Paper to be presented at

DRUID15, Rome, June 15-17, 2015

(Coorganized with LUISS)

Towards a microstructural view of architectural change in complex systems: the role of design hierarchy and modularity

Mahdi Ebrahim
Bocconi University
Management & Technology
mahdi.ebrahim@phd.unibocconi.it

Arnaldo Camuffo
Bocconi University
Technology and management
arnaldo.camuffo@unibocconi.it

Abstract

Despite the large body of technology and innovation management research analyzing how product architectures evolve over time, the debate about what factors shape such evolution is inconclusive. While hierarchy and near-decomposability are argued to enhance the evolvability of complex systems, scant empirical research analyzed the different mechanisms through which design modularity and hierarchy affect product architecture complexity. Leveraging on metrics developed by recent advances in network and graph theories, we apply panel data regression analysis on real data coming from multiple versions of an industrial software application to empirically test the effect of modularity and hierarchy on design complexity change. The results show that both dimensions reduce the degree of complexity of the software. Unexpectedly, however, they also show that design hierarchy impacts design complexity more than design modularity, which is somehow at odds with the current emphasis on modularity as a complexity reduction approach. The absence of correlations and interaction effects suggest that these two dimensions are both needed to capture and contain architectural complexity

Jelcodes:O32,-

Towards a microstructural view of architectural change in complex systems: the role of design hierarchy and modularity

ABSTRACT

Despite the large body of technology and innovation management research analyzing how product architectures evolve over time, the debate about what factors shape such evolution is inconclusive. While hierarchy and near-decomposability are argued to enhance the evolvability of complex systems, scant empirical research analyzed the different mechanisms through which design modularity and hierarchy affect product architecture complexity. Leveraging on metrics developed by recent advances in network and graph theories, we apply panel data regression analysis on real data coming from multiple versions of an industrial software application to empirically test the effect of modularity and hierarchy on design complexity change. The results show that both dimensions reduce the degree of complexity of the software. Unexpectedly, however, they also show that design hierarchy impacts design complexity more than design modularity, which is somehow at odds with the current emphasis on modularity as a complexity reduction approach. The absence of correlations and interaction effects suggest that these two dimensions are both needed to capture and contain architectural complexity.

Keywords:
modularity; hierarchy; architectural design; software complexity

INTRODUCTION

Our everyday life relies on complex technological systems to perform ever exceedingly complex tasks. These technological systems, often taking the form of products and services, must be designed so that they are able to evolve and flexibly adapt so that they can address new needs and incorporate new functionalities.

On the one side, a large body of research shows that the design of complex product systems is inert, rigid over time, and hard to change and improve (Baldwin, Maccormack, & Rusnak, 2014). On the other side, evidence from a variety of industries, from airplanes to smartphones and from banking to telecom, shows that product technologies often fail to evolve because of exceeding design complexity. This largely stems from the fact that product design requires massive collective work whose effects are often unknown and not calculable, and whose nature is chaotic and unplanned.

Simon's (1962) groundbreaking work on the architecture of complexity suggested that products, conceptualized as complex, technological, adaptive systems should be designed, in order to advance their evolvability, according to the dual properties of hierarchy and near-decomposability. Ethiraj and Levinthal (2004, p. 404) elaborate on this, arguing that hierarchy and near-decomposability are desirable attributes of complex technological systems, defining hierarchy as the fact that some product elements or components provide constraints on lower-level elements or components, and near-decomposability as the fact that patterns of interactions among elements or components of a system are not diffuse but tend to be tightly clustered into nearly isolated subsets of interactions.

Despite the enormous influence of Simon's (1962) architecture of complexity on design literature, no study has, to date, taken on the task to empirically investigate whether and to what extent near-decomposability (modularity[1]) and hierarchy actually affect changes in design[2] of product technology.

This study undertakes such challenge providing an empirical test of the individual and combined effects of modularity and hierarchy on the evolution of the complexity of a technological system (i.e. software). In order to do that, it uses metrics developed by recent advances in network and graph theory and apply panel data regression analysis on real data coming from multiple versions of an industrial software application to empirically assess the effect of modularity and hierarchy on design complexity change. We test the hypotheses that the degree of modularity and hierarchy of a given system's element differently affects how product complexity changes over time. Less modular and, surprisingly to a larger extent, more hierarchically constrained design elements, are at higher risk of getting more complex as the product evolves. As design complexity is widely known to drive product malfunctions (Banker, Davis, & Slaughter, 1998), customers' dissatisfaction (Ethiraj, Ramasubbu, & Krishnan, 2012; Ramasubbu & Kemerer, 2015), designers' rework and stress (Sturtevant & MacCormack, 2013), and even increased employees' turnover (Sturtevant, Maccormack, Magee, & Baldwin, 2013), managers need to keep it under control and prevent it from increasing unintentionally.

[1] Please note that in the rest of the paper we use near-decomposability and modualrity interchangeably, as the two concepts are theoretically equivalent.

[2] For the sake of simplicity and readability, in the rest of the paper we ususally use terms design or archtiecture instead of archtiectural design, which refer to a set of design elements and the way they are connected to each other.

The paper is structured as follows. In the next section, we set the theoretical framework, define design complexity, modularity and hierarchy and state the hypotheses. Then, we move to the description of the research design, data and method. We illustrate the empirical setting and the data, describe the measures used to capture the evolution of design complexity, modularity and hierarchy, and clarify the model specification and estimation. Section four presents the findings, while the fifth section discusses them. The concluding section draws theoretical and managerial implications and highlights the study's limitations and directions for future research.

