nodes for requirements and electrical components, with edges representing technical constraints. All the nodes in the product layer are connected to one another because ω_c, Q, R_1, R_2, C_1, and C_2 all depend on one another through Eq. 1 and Eq. 2. The inter-layer edges complete the story. The social-to-change edges represent how David, John, and Susan worked on changes #1, #2, and #3-4, respectively. The change-to-product edges represent how change #1, #2, #3, and #4 affected ω_c, R_1, C_1, and C_2, respectively. For visual simplicity, Figure 3 does not show the product-to-social edges. However, if drawn, these edges would represent how John, Susan, and David were in charge of R_1 and R_2, C_1 and C_2, and ω_c and Q, respectively.

4 TOOL AND METRIC DEVELOPMENT

The multilayer network model provides a platform for an array of tools and metrics for analyzing and managing change propagation. *Tools* here refer to methods for analyzing or visualizing the model's nodes and edges, while *metrics* refer to quantitative or qualitative measures for characterizing them.

This section presents a baseline repository of tools and metrics applicable to the multilayer network model. Table 1 summarizes the repository by categorizing each tool and metric according to the specific layer or layers it targets. The displayed matrix has a row and column for each layer of the model. As such, the items located along the diagonal are single-layer tools and metrics for the corresponding individual layer. The items in the upper-right triangle are double-layer tools and metrics for the corresponding pairs of layers. Finally, the items in the lower left triangle are triple-layer tools and metrics for all three layers at once.

As denoted by references in square brackets, Table 1 contains many tools and metrics that have already been proposed and utilized in the literature on change propagation, product development, and project management. Although these tools and metrics were not always explicitly classified in a multilayer context, they are easily incorporated into the multilayer network model. Consequently, the multilayer network model serves as a comprehensive paradigm that unifies past research in a common framework. Table 1 also contains a few new tools and metrics (marked with a "*") that are being proposed for the first time in this paper. These new tools and metrics are introduced in the next sub-section.

Table 1. Baseline repository of tools and metrics for the multilayer network model

	Product	**Change**	**Social**
Product	Tools - DSM [3][4] - CPM [9][10] Metrics - Graph properties [11] - Component type [2]	Tools - DMM [12] - Component-PDSM [2] - Change Prop. Freq Matrix [2] Metrics - Component CPI [2][5] - CAI/CRI [2] - Propagation Directness*	Tools - DMM [12] - Alignment Matrix [13]
Change		Tools - DSM [3][4] - Change motifs [2] Metrics - Graph properties [11] - Approval status [2] - Magnitude [2]	Tools - DMM [12] - Engineer-PDSM* Metrics - Engineer CPI* - Proposal Acceptance Rate [6]
Social	Tools - ESM [14] Metrics - Graph properties [11]		Tools - DSM [3][4] Metrics - Graph properties [11] - Organizational role*

* Proposed first in this paper

⬜ Single-layer ▒ Double-layer ▓ Triple-layer

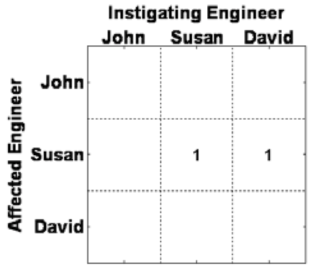
4.1 New Tools and Metrics

Thus far, previous research has provided a good number of tools and metrics applicable to the multilayer network model. However, the literature still seems to have a few weak areas, particularly if one wishes to analyze the social layer. Indeed, the literature on change propagation has lacked substantial quantitative treatment of the people involved in the design change process. This paper establishes a couple of new tools and metrics for this very purpose: the *Engineer Propagation DSM* and the *Engineer Change Propagation Index*. Another new item introduced here is a metric called *Propagation Directness*, which counts how many technical interfaces are spanned by an instance of parent-child propagation.

4.1.1 Engineer Propagation DSM

One goal of this research is to determine a way to analyze the propagation effects in the social layer. To this end, this paper proposes a double-layer tool called the *Engineer Propagation DSM* (*Engineer-PDSM*). The Engineer-PDSM identifies instances of change propagation from one engineer to another over some time period in the design process. The matrix is square with a row (m) and column (n) for each engineer in an organization. Element (m, n) of the Engineer Propagation DSM counts the number of times a parent change implemented by the *instigating* engineer n spawned a child change implemented by the *affected* engineer m.

