10TH INTERNATIONAL DESIGN STRUCTURE MATRIX CONFERENCE, DSM'08
11 -12 NOVEMBER 2008, STOCKHOLM, SWEDEN

A DYNAMIC, DSM-BASED VIEW OF SOFTWARE ARCHITECTURES AND THEIR IMPACT ON QUALITY AND INNOVATION

Manuel E. Sosa[1], Tyson R. Browning[2], and Jürgen Mihm[1]

[1]INSEAD, Fontainebleau, France
[2]Neeley School of Business, Texas Christian University, Fort Worth, Texas, USA

Keywords: product architecture, modularity, software development, organizational capabilities

1 INTRODUCTION

Previous work has studied the implications of product architecture decisions to various aspects of the firm [e.g., 1,7]. However, little attention has been devoted to understand the link between product architecture characteristics and product performance. How does the architecture of a system influence its quality and innovative features? More specifically, what are the system architecture features that managers need to pay attention to in order to prevent quality issues from emerging? What system architecture features would influence the capability of the development team to improve the system over time? These questions are particularly relevant in the development of adaptive systems such as software products which are developed in an additive manner over successive generations [2,3,5].

We address the questions posed above by examining the architecture of software products because they are complex, exhibit fast change rates (like fruit flies in studies of biological evolution), and offer (through their source code) an efficient, reliable, and standardized medium to capture their architecture. Specifically, we examine architectural data as well as data about "bugs" and "new features" available over successive versions of several software applications developed by the open-source foundation, Apache. We provide a structured approach and a set of measures to examine the architecture of software products not only to predict the number of bugs in the next product release but also the capability of the development team to fix bugs as well as to add new features. Our results have theoretical and managerial implications for the development of complex adaptive systems.

This abstract provides only a brief introduction to our work. Our presentation extends the work discussed by [5] by empirically testing the relationship between software architecture features and the fixing of bugs as well as the creation of new features.

2 A FRAMEWORK TO STUDY THE IMPACT OF SOFTWARE ARCHITECTURE ON QUALITY AND INNOVATION

In order to understand how the architecture of a complex, evolving system, such as software products, influences its quality and innovative features, it is crucial to take a dynamic perspective on the development process. Figure 1 shows a simplified view of two successive releases of a software application.

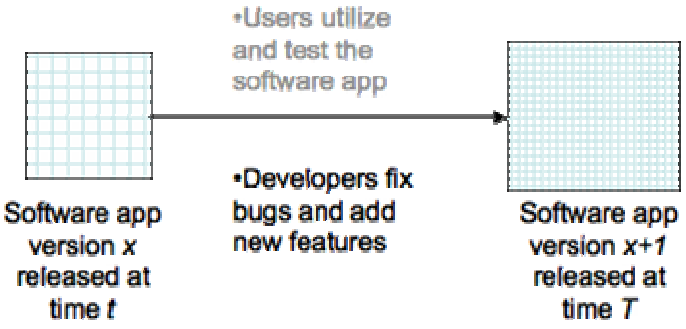


Figure 1. A Dynamic View of Software Development

From a user's perspective, software applications provide certain functionality and capability. As long as the application provides these reliably and without consuming too many resources (cost, user friendliness, computer memory, or disk space), the user is generally satisfied. However, that is rarely the case. Users typically uncover "bugs" and request new functionalities that become evident after testing and using version *x* of the software application. From a designer's standpoint, there are many alternative ways for the software to provide the specified functionality. Designers or architects must determine how to allocate the software's functions to its various components or groups thereof, called modules. Architects must also determine how the software system will be organized in terms of command and control modules and components, utilities and other supporting infrastructure components and modules. These choices determine the nature and extent of the relationships between components and modules of any version of the software application. These relationships affect the ease with which components and modules can be changed in successive versions [2,3,5].

The software application version *x* released at time *t* has an architecture that, eventually, conditions how components and modules interact to deliver the specified functionality [2,4,5]. We argue that there are certain measurable architectural features that predict the deviation between the actual and specified functionality of the product. In addition, because the product architecture at time *t* provides the main platform used by developers to fix its bugs as well as to add novel features to it, it also conditions the capability of the development team to improve version *x*+1 to be released at time *T*.