THEORY

In this section, we follow mainstream complexity literature (Ethiraj & Levinthal, 2004, 2002; Murmann & Frenken, 2006) articulating complex system's architecture into two key dimensions: modularity and hierarchy. We operationalize these constructs by introducing the concepts of Design Structure Matrix (DSM) (Eppinger, Whitney, Smith, & Gebala, 1994; Robert P. Smith & Eppinger, 1997), Transitive Design Structure Matrix (TDSM) (MacCormack, Rusnak, & Baldwin, 2006), and Hidden Structure (Baldwin, MacCormack, & Rusnak, 2014), formalizing the associated types of architectural configurations. Finally, a set of hypotheses summarize the theoretical arguments, explaining the mechanisms through which design modualrity and design hierachy influence the evolution of architecture's complexity.

Conceptual Framework

Design structure matrix (DSM)

Most of the methodologies to represent the design of a complex system are based on the assumption that design can be well modeled by recording the pattern of (directional) links among its constituent elements (Robert P Smith & Eppinger, 1997). Therefore, Design Structure Matrix

(DSM) captures such representation of links, or alternatively, dependencies[3]. The DSM is a square matrix where each element of the system takes one column and one row. Then, if element A depends on element B, there will be a 1 in cell (A,B) of the matrix. In other words, we put 1 on the Ath row and Bth column. The whole matrix then represents directional dependencies among all elements of the system. The figure bellow is an example of a design configuration with three elements such that A depends on B, and there are no other dependencies.

Alternatively, if element A is dependent on element B, and in turn, element B is dependent on element C, then, A is indirectly dependent on C. to capture both direct and indirect dependencies in our analyses we use an extension of DSM, Transitive Design Structure Matrix, that represents indirect dependencies as well as direct ones.

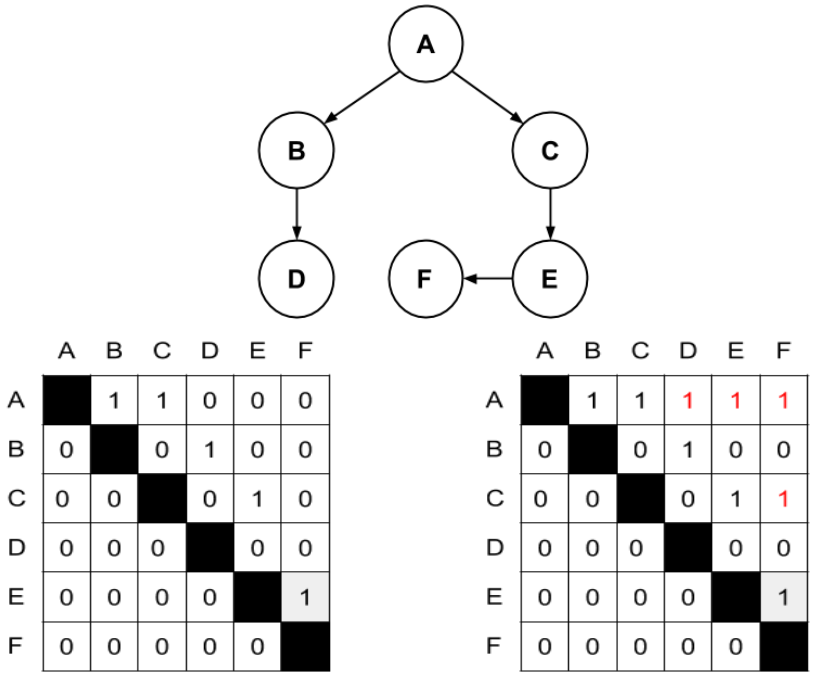

Figure 1 An example of design configuration (top), represented by DSM (bottom left) and TDSM (bottom right)

[3] When element A depends on element B, for example to accomplish a task, one could assume a dependency taking the form of a directional link that connects element A to element B.

Representing architectural design based on modularity and hierarchy

Complex architectural design could be parsimoniously categorized by two distinctive design characteristics: design modularity and hierarchy (Ethiraj & Levinthal, 2004, 2002; Murmann & Frenken, 2006). A design is modular when it is nearly-decomposable into subsets of elements (Simon, 1962), rendering maximum internal coherence and minimum external coupling to elements of other subsets of the system (Alexander, 1964). A DSM representation of modular design configuration consists of square sub-blocks along the matrix's diagonal. The sub-blocks represent modules of the system and capture links between elements inside the same module. If elements in different modules are not linked to each other, its corresponding visual representation in the TDSM would be blank cells outside the sub-blocks. As an example, the design configuration and its corresponding TDSM are presented in Figures 2 and 3.

Alternatively, architectural design configuration could be identified based on its level of hierarchy. A completely hierarchical design does not involve any reverse (or reciprocal feedback) link between elements. In such architectural configuration, a change in one element only propagates to elements below it in the dependency chain, and does not affect any element above it. The graphical demonstration and corresponding TDSM of hierarchical design is presented in Figures 2 and 3.

Based on these two distinctive design characteristics, any architectural configuration could be identified by a 2 by 2 framework of modularity and hierarchy. This framework facilitates thinking about microstructure of design, as the two design aspects could be assigned to each element of design. Accordingly, modular elements possess more links to elements inside their module compared to links with elements outside the module. Similarly, hierarchical elements possess fewer reciprocal feedback links.