Figure 4 shows the Engineer-PDSM corresponding to the three engineers (John, Susan, and David) from the hypothetical application in Section 3. The matrix indicates that parent-child propagation occurred twice. One change propagated from David to Susan, i.e., when David changed ω_c, Susan had to change the value of C_1. Another change propagated from Susan to herself, i.e., when Susan changed C_1, she also had to change C_2. It should be noted that David's change initially triggered a change for John to implement as well. However, because John's change (to R_1) was ultimately rejected, actual change propagation did not occur. Consequently, the rejected propagation does not appear in the Engineer-PDSM. This convention is also followed by [2].



Affected Engineer \ Instigating Engineer	John	Susan	David
John			
Susan		1	1
David			

Figure 4. Engineer Propagation DSM for the hypothetical application

4.1.2 Engineer-CPI

The Engineer-PDSM can be used to calculate a meaningful double-layer metric called the *Engineer Change Propagation Index* (*Engineer-CPI*). The Engineer-CPI quantifies an engineer's performance with respect to the propagation effects of his (or her) *implementation* of changes. The Engineer-CPI is a number between -1 and +1, calculated in the following equation.

$$Engineer-CPI(j) = \frac{E_{out}(j) - E_{in}(j)}{E_{out}(j) + E_{in}(j)} \qquad (3)$$

In the equation, $E_{out}(j)$ is the number of changes that propagated downstream from changes implemented by engineer *j*. $E_{in}(j)$ is the number of changes implemented by engineer *j* that propagated from changes implemented by other engineers. In other words, $E_{in}(j)$ and $E_{out}(j)$ are the in-degree and out-degree, respectively, of the Engineer-PDSM. Returning to the hypothetical application, one can calculate the Engineer-CPIs of David, Susan, and John to be 1, 0, and undefined, respectively.
It should be obvious that the Engineer-PDSM and Engineer-CPI are basically extensions of Giffin et al.'s [2] Component-PDSM and Component-CPI, respectively. Just as the Component-PDSM captures the occurrence of change propagation between product components or subsystems, the Engineer-PDSM captures the occurrence of change propagation between the engineers implementing those changes. As such, the Engineer-CPI spectrum can be interpreted similarly to the Component-CPI spectrum [2][5]; namely, positive, negative, zero, and undefined Engineer-CPIs correspond to multipliers, absorbers, carriers, and constants, respectively.
This paper proposes further that the Engineer-CPI spectrum should also map onto the spectrum of organizational roles. That is, an engineer's CPI should theoretically correspond with his or her job description. *Managers* and *systems engineers* will typically be *multipliers* ($E_{out} > E_{in}$) because they initiate high-level changes that potentially require many lower-level changes to be completed. For example, a manager might coordinate with customers and consequently change the requirements for a product to satisfy. Similarly, a systems engineer might recognize a high-level problem (e.g., given unsatisfactory test results) and consequently initiate corrective action that propagates through the product. By contrast, *specialists* tend to behave more like *absorbers* ($E_{in} > E_{out}$), because they perform changes in detailed areas of the product where there is little chance of further propagation. Specialists essentially implement changes at the end of propagation chains. Meanwhile, *team leaders* might correspond with *carriers* ($E_{in} = E_{out}$), since they pass on some high-level changes and may initiate changes on their own, but are also involved themselves with implementing low-level changes in the product. Finally, *constants* ($E_{in} = E_{out} = 0$) do not seem to have an obvious corresponding organizational role. If an engineer is a constant, that means he (or she) only implements isolated changes (i.e., they have no parent change and no children changes) or they are not involved in engineering change activity at all.

4.1.3 Propagation Directness

Propagation Directness (*PD*) is another double-layer metric proposed for the first time here. PD is defined as the number of internal product interfaces spanned by an instance of parent-child propagation. PD can be calculated using the Component-PDSM and Product DSM. Specifically, if the Propagation DSM indicates that a change propagated from component *n* to component *m*, then the PD of that propagation is equal to the geodesic (shortest) path from component *n* to *m* in the Product DSM.