Figure 2 illustrates graphically the framework we aim to test empirically. Using a DSM-based representation of the software architecture of version *x* we measure various architectural features that are supposed to influence not only the number of bugs associated with version *x* but also the capability of the organization to fix those bugs as well as to develop novel features (to be released in the next version, *x*+1). From an empirical viewpoint, it is interesting to realize that the number of bugs associated with version *x* could also influence the bug fixing and innovative capacity of the development team working on the next release, *x*+1. Testing the model shown in Figure 2 has important managerial implications because it could guide managers to examine the architectural features that matter for quality and innovation of software products.

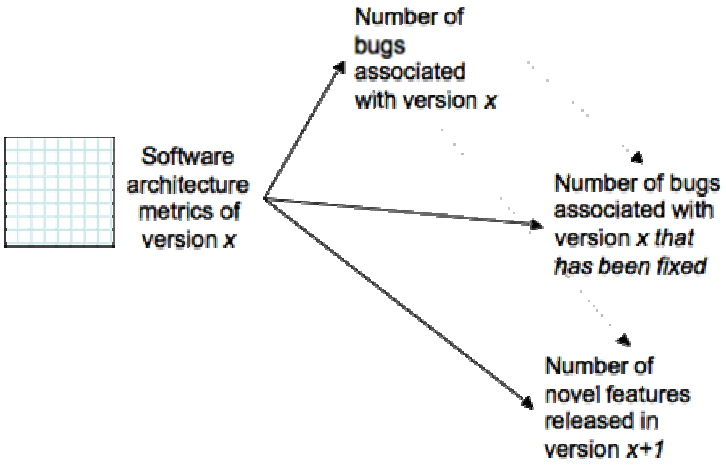


Figure 2. A Simple Theoretical Model for Empirical Testing

3 THE EMPIRICAL STUDY

We carry out an empirical study to test whether and how the architecture of software applications influences their quality and innovative features. Our empirical approach is structured in three steps:

1. Capture the architecture of the software application for successive versions.
2. Capture the number of bugs associated with each version released.
3. Capture the number of novel features associated with each version released.

In order to address our research questions by applying our structured approach, we study readily-accessible, open-source, Java-based software applications from the Apache foundation. We collected information from various public sources. (All the information used to build our database is available at www.apache.org.) We examined all the Java-based applications developed by Apache that would allow us to apply all three steps of our approach. Those were applications which we could access, for

successive major releases, their source code (to codify product architecture features), their bug reports (to determine number of bugs and bug fixing capability associated with each product version), and their release notes (to determine the number of innovative features included on each version of the product). After data purification, we compiled a set of 22 applications with an average of five versions each.

For each version, developers made up to three types of modifications from the previous version: "changes," "tasks," and "bug fixes." *Changes*, as the term is used by Apache, signify a new and better way of implementing an existing feature or capability of the software. That is, the feature was not necessarily performing incorrectly (which would be a bug), but someone found a way to provide the feature more efficiently or effectively—or, the way the feature was implemented had to be adjusted to accommodate some other change, bug fix, or task. *Tasks* are new features or capabilities added to the current version. *Bug fixes* are corrections of existing features that were not performing correctly.

To measure software architectures features, we first need to represent how the components of the product interact, how they are grouped into modules, and how modules are organized into a hierarchy. To capture the basic features that characterize complex system architectures, we use two complementary representations: a hierarchy tree and a partitioned product DSM [6]. A tree representation indicates module membership and layering, whereas a product DSM captures the interactions between components both within and across modules. We invite the interested reader to refer to [5] for a full description of how we use these representations to capture the architecture of software applications.

4 DISCUSSION

In our presentation we provide further discussion of our approach and statistical analysis carried out to test the model shown in Figure 2. We find empirical evidence that shows a significant and positive association between our simple measures based on the amount of coupling between the elements of a software application and the number of bugs associated with it. We will also discuss the implications of this result for quality and innovation management in software development.