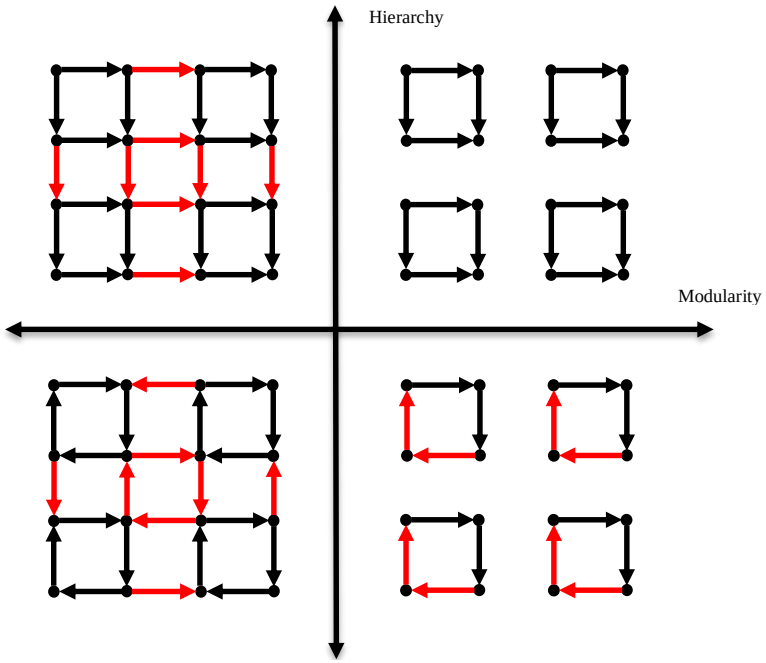

Figure 3 design configurations categorized by modularity and hierarchy

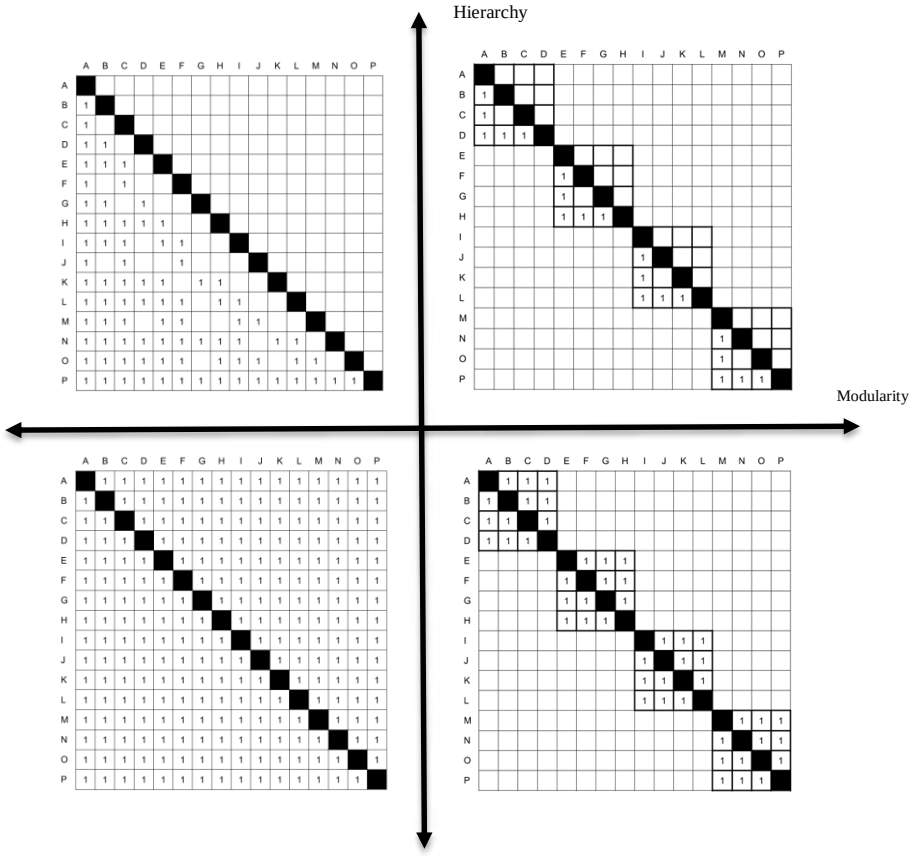

Figure 2 TDSMs corresponding design configurations categorized by modularity and hierarchy

Hypothesis Development

What explains the evolution of architecture in complex systems? Among the different driving factors that affect the architecture of the product in next generation, the architecture of previous generation appears to play a key role, as complex systems found to be inert in their architectural design (Baldwin et al., 2014; MacCormack, Rusnak, & Baldwin, 2007; MacCormack et al., 2006). Therefore, here we examine the role of two distinctive design characteristics that parsimoniously identify different types of architectural configurations, and discuss the alternative mechanisms suggested by literatures of architectural design and complex theory. We conclude this section with a discussion focused on knowledge combination versus knowledge disruption, when there are changes in the architecture of a product. There we argue for the dominant role of non-hierarchical design over non-modular design in making the next generation of design more complex.

Design modularity and architectural change

The modularity breeds modularity hypothesis, prevalent in design literature, suggests that modular design is combinable (Cabigiosu, Camuffo, & Schilling, 2014), meaning that it readily accommodates new modules. Modular design is adaptive and flexible in that, its modules are nearly decomposed from each other, and therefore, any of them could adapt to changes without a need to change other modules of the system. Therefore, the extant research in design literature proposes that the more modular elements are more adaptable to change and less likely to increase the complexity of the system(Baldwin & Clark, 2000; MacCormack et al., 2007; Sanchez & Mahoney, 1996).