Propagation Directness indicates whether propagation is direct or indirect. *Direct propagation* implies PD ≤ 1, because direct propagation occurs when a child change arises in a component that is adjacent (PD = 1) or identical (PD = 0) to the component affected by the parent change. By contrast, *indirect propagation* has PD > 1, because a child change arises in a component nonadjacent to the component affected by the parent change. Propagation Directness has obvious implications for the successful prediction of change propagation. Conventional wisdom says that Propagation Directness should always be PD ≤ 1; in other words, all propagation should be direct propagation. Accordingly, Clarkson et al. [9] and Keller et al. [10] notably only allow for direct propagation, but emphasize that recursive direct propagation can form propagation chains spanning several product interfaces. However, the program in the following case study experienced a considerable amount of indirect propagation, in which Propagation Directness was usually PD = 2, and occasionally PD = 3.

5 CASE STUDY

The case under investigation here is that of a large technical program whose purpose was to develop a large scale sensor system of globally distributed hardware and software segments. The software-dominated system can be decomposed into 46 areas, or coherent segments, of software, hardware, and different levels of associated documentation. These “areas” are roughly analogous to subsystems.

A rich dataset was extracted from the program‘s configuration management records. Giffin et al. [2] provide details on the data extraction methodology in their previous analysis of the same program. The dataset contains detailed information about 41,551 change requests (CRs) generated by the program over an eight year period.

5.1 Model Construction

The dataset enables us to construct a multilayer network model of the program’s experience with change propagation. In all, the dataset identifies 46 system areas, 41,551 change requests, and 501 engineers and administrators that constitute the nodes of the product layer, change layer, and social layer, respectively. The dataset also provides information on some, but not all, of the types of intra-layer and inter-layer edges. Of the intra-layer edges, only those in the product layer and change layer are discernible from the data. Of the inter-layer edges, only the social-to-change and product-to-change are discernible.

To illustrate the resulting multilayer network model, Figure 5 draws the multilayer network associated with a stand-alone change network referred to (by [2]) as 11-CR. 11-CR consists of 11 related change requests evaluated and implemented by nine engineers and affecting only three of the 46 system areas. In Figure 5, all the node labels correspond exactly with those in the raw dataset. For visual ease, the edge arrows have been removed. No intra-layer edges are shown in the social layer because the data were unavailable.

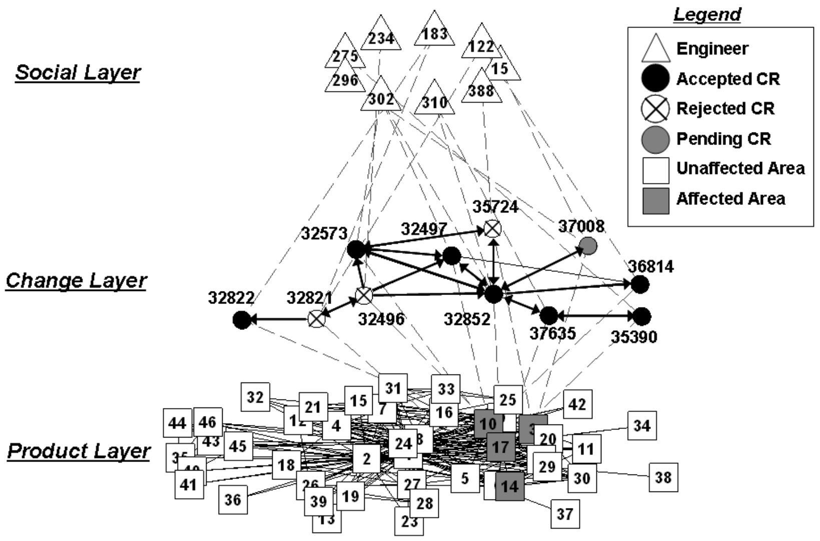


Figure 5. Multilayer network model for 11-CR

5.2 Analysis of Engineer Performance

The first thrust of this case study focuses on the social layer and its influence on change propagation and the change process. Specifically, the program's engineers are analyzed as *implementers* and *proposers* of change using the Engineer-CPI and Proposal Acceptance Rate, respectively.