REFERENCES

1. Baldwin, C. and K. B. Clark, *Design Rules: The Power of Modularity*, vol. 1. Cambridge, MA: MIT Press, 2000.
2. MacCormack, A., J. Rusnak, and C. Y. Baldwin, "Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code," *Management Science*, vol. 52, pp. 1015-1030, 2006.
3. Parnas, D.L. On the Criteria To Be Used in Decomposing Systems into Modules. *Communications of the ACM*, 15(12), 1053-1058, 1972.
4. Sangal, N., Jordan, E., Sinha, V. and Jackson, D. Using Dependency Models to Manage Complex Software Architecture. *20th ACM SIGPLAN Conf. on Object-Oriented Programming, Systems, Languages And Applications (OOPSLA)*, pp. 167-176 San Diego, CA, 2005).
5. Sosa, M.E., Browning, T.R. and Mihm, J. Studying the Dynamics of the Architecture of Software Products. *ASME 2007 International Design Engineering Technical Conf. & Computers and Information in Engineering Conf. (IDETC/CIE 2007)* Las Vegas, NV, 2007).
6. Steward, D.V. The Design Structure System: A Method for Managing the Design of Complex Systems. *IEEE Transactions on Engineering Management*, 28(3), 71-74, 1981.
7. Ulrich, K. "The Role of Product Architecture in the Manufacturing Firm," *Research Policy*, vol. 24, pp. 419-440, 1995.

Contact: Manuel Sosa
Associate Professor of Technology and Operations Management
INSEAD
Boulevard de Constance
77305 Fontainebleau
France
+33 1 60 72 45 36
manuel.sosa@insead.edu
http://faculty.insead.edu/sosa/personal/

10TH INTERNATIONAL DSM CONFERENCE

A Dynamic, DSM-Based View of Software Architectures and Their Impact on Quality and Innovation

Manuel E. Sosa
INSEAD, Fontainebleau, France

Tyson Browning
Neeley School of Business, Texas Christian University, USA
Jürgen Mihm
INSEAD, Fontainebleau, France

Technische Universität München

FOLLOW THE FLOW OF INFORMATION!
MANAGE COMPLEX SYSTEMS

Quality in Software Development

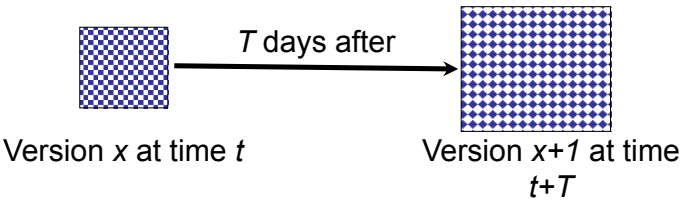
- Bugs and Bug Fixing
  - Bugs are “defects” in a software application that prevents it to conform with its functional requirements
  - Most bugs are *produced* when writing source code, many of them are *identified* (by testers and users), and many of them are *fixed* in successive releases of the application

Technische Universität München

Innovation in Software Development

- Software applications are developed in a flexible manner (Parnas 1979, MacCormack et al. 2001)
- New and improved functionality are added over successive generations of an application
- Two types of innovations
  - *Incremental* changes to improve existing software functionality
  - *Radical* changes to add new features/functionality

Research Questions

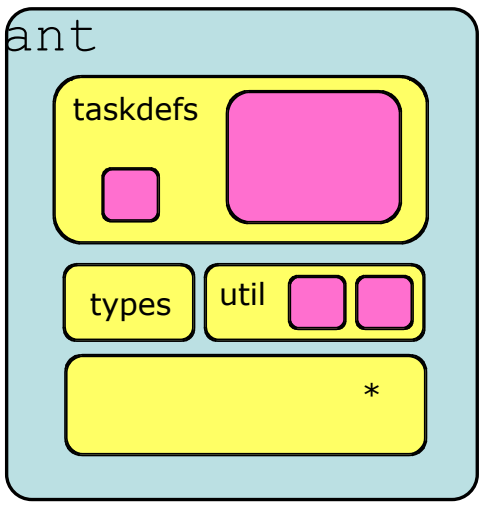


- How does the system architecture influence quality and innovation in software development?
  - Which characteristics of the architecture in version *x* predict the number of **bugs** in version *x+1*?
  - What characteristics of the architecture in version *x* influence the capability of the organization to **fix bugs** and **improve** version *x+1*?