The main argument here is that design modularity helps the developers to limit the propagation of change inside a module and prevent it from affecting the rest of the system. The crux of the argument points to individual's cognitive limitations and bounded rationality (Ethiraj & Levinthal,

2004). If the system is not modular though, a change in an element could propagate to the rest of the system and affect many other elements. Then, since checking all the affected elements surpasses employee's cognitive capacity, it causes bugs in design and deflections in the functioning of the whole system, and finally, leads to a more complex design and incurs technical debt (Ramasubbu & Kemerer, 2015; Sturtevant, MacCormack, Magee, & Baldwin, 2013; Sturtevant & MacCormack, 2013).

Hypothesis 1. Evolving from one generation to the next, modular elements are less likely to increase their complexity.

Design hierarchy

As discussed earlier hierarchy of design is one of the two distinctive factors that identify architectural configurations. Therefore, in studying the change of system's architecture, examining the role of design hierarchy is valid and relevant. However, extant architectural design literature has just recently started to study this effect (for example refer to C. Baldwin, MacCormack, & Rusnak, 2014). It appears that the design literature is still immature to uncover the microstructural aspects of the effect of design hierarchy on architectural changes.

Yet, to shed some light on the effect of design hierarchy on architectural evolution extant literature of problem solving in complex contexts provides beneficial arguments and insights. Research in disciplines of organization design and software engineering suggests that product architecture corresponds to the organizational arrangement of developing teams (Colfer & Baldwin, 2010; Sanchez & Mahoney, 1996). This proposition is referred to as mirroring hypothesis in organization theory literature (Baldwin & Clark, 2000; Henderson & Clark, 1990; Sanchez & Mahoney, 1996; von Hippel, 1990), *Conway's law* in software engineering (Conway,

1968; Kwan, Cataldo, & Damian, 2012), and more recently socio-technical congruence (Betz et al., 2013; Cataldo, Herbsleb, & Carley, 2008; Herbsleb, 2007; Valetto et al., 2007). Therefore, to understand the effect of design hierarchy on change in the product's architecture, organizational design literature lends promising insights.

The mirroring hypothesis suggests in complex systems technical architecture, division of labor and division of knowledge (ideally) correspond to each other. In the other words, these three features "mirror" each other and the network structure of each, correspond to the network structures of the others (Brusoni & Prencipe, 2001, 2011; MacCormack, Baldwin, & Rusnak, 2012; MacCormack et al., 2006; Sanchez & Mahoney, 1996). For example, imagine there is a non-hierarchical link between two design elements (e.g. a reciprocal dependency between two elements). Then, the hypothesis predicts that there is reciprocal communication between the two employees that work on each of the two elements.

The main argument behind the mirroring hypothesis is information sharing requirements. Technical dependencies among elements of complex systems necessitate information sharing among the developers of those elements. To adapt one element to any change in the other reciprocally dependent element, sharing information about each of the elements is crucial (Furlan, Cabigiosu, & Camuffo, 2014). The need for information sharing is escalated when instead of having only two dependent elements there are a number of them, simultaneously and reciprocally linked to each other, forming a cyclic group.

In a cyclic group any change in one element requires a lot of experimentations, reiterations of design, and rework, to make every element in the cyclic group function well together (C. Baldwin et al., 2014). Fulfilling this difficult task incurs costs such as higher technical debt, lower

productivity of employees, and even higher rates of personnel turnover (Sturtevant et al., 2013; Sturtevant & MacCormack, 2013). It requires an ongoing communication between employees working on the elements that belong to the same cyclic group. This is necessary to create common knowledge of system elements and the way they are connected, and to share it among the developers (Srikanth & Puranam, 2011). Moreover, the process of assimilating new knowledge by developers is not immediate (Srikanth & Puranam, 2014). Consequently, the impediments to sharing necessary information and knowledge lead to more complex architectural design.

Hypothesis 2. Evolving from one generation to the next, hierarchical elements are less likely to increase their complexity.

Design modularity vs. design hierarchy on influencing architectural change

Non-modularity of an element means that it has links that crosses the boundary of a module and connects two nearly decomposable modules. It provides new opportunities for innovation by tapping into new combination of previously isolated bodies of knowledge embedded in separate modules. If the interface between two modules is carefully considered and addressed the outcome of the design non-modularity is balanced and limits the risk of disrupting current knowledge base of the firm (Baldwin & Clark, 2000; Karim & Kaul, 2015).

Alternatively, non-hierarchical element has links that are heterarchical, reversing the flow of dependency and forming a cyclic group of at least two elements. These types of links mix different bodies of knowledge embedded in elements of the cyclic group. To make these types of links work seamlessly, the developer should carefully consider and redesign dependencies between all elements in the cyclic group. This demanding task requires a lot of iteration and rework on the design and experimenting and testing numerous variations to get the best performance of the

system. The task could easily get out of control when the employee tries to orchestrate all the dependent elements to function properly; let alone learning the knowledge of every single element in the cyclic group. Therefore, the task could simply lead to more haphazard design decisions, which in turn, makes the design more complex (Ramasubbu & Kemerer, 2015).

Hypothesis 3. Evolving from one generation to the next, non-hierarchical elements, compared to non-modular elements, are more likely to increase their complexity.

METHODOLOGY

Description of the Data

This research uses technical data of architectural design of a software system. It captures the data of system's elements (i.e. files) and their links (i.e. dependencies). It has been extracted by the means of a software package called Understand, which parses lines of codes in all files, and extract various types of dependencies between them, such as function calls, uses, reads, etc. In addition, the dataset contains information of every single change to the architecture of the software over the period of the study.