5.2.1 Implementers of Change

One element of an engineer's work is the *implementation* of changes. To assess an engineer's performance in this regard, this case study uses the newly proposed Engineer-CPI. Figure 6(a) shows the distribution of Engineer-CPIs calculated for all 501 engineers identified in the dataset. The bars do not sum to 501, because nearly half of the engineers (226) actually behaved like constants (i.e., CPI undefined) who only work on isolated changes, i.e., they did not contribute to any change propagation.

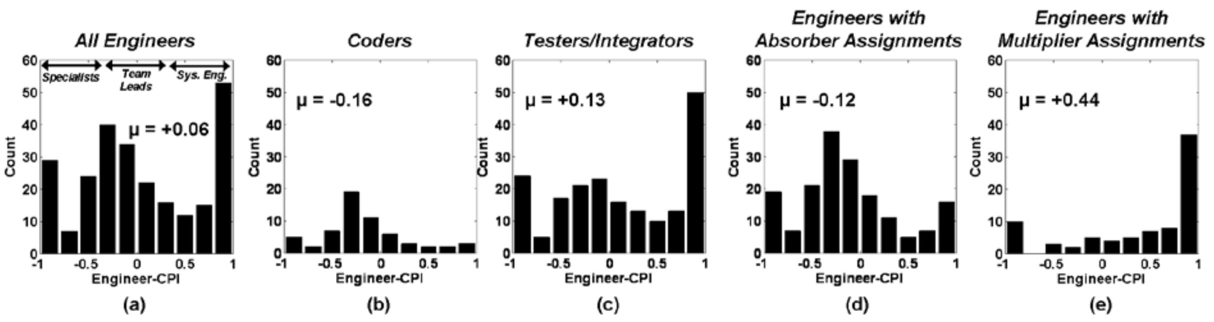


Figure 6. Distributions of Engineer-CPIs for various groups of engineers

The authors postulated earlier that the Engineer-CPI should correspond to the organizational role of an engineer, i.e., systems engineers are multipliers (CPI > 0), team leads are carriers (CPI = 0), and specialists are absorbers (CPI < 0). The data confirms this intuition. To determine the effects of an engineer's organizational role on his Engineer-CPI, the engineers in this program were divided into two classes: *coders* and *testers/integrators*. Coders were the specialists who actually made changes to lines of code within the system's software areas. By contrast, testers and integrators were more like systems engineers who tested and integrated the system areas together. In the absence of a detailed project directory, it was still possible to roughly classify each engineer according to a heuristic recommended by the lead systems engineer interviewed in this study. The heuristic classified an engineer as a "coder" if 60% or more of his work focused on core technology in the system. Otherwise, the engineer was classified as a "tester/integrator."

Figure 6(b) and (c) show the distribution of Engineer-CPIs for the coders and testers/integrators, respectively. The distributions offer some evidence that the Engineer-CPI indeed corresponds with an engineer's organizational role. As expected, the coders' distribution is skewed toward the absorber end of the spectrum. In fact, 74% of coders had negative CPIs. By contrast, the testers/integrators' distribution is skewed towards the multiplier end of the spectrum, with 53% having positive CPIs. The average coder's CPI was -0.16 (weak absorber), while the average tester/integrator's CPI was 0.13 (weak multiplier). Thus, this case study offers some verification of the correspondence between the Engineer-CPI and organizational roles. Namely, the coders (or specialists) tended to be absorbers, while the testers and integrators (or "systems" engineers) tended to be multipliers of change.

The data also suggests that another influence on an engineer's CPI is the context of his work, i.e., the propagation behavior of the areas to which an engineer is assigned to implement changes. The rationale here is that some engineers may be assigned to parts of the product that are inherently multipliers or inherently absorbers, as measured by their Component-CPIs. As a result, these engineers may have little independent control over the propagation effects of their work. To determine the effect of Component-CPIs on the Engineer-CPI, the engineers in this program were divided in two groups: those with absorber assignments and those with multiplier assignments. An engineer was said to have "absorber assignments" if the average Component-CPI of his assigned areas was negative (i.e., an absorber). Conversely, an engineer was said to have "multiplier assignments" if the average Component-CPI of his assigned areas was positive (i.e., a multiplier).