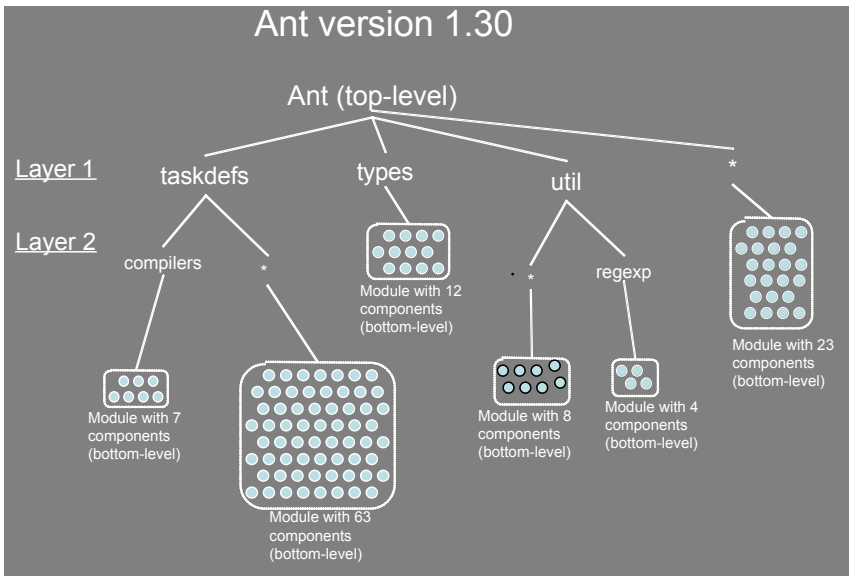
Software architecture

- The scheme by which the elements of the source code are arranged to deliver the functionality that is required (Parnas 1972, Ulrich 1995)
- Source code of software application as a collection of interdependent components organized in a hierarchical manner
(Sangal et al. 2005, MacCormack et al. 2006, Sosa et al. 2007)