The software system analyzed in this research is developed by a leading and innovative HVAC (heating, ventilation, and air conditioning) company. The main function of the software is supervising HVAC micro-controllers in plants such as industrial fridges and supermarkets. The software has undertaken 13 minor and major versions over the period of 5 years. Therefore, it provides a good opportunity to study the change in the architecture of a complex system, while demonstrating a reasonable heterogeneity of design modularity and hierarchy at the element level.

Variables and Measurements

Dependent variable.

Change in design complexity is the dependent variable of all three hypotheses. It is measured at the element level and is the difference between the number of links (direct or indirect) a focal element has, from version t to version t+1 (MacCormack et al., 2007). More formally, it could be stated as the following:

$$\Delta\ links\ of\ the\ focal\ element = number\ of\ links\ of\ the\ focal\ element_{\ t+1} - number\ of\ links\ of\ the\ focal\ element_{\ t}$$

Capturing design complexity by the number of dependencies between its comprising elements is prevalent in complexity literature (Rivkin & Siggelkow, 2003; Zhou, 2013). Any dependency between system's elements stipulates dependency between design decisions and tasks; any design change in one of the two interdependent elements necessitates modifications in design of the other, to make sure that the two elements could still seamlessly work together.

Independent variables.

Design modularity. According to the hypotheses there are two main independent variables: design modularity, and design hierarchy, both measured at the element level. Design modularity, as suggested by the technical design literature (Cabigiosu et al., 2014; Huang & Kusiak, 1998) is measured by conventional measurement of proportion of links to elements outside of the focal element's module to its total number of links. This measurement, roots back to the concept of decomposability of modules and their internal coherence and external decoupling (Alexander, 1964), captures the partial membership of a focal element to its module.

$$Modularity = \frac{number\ of\ links\ to\ elements\ inside\ the\ focal\ element's\ module}{total\ number\ of\ links\ of\ the\ focal\ element}$$

For identifying the modules in the system's architecture, we benefited from recent mathematical developments in graph theory and novel improvements in calculation methods in the network literature. We used the Louvain algorithm (Blondel, Guillaume, Lefebvre, & Lambiotte, 2008) for identifying the communities (clusters or modules) inside a large network of elements and links. Among different methods of identifying clusters in networks Louvain algorithm is based on modularity optimization. It is shown to outperform all other known community detection method in terms of computation accuracy and time (Blondel et al., 2008).

The method consists of two phases. First, it looks for small communities by optimizing modularity in a local way. Then, it aggregates nodes of the same community and builds a new network whose nodes are the communities. These steps are repeated iteratively until a maximum of modularity is attained. The modularity of a partition is a scalar value between -1 and 1 that measures the density of links inside communities as compared to links between communities.

$$Q = \frac{1}{2m}\sum_{i,j}\left[A_{ij} - \frac{K_i K_j}{2m}\right]\delta(c_i, c_j),$$

where A_{ij} represents the weight of the link between i and j, $k_i = \sum_j A_{ij}$ is the sum of the weights of the edges attached to vertex i, c_i is the community to which vertex i is assigned, the δ-function δ(u, v) is 1 if u = v and 0 otherwise, and $m = \frac{1}{2}\sum_{ij} A_{ij}$ (Blondel et al., 2008).

Design non-hierarchy. It is measured as the size of the cyclic group the focal element belongs to. The higher the size of the cyclic group means the more non-hierarchical links and reciprocal dependencies exist. The size of the cyclic group is calculated by an algorithm suggested by

Baldwin, MacCormack, and Rusnak (2014). It is based on specific type of reordering the Transitive Design Structure Matrix (TDSM) of the whole system's architecture, to identify the links that could not be pushed below the matrix's diagonal, and hence, are non-hierarchical. Sorting the TDSM, first descending based on inward links and then, ascending based on outward links, identifies the non-hierarchical links. For a detailed review of the methodology and proofs, please refer to the original paper.

Control variables.

A set of control variables, as suggested in the software engineering literature as key factors, is added to control for the heterogeneity in the content of files. Controlling for technical file-related variations of the dependent variable is necessary to isolate the effect of proposed explanatory variables from other possible sources of variations.

Accordingly, the variable Java language captures the language that the file is written in (Java files=1, C files=0). Lines of codes, LOC, is a software metrics used to measure the size of a computer program by counting the number of lines saved in the program's source code. Comment to LOC calculates the ratio of lines of comments added inside the source code to statement lines of codes. It is a measurement of knowledge-intensity of the file develped. Finally, cyclomatic complexity measures the internal (and not structural) complexity of the file and hygiene of codes written and saved in the file.

Model Specification

To test the proposed hypotheses and estimate the effects of dependent variables we run panel data regressions on our almost balanced longitudinal dataset. The formal specification of our base econometric model is the following:

$$\Delta\, links_{it} = \beta_0 + \beta_1\, modularity_{it} + \beta_2\, hierarchy_{it} + X_{it}\mathrm{B} + \mu_i + \varepsilon_{it}$$

where, the dependent variable records the change in the number of links each individual file i has at version t. Independent variables capture individual file's modularity and hierarchy at version t. A set of control variables, mostly concerns technical aspects of files as suggested by software engineering literature, is added to control for variation in dependent variable generated by factors other than those hypothesized.