Figure 6(d) and (e) show the distribution of Engineer-CPIs for the engineers with absorber and multiplier assignments, respectively. The distributions offer some evidence that the Engineer-CPI indeed depends on the Component-CPI of an engineer's assigned areas. In fact, 67% of engineers with absorber assignments had negative CPIs (i.e., were absorbers), and 75% of engineers with multiplier assignments had positive CPIs (i.e., were multipliers). The average CPI for each group was -0.12 (weak absorber) and 0.44 (moderate multiplier), respectively. Thus, an engineer's CPI appears to be dictated not only by his organizational role, but also by the Component-CPIs of his assigned areas. That is, those engineers who work on multipliers and absorbers tend to be multipliers and absorbers themselves, respectively.

5.2.2 Proposers of Change

The other element of an engineer's work is the *proposal* of changes. A proposed CR will ultimately be accepted or rejected, depending on its costs, benefits, and risks from a systems perspective. The authors propose a two-dimensional scale for judging the performance of engineers as proposers of change. The scale's two dimensions are an engineer's Proposal Acceptance Rate (PAR) and the number of changes he/she proposed. Figure 7 plots the position of the 382 engineers who proposed any changes on this scale.

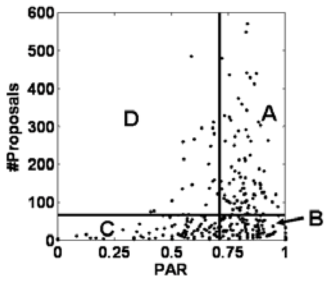


Figure 7. Proposal Acceptance Rate results

Following the advice of one of the program's lead systems engineer, Figure 7 is additionally broken into four quadrants, A, B, C, and D, which contain 85 (22%), 151 (40%), 123 (32%), and 23 (6%) of the 382 engineers, respectively. The quadrant boundaries are located at the average PAR and average proposal count of all 382 data points. Each quadrant has different implications for an engineer's performance, depending on his/her PAR and proposal count, relative to the average engineer:

- Quadrant A contains engineers with high PARs and high numbers of proposals. These engineers might be termed "high performers."
- *Quadrant B* contains engineers with high PARs but low numbers of proposals. These engineers likely have great ideas and good systems awareness, since their change requests are usually accepted. However, for some reason, they propose a relatively low number of change requests. The reason for the low proposal count may lie in the engineer's organizational role, personality, assigned system areas, or some other factor.
- *Quadrant C* contains engineers with low PARs and low numbers of proposals. These engineers

are relatively passive with only moderate activity level and little success as proposers of change.

- *Quadrant D* contains engineers with low PARs but high numbers of proposals. These engineers are very active in initiating changes but only a modest number of the proposed changes are actually implemented. There are two possible explanations for this troubling behavior. One is that these engineers tend to have lots of ideas that are ultimately rejected because the proposals are not well conceived. The alternative explanation is that the engineer is actually quite innovative, but the organization or product itself is stubborn or sluggish to change. Whatever the explanation, these engineers should be managed in a more focused way since they generate many change requests – each of them causing some effort for proper review and disposition – but a substantial fraction of them are not implemented.

5.3 Characterization of Change Propagation

The second thrust of this case study involves the general characterization of change propagation. The primary issue addressed here is the counterintuitive phenomenon of *indirect propagation*, a common occurrence for this program. Secondly, the study considers the issue of *propagation extent*, the number of generations of descendants propagated by an initiating change. In this program, propagation always stopped after five, and rarely more than three, generations of descendants.