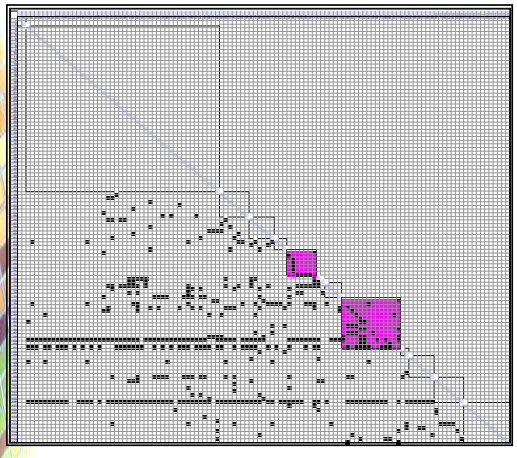
Ant 1.30



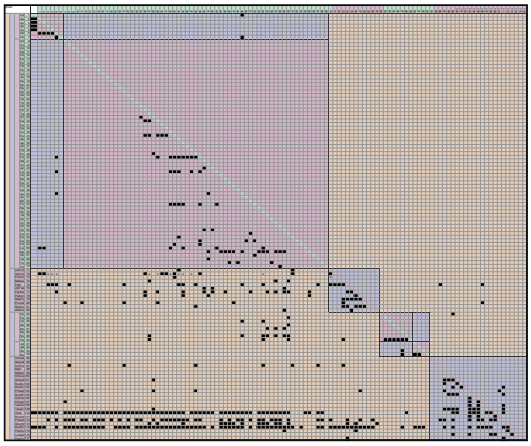
Hierarchical View



Two DSM Representations



A Flat Product DSM

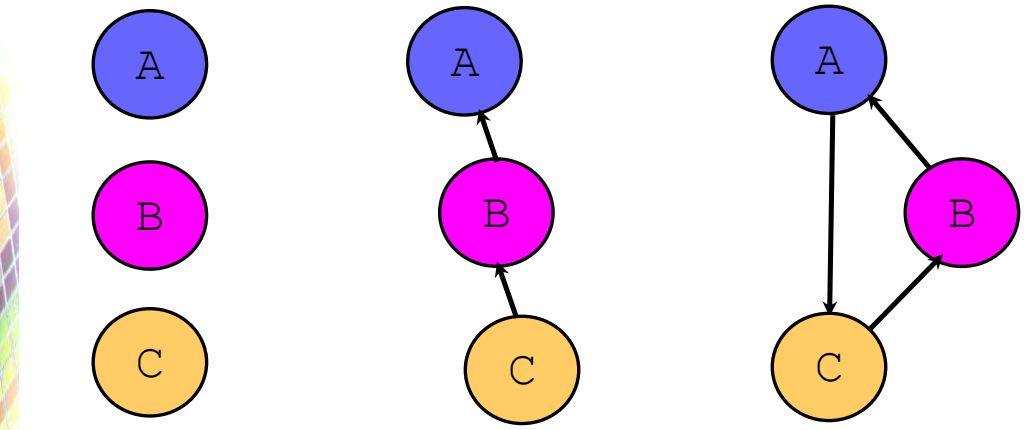


A Hierarchical Product DSM

Architectural Characteristics

- Number of components (*N*)
- Number of interactions (*K*)
- Complexity (*NK*), Density (*K/(N(N-1)*)
- Connectedness
  - Propagation cost (MacCormack et al. 2006)
    - Average probability that any two components are (directly or indirectly) connected
- Coupling
  - The section(s) of the architecture in which the components are involved in coupled interdependence (i.e. design loops)

Types of Interdependence



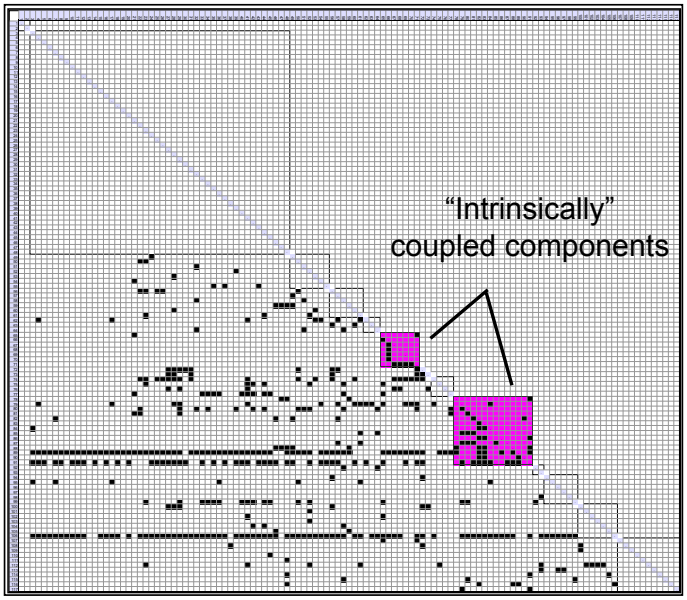
No Interdependence

Serial Interdependence

Coupled Interdependence

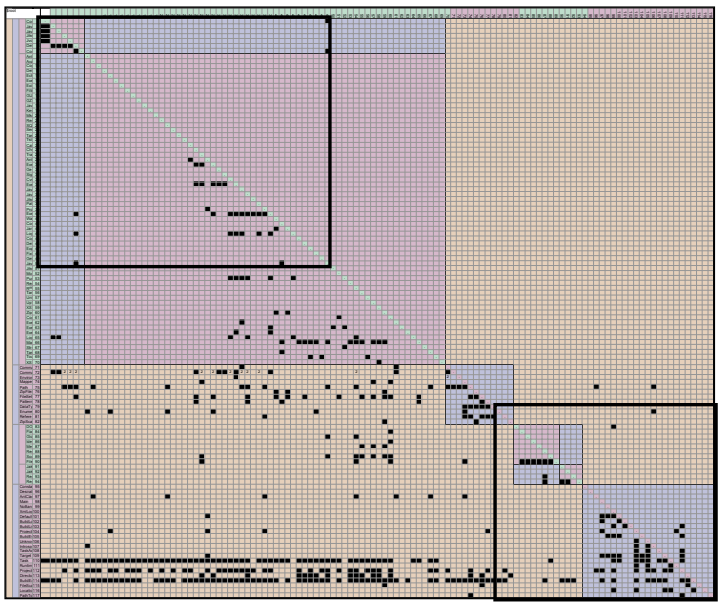
Intrinsic Coupling

The number of components involved in design iterations in a unconstrained “flat” product DSM



Actual Coupling

The number of components involved in design iterations in a hierarchical constrained product DSM



The Effects of Coupling

- Coupling in the product architecture is associated with iterative problem solving (Smith and Eppinger 1997, Mihm et al. 2003)
- Developers involved in design iterations typically make assumptions to solve their technical problems
  - If assumptions were not correct, either the source code is reworked or bugs are likely to be generated