The panel data model is a random effects model, assuming that individual's fixed effects on the dependent variable is randomly assigned. The assumption is valid in the context of the study, wherein not any systematic and unobserved differences between files exist. Files, unlike human individuals or social systems and organizations, are transparent entities and every aspect of them (either their content or architectural characteristics) is extracted and captured in our dataset. Therefore, in the econometric model the distribution of μ_i and ε_{it} are assumed to be i.i.d N(0,σ^2) and independently realized.

EMPIRICAL RESULTS

This section delivers the results of the empirical analyses. We first, demonstrate a statistical description of main variables, followed by correlation results. Then, the identification strategy and statistical regression models are reported. The regression results are communicated next, which provide support for the three hypotheses proposed in the theory section. Lastly, the results of a series of robustness checks are delivered, which confirms the robustness of results over partially different conditions.

Descriptive data

The dataset used in this study comprises 13 versions of a single software over a period 5 years. Overall, it includes 41472 observations (files*versions), which provides a promising opportunity to study the microfoundation of architectural change over time and at micro-level data of design elements. Table 1 bellow provides a statistical description of main variables used in this research.

Insert table 1 about here

First two variables, dependent variables of this study (added links to the focal element from version t to version t+1, either directly or indirectly), demonstrate a reasonable range and heterogeneity. Therefore, and considering the longitudinal nature of the research question, we used panel data regressions to test the hypotheses. Other dependent variables and controls demonstrate a good range and variation as well, and could be entered the to the regression model without any further empirical consideration.

Table 2 below reports the correlation results for the variables entered the regression models. The correlations between variables that enter the same model, as dependent variables, are reasonably low (all are below 0.3). This fact indicates that in regression models, we explain the variation of the dependent variable by orthogonal variables, and hence, not mixing the effects of different factors influencing the dependent variable. In other words, in the empirical part we test the hypotheses correctly, and isolate the effect of each proposed independent variable in the regression model.

Insert table 2 about here

Regression results

Table 3 reports the results of four regression models. Considering the range and continuous nature of dependent variables, which is observed longitudinally, the panel data is used to test the hypotheses. Model 1 and Model 2 regress the dependent variable (links added to a focal file from version t to version t+1) on "modularity (direct)" and "cyclic group size" as independent variables. Please note that the level of analysis is the micro-level of design element, i.e. files in the software system. A set of control variables, as suggested in the software engineering literature, is added to control for the differences in content of files. This is important for isolating the effect of hypothesized independent variables from other sources of variations that is generated by the differences in content of files.

The results find evidence for all hypotheses. While both design modularity and design hierarchy are significantly associated with change in the complexity of the architecture, the effect of design hierarchy highly dominates that of modularity. Results are robust with and without the most significant control variables, the language of file. Consistent with findings of other research in the context of software industry, the results here confirm that Java files, compared to C files, demonstrate different behavior in influencing the change in software systems (Baldwin et al., 2014). Consequently, the R-square of the model 2, compared to model 1, improves for about one-third.

Similarly, in Models 3 and 4, the dependent variable (links added to a focal file from version t to version t+1) is regressed on modularity and cyclic group size. However this time the modularity

is calculated based on number of links that enter the focal file, indirectly. Again, the results support all three hypotheses, indicating that results still hold when the indirect links are considered. More interestingly, the increase in R-square indicates that by considering indirect links more of the variation in dependent variable could be explained.

Insert table 3 about here

Robustness checks.

A series of robustness checks conducted to examine the validity of results over partially different identification strategies. First, we checked the interacting effects of two independent variables. Not a significant interaction between the influence of modularity and hierarchy on the dependent variables observed. This reconfirms the theoretical assumption that the two independent variables measure two different aspects of design, affect the dependent variable differently, and are empirically orthogonal.

As a further check, we examined the same hypothesis considering outward links. The results reconfirmed the hypotheses and showed a significant associate between modularity and addition of new outward links, though the effect was less strong.

As the final robustness check, we tested the hypotheses with a partially different definition of modularity. To make the comparison between the effects of the two independent variables more reasonable, we used a reverse-coded measurement of modularity that captures the number of links that cross the boundary of the module and connect with elements in other modules. The effect of

the cyclic group size is still significantly stronger than the effect of modularity, to the proportion of about 15 times.

DISCUSSION AND CONCLUSION

This study examines the role of design characteristics in the architectural change of complex systems. While innovation in complex systems has long been studied in the management literature, still, insights from studying microfoundations of architecture shed more light on the microprocesses of change in complex systems. To fulfill this purpose, we theoretically explained how design modularity and design hierarchy of each element differently add to system's complexity.

Moreover, we argued that non-hierarchical elements contribute much more to the complexity of the architecture of next generation, compared to non-modular elements. Drawing on discussions and insights from organizational design literature, we contribute to the debate of innovation in complex system, and bring back in the organizational mechanisms that are manifested in the system's architecture.

Furthermore, in the empirical analyses we found support for the discussed hypotheses. A more modular element is less likely to increase its complexity in the next version. This finding at the level of design microfoundation validates arguments in design literature that are in favor of modularity at system level. The theoretical discussion argues that modularity helps the developers to limit the propagation of change inside a module and prevent it from affecting the rest of the system. The crux of the argument points to individual's cognitive limitations and bounded rationality. If the system is not modular though, a change in an element could instantly propagate to the rest of the system and affect many other elements. Then, since in a complex system checking

all the affected elements exceeds one's cognitive capacity, it causes bugs in design and deflections in the functioning of the whole system, and finally, leads to a more complex design and incurs technical debt (Sturtevant et al., 2013; Sturtevant & MacCormack, 2013).