5.3.1 Indirect Propagation

Conventional wisdom about change propagation assumes that only direct propagation is possible, i.e., a parent change in one component can only yield child changes in itself or immediately adjacent components [9]. However, the program discussed here experienced considerable indirect propagation, whereby child changes occurred in nonadjacent areas. Figure 8(a) displays the distribution of Propagation Directness values, considering every instance of parent-child propagation in the program in which the child change was accepted (regardless of the parent change's approval status). The distribution reveals that 78% of all parent-child propagation in the program was direct (PD$\leq$ 1), while a surprising 22% was indirect (PD > 1). The vast majority of indirect propagation occurred across two interfaces (PD = 2) and a handful (3) occurred across three interfaces (PD = 3). It should be noted that the maximum possible PD was three because the system network's diameter is three.

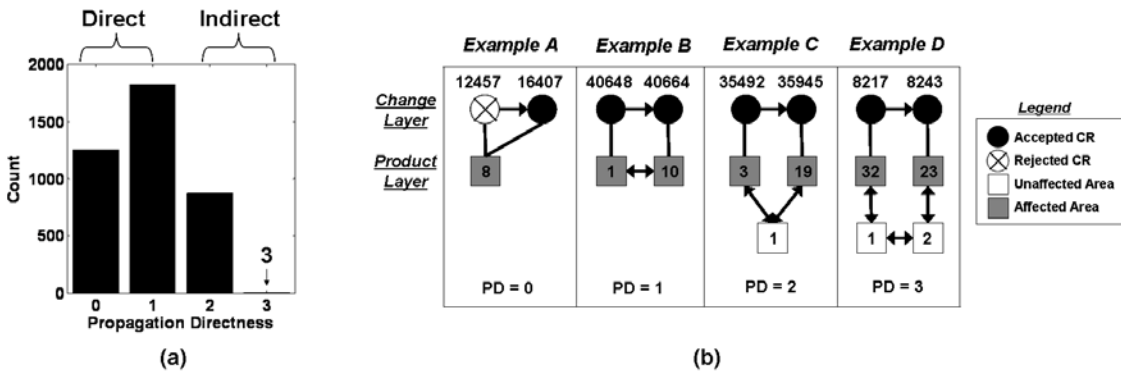


Figure 8. (a) Distribution of Propagation Directness, (b) Examples of change propagation

Delving further, Figure 8(b) illustrates a few examples of parent-child propagation from the dataset. In each illustration, the change layer contains the parent change and child change connected by a directed intra-layer edge. Meanwhile, inter-layer edges connect these changes to the affected areas in the product layer. For PD > 1, the product layer also contains the unaffected areas on the shortest path between the two affected areas. All nodes are labeled as they appear in the raw data. For simplicity, the social layer is omitted. Each example in Figure 8(b) has a different Propagation Directness value, which should be clear from the number of product interfaces spanned by the propagation. In Example A, self-propagation (PD = 0) occurred in Area #8; interestingly, the parent change in this example was ultimately rejected. Next, Example B shows direct propagation between adjacent areas (PD = 1); a change to Area #1, which contains requirements documentation, caused a change in Area #10, a core technology area. Example C exhibits indirect propagation; Areas #3 and #19 are separated by two interfaces (PD = 2) with Area #1 in between them. It should be noted that several geodesic (length-2) paths exist between Areas #3 and #19, besides the one through Area #1. Finally, Example D shows one of only three scenarios in the entire dataset with PD = 3. It is important to remember that in Examples C and D, the intermediate areas (connecting the two affected areas) were unaffected by any related change.

The phenomenon of indirect propagation contradicts conventional wisdom on change propagation. As such, one might conclude that if indirect propagation appears to have occurred, then the Product DSM must be missing some interfaces that actually exist; in other words, any observed indirect propagation is really direct propagation in disguise. If this explanation is true, then the Product DSM in this case study would be missing 192 interfaces. This seems unlikely. In fact, a lead systems engineer from the program explained that indirect propagation is a legitimate artifact of software system development. Apparently, engineers in this program would frequently violate the intended structure of the system in order to achieve a quick solution for a redesign. These ill-advised maneuvers were sometimes necessary during time crunches to meet development milestones (e.g., PDR, CDR, etc.). For example, one area of the system contained System Adjustable Parameters (SAPs). A SAP is a system variable kept in a loadable file, rather than in the software code itself. Many areas of the system were nominally disconnected from the SAP file. Still, on occasion, a hasty redesign effort would change the SAP file (e.g., adding an SAP), despite the lack of a direct interface between the SAP file and the parent area. In effect, a new interface was created, allowing a change to propagate; however, this interface was not part of the original Product DSM. Thus, indirect propagation, though unintended, can and does occur during product development. Additional case studies are necessary to determine if indirect propagation is a common artifact among software systems only, or hardware systems as well.

5.3.2 Propagation Extent

Propagation extent refers to the number of generations of descendants triggered by an initiating change. Eckert et al.'s [15] study of Westland Helicopters found that a change rarely occurs by itself and usually propagates no more than four generations. The data for the program here reaffirms the latter finding, but differs from the former.

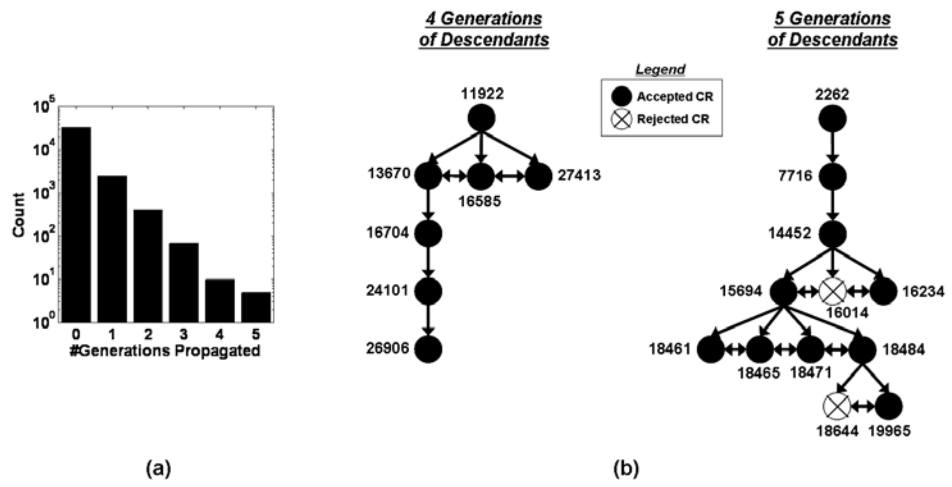


Figure 9. (a) Distribution of the #generations per un-parented change, (b) Examples of propagation chains

Figure 9(a) shows the program's distribution of the number of generations flowing from each *un-parented change* over this program's eight year period. An un-parented change is an individual change that is not the child of another change, and may or may not have any child changes of its own. In other words, each count in Figure 9(a) corresponds with a distinct propagation chain, whether it contains one isolated change or a line of descendants. In all, the program generated 36,184 un-parented changes. The results show that change propagation in the system almost always (99.99%) halted after four generations, just as Eckert et al. [15] reported in their study. There was only a handful (5) of changes that yielded five generations of changes, which was the maximum number of generations experienced; in other words, change propagation always ceased after five generations. Examples of propagation chains from the dataset with four and five generations of descendants are illustrated in Figure 9(b). All the node labels correspond exactly with those in the raw dataset.

Interestingly, the results in Figure 9(a) differ from Eckert et al.'s [15] finding that a change rarely occurs alone. In fact, isolated changes were actually the norm for this system; 91% of un-parented changes (33,152 out of 36,184) did not have any children (i.e., zero generations propagated). A deeper look into the context of each change may explain these statistics more. For instance, the large majority (80%) of changes in this program were low magnitude (0 or 1 on a scale of 0 to 5), which may explain the generally low probability of propagation. Overall, propagation extent likely stands as an extremely context-dependent feature of change propagation. This case study, at least, confirms that propagation vanishes after five generations of descendants, and rarely exceeds three generations.