The paper theoretically contributes to technical design and architectural innovation literature in several ways. First, it provides a framework to study the architectural innovation at the microstructural level. The insights provide theoretical tools to open up an avenue of research to uncover the micro-level mechanisms of change that could not be explained before. Second, it helps to understand how architectural design of complex systems emerges and evolves. The fact that complex systems are inert in their architecture reemphasizes the role of early design in shaping the next generations of architecture in complex systems. By disentangling different effects of design modularity and hierarchy on architectural change, this paper sets the first step to explore various aspects of complex product's design.

Moreover, the findings of this research have managerial implications. Complexity of design brings about technical debts and incurs costs of coordination, collaboration, and rework. Research shows that complexity of design is significantly associated with more deflections in functioning of the system, less productivity of the developers, and even higher rates of employees' turnover (Sturtevant et al., 2013; Sturtevant & MacCormack, 2013). Therefore, it is the first priority of managers to control the complexity of design, from a generation to the next. However, it is an arduous task, since the complex systems tend to increase in complexity as they grow in size over time (Lehman & Ramil, 2003). The findings of this study identify the elements in the design that are at risk of getting more complex in the next version. This valuable information guides managers, when they plan to redesign the system for next generation, as they have the chance to focus their limited resources to specific and more important elements of the system.

REFERENCES

Alexander, C. 1964. **Notes of the Synthesis of Form**. **Vasa**.

Baldwin, C., MacCormack, A., & Rusnak, J. 2014. Hidden structure: Using network methods to map system architecture. **Research Policy**, 43(8): 1381–1397.

Baldwin, C. Y., & Clark, K. B. 2000. **Design rules: The power of modularity**. The MIT Press.

Banker, R. D., Davis, G. B., & Slaughter, S. A. 1998. Software Development Practices, Software Complexity, and Software Maintenance Performance: A Field Study. **Management Science**, 44(4): 433–450.

Betz, S., Fricker, S., Moss, A., Afzal, W., Svahnberg, M., Wohlin, C., et al. 2013. An Evolutionary Perspective on Socio-Technical Congruence: The Rubber Band Effect. **Replication in Empirical Software Engineering Research (RESER), 2013 3rd International Workshop on**: 15–24.

Blondel, V. D., Guillaume, J.-L., Lefebvre, E., & Lambiotte, R. 2008. Fast unfolding of communities in large networks. **Journal of Statistical Mechanics: Theory and Experiment**, 2008(10).

Brusoni, S., & Prencipe, A. 2001. Managing Knowledge In Loosely Coupled Networks: Exploring The Links Between Product And Knowledge Dynamics. **Journal of Management Studies**, 38(7): 1019–1035.

Brusoni, S., & Prencipe, A. 2011. Patterns of modularization: The dynamics of product architecture in complex systems. **European Management Review**, 8: 67–80.

Cabigiosu, A., Camuffo, A., & Schilling, M. A. 2014. **You get what you measure: grounding product modularity metrics on real data analysis**. Unpublishe working paper.

Cataldo, M., Herbsleb, J., & Carley, K. 2008. Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity. **symposium on Empirical software analysis** (March): 1–20.

Colfer, L., & Baldwin, C. 2010. The mirroring hypothesis: Theory, evidence and exceptions. **Harvard Business School Finance Working.**

Conway, M. E. 1968. How do committees invent? **Datamation**.

Eppinger, S. D., Whitney, D. E., Smith, R. P., & Gebala, D. A. 1994. A model-based method for organizing tasks in product development. **Research in Engineering Design**.

Ethiraj, S. K., & Levinthal, D. 2004. Bounded rationality and the search for organizational architecture: An evolutionary perspective on the design of organizations and their evolvability. **Administrative Science Quarterly**, 49(3): 404–437.

Ethiraj, S. K., & Levinthal, D. A. 2002. Search for architecture in complex worlds: an evolutionary perspective on modularity and the emergence of dominant designs. **Wharton School, University of Pennsylvania**.

Ethiraj, S. K., Ramasubbu, N., & Krishnan, M. S. 2012. Does complexity deter customer-focus? **Strategic Management Journal**, 33(2): 137–161.

Furlan, A., Cabigiosu, A., & Camuffo, A. 2014. When the mirror gets misted up: Modularity and technological change. **Strategic Management Journal**, 35(6): 789–807.

Henderson, R., & Clark, K. 1990. Architectural innovation: the reconfiguration of existing product technologies and the failure of established firms. **Administrative science quarterly**, 35(1): 9–30.

Herbsleb, J. D. 2007. Global software engineering: The future of socio-technical coordination. **2007 Future of Software Engineering**: 188–198. IEEE Computer Society.

Huang, C. C., & Kusiak, A. 1998. Modularity in design of products and systems. **Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on**, 28(1): 66–77.

Karim, S., & Kaul, A. 2015. Structural Recombination and Innovation: Unlocking Intraorganizational Knowledge Synergy Through Structural Change. **Organization Science**, 26(2): 439–455.

Kwan, I., Cataldo, M., & Damian, D. 2012. Conway's Law Revisited: The Evidence For a Task-based Perspective. **IEEE software**, 29(1): 1–4.

Lehman, M., & Ramil, J. 2003. Software Evolution – Background , Theory , Practice. **Information Processing Letters**.

MacCormack, A., Baldwin, C., & Rusnak, J. 2012. Exploring the duality between product and organizational architectures: A test of the "mirroring" hypothesis. **Research Policy**, 41(8): 1309–1324.

MacCormack, A., Rusnak, J., & Baldwin, C. 2007. The Impact of Component Modularity on Design Evolution: Evidence from the Software Industry. **Harvard Business School Working Paper, 08-038**.