6 CONCLUSION

This paper introduced a multilayer network model of change propagation, along with a baseline repository of single-layer, double-layer, and triple-layer tools and metrics. As summarized in Table 1, the multilayer network provides a comprehensive paradigm that unifies previous research (i.e., tools and metrics) in a common framework. A case study of a large technical program demonstrated the practical utility of the multilayer network model. The Engineer-CPI, a newly introduced metric, was used to quantify the propagation effects of an engineer's implementation of changes. The data indicated that the Engineer-CPI is partially dependent on an engineer's organizational role and the context of his assignments. Coders and engineers who worked on absorbers in the system tended to behave like absorbers themselves. Meanwhile, testers/integrators and engineers who worked on multipliers in the system tended to behave like multipliers themselves. The program's engineers were also analyzed as proposers of change with respect to their Proposal Acceptance Rate (PAR) and the total number of changes they proposed. The case study also investigated the counterintuitive occurrence of indirect propagation, by which changes propagate between nonadjacent product components. It was found that software-intensive systems may be particularly susceptible to indirect propagation. Finally, the study found that most changes did not lead to any propagation. Propagation that did occur always stopped after five, and rarely more than three, generations of descendants.

Taking a multilayer view in an engineering program holds the promise of turning change management from a rather passive administrative process to a more predictive and proactive systems engineering process. An example of this would be the ability to rank or tag change requests by their likelihood of initiating long propagation chains based on the subsystem areas involved and other factors. Quantifying areas of the system as multipliers and absorbers and "hot spots" for change allows for a more statistically-based and predictive approach leading to more accurate cost and schedule estimates. Another promising area is the use of the Engineer-CPI and PAR metrics as measures for personnel management and performance assessment. A more conscious assignment of roles and identification of engineers who fall into quadrant D (Figure 7) may help identify those who might benefit from additional training. Future work includes the implementation of the new metrics and multi-layer visualizations in industrial practice, as well as the ability to leverage the insights gained here for statistical change propagation predictions.

REFERENCES

[1] Nichols, K. Getting engineering changes under control. *Journal of Engineering Design*, 1990, 1(1), 1-6.

[2] Giffin, M., de Weck, O., Bounova, G., Keller, R., Eckert, C., and Clarkson, J. Change propagation analysis in complex technical systems. *Journal of Mechanical Design*, 2009, 131(8), 081010.

[3] Steward, D.V. The design structure matrix: a method for managing the design of complex systems. *IEEE Transaction on Engineering Management*, 1981, 28(3), 71-74.

[4] Eppinger, S., Whitney, D., Smith, R. and Gebala, D. A model-based method for organizing tasks in product development. *Research in Engineering Design*, 1994, 6(1), 1-21.

[5] Suh, E.S. and de Weck, O.L. Flexible product platforms: framework and case study." *Research in Engineering Design*, 2007, 18 (2): 67-89.

[6] Giffin, M. Change propagation in large technical systems, 2007, S.M. Thesis. SDM, MIT.

[7] Eppinger, S. D. Patterns of product development interactions. In *International Conference on Engineering Design, ICED'01*, Vol. 1, Glasgow, August 2001, pp.283-290 (Professional Engineering Publishing, Bury St Edmunds).

[8] Morelli, M.D., Eppinger, S.D. and Gulati, R.K. Predicting technical communication in product development organizations. *IEEE Transactions on Engineering Management*, 1995, 42(2), 215-222.

[9] Clarkson P.J., Simons, C. and Eckert, C. Predicting change propagation in complex design. *Transactions of ASME*, 2004, 126, 788-797.

[10] Keller, R., Eger, T., Eckert, C.M. and Clarkson, P.J. Visualizing change propagation. In *International Conference on Engineering Design*, *ICED'05*, Melbourne, August 2005, pp.62-63.

[11] Newman, J. The structure and function of complex networks. *Society of Industrial and Applied Mathematics*, 2003, 42(2): 167-256.

[12] Danilovic, M. and Browning, T.R. Managing complex product development projects with design structure matrices and domain mapping matrices." *International Journal of Management*, 2007, 25, 300-314.

[13] Sosa, M.E., Eppinger, S.D., and Rowles, C.M. Are your engineers talking to one another when they should? *Harvard Business Review*, 2007, 133-142

[14] Bartolomei, J. Qualitative knowledge construction for engineering systems: extending the design structure matrix methodology in scope and procedure, Thesis, 2007, MIT Engineering Systems Division.

[15] Eckert C., Clarkson P. and Zanker W. (2004). Change and customization in complex engineering domains. *Research in Engineering Design*, 2004, 15, 1-21.