MacCormack, A., Rusnak, J., & Baldwin, C. Y. 2006. Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code. **Management Science**, 52(7): 1015–1030.

Murmann, J. P., & Frenken, K. 2006. Toward a systematic framework for research on dominant designs, technological innovations, and industrial change. **Research Policy**, 35(7): 925–952.

Ramasubbu, N., & Kemerer, C. F. forthcoming. Technical Debt and the Reliability of Enterprise Software Systems: A Competing Risks Analysis. **Management Science**, 1–48.

Rivkin, J. W., & Siggelkow, N. 2003. Balancing Search and Stability: Interdependencies Among Elements of Organizational Design. **Management Science**, 49(3): 290–311.

Sanchez, R., & Mahoney, J. T. 1996. Modularity, flexibility and knowledge management in product and organization design. **Strategic Management Journal**, 17(SI): 63–76.

Simon, H. 1962. The architecture of complexity. **Proceedings of the American Philosophical Society**, 106(6): 467–482.

Smith, R. P., & Eppinger, S. D. 1997. Identifying controlling features of engineering design iteration. **Management Science**, 43(3): 276–293.

Srikanth, K., & Puranam, P. 2011. Integrating distributed work: Comparing task design, communication, and tacit coordination mechanisms. **Strategic Management Journal**, 32(February): 849–875.

Srikanth, K., & Puranam, P. 2014. The Firm as a Coordination System: Evidence from Software Services Offshoring. **Organization Science**, (February).

Sturtevant, D., & MacCormack, A. 2013. The Impact of System Design on Developer Productivity. **Academy of Management Conference. Orlando (USA)**.

Sturtevant, D., MacCormack, A., Magee, C., & Baldwin, C. 2013. **Technical Debt in Large *Systems : Understanding the cost of software complexity***. Unpublished thesis, MIT.

Valetto, G., Helander, M., Ehrlich, K., Chulani, S., Wegman, M., & Williams, C. 2007. Using software repositories to investigate socio-technical congruence in development projects. **Proceedings of the Fourth International Workshop on Mining Software Repositories**: 25. IEEE Computer Society.

Von Hippel, E. 1990. Task partitioning: An innovation process variable. **Research Policy**.

Zhou, Y. 2013. Designing for complexity: using divisions and hierarchy to manage complex tasks. **Organization Science**, (October).

TABLE 1

Descriptive statistics

Name of Variable	N	Mean	Std. Dev.	Min	Max
Δ links (direct)[4]	37633	0.0801	1.405	-64	75
Δ links (direct & indirect)	37633	1.455	6.334	-64	100
Modularity (indirect)	41472	223.749	250.597	0.019	1223
Modularity (direct)	41472	243.850	239.533	0.156	1223
Non-hierarchy (# of reciprocal links)	41472	0.683	2.729	0	21
# of direct links	41472	6.153	27.829	1	676
# of indirect links	41472	84.086	268.506	1	1991

[4] From version t to version t+1

TABLE 2

correlation results

	Δ links (indirect)	Δ links (direct)	Non-hierarchy	Modularity (indirect)	Modularity (direct)	Indirect links	Direct links	Java files	Lines of Code	Comment/ Code	Cyclomatic Complexity
Δ links (indirect)	1										
Δ links (direct)	0.3637	1									
Non-hierarchy	0.1646	0.0434	1								
Modularity (indirect)	-0.2144	-0.0498	-0.2282	1							
Modularity (direct)	-0.1873	-0.0551	-0.2025	0.982	1						
Indirect links	0.5171	0.0972	0.2661	-0.2933	-0.258	1					
Direct links	0.2165	0.2912	0.1528	-0.1692	-0.1839	0.4289	1				
Java files	0.074	0.0213	-0.0085	0.267	0.2945	0.0898	0.028	1			
Lines of Code	-0.034	0.0057	0.0421	-0.0142	-0.0287	-0.056	0.0255	-0.1789	1		
Comment/Code	0.0028	0.022	0.0217	-0.0927	-0.0983	-0.0344	0.0036	-0.1147	-0.0617	1	
Cyclomatic Complexity	-0.0268	0.0069	0.0487	-0.0302	-0.0435	-0.0439	0.0329	-0.1846	0.9474	-0.042	1

TABLE 3

Panel data regression results of "links added"

	(1) Δ Links	(2) Δ Links	(3) Δ Links	(4) Δ Links
Modularity (direct links)	-0.00299*** (-14.79)	-0.00382*** (-18.71)		
Modularity (indirect links)			-0.00348*** (-17.83)	-0.00431*** (-21.96)
Non-hierarchy (# of reciprocal links)	0.324*** (17.07)	0.313*** (17.16)	0.306*** (16.30)	0.293*** (16.26)
Control variables				
Java Files		2.339*** (13.54)		2.413*** (14.25)
Lines of code	-0.00193* (-2.46)	-0.00162* (-2.14)	-0.00181* (-2.33)	-0.00147* (-1.97)
Ratio comment/code	-0.109 (-1.86)	-0.0264 (-0.46)	-0.117* (-2.01)	-0.0296 (-0.53)
Cyclomatic complexity	0.00437 (0.92)	0.00621 (1.36)	0.00381 (0.81)	0.00569 (1.26)
Constant	2.009*** (24.88)	0.00938 (0.06)	2.069*** (26.93)	-0.0190 (-0.11)
Observations	37633	37633	37633	37633
R^2	0.1431	0.1960	0.1669	0.2235

t statistics in parentheses, * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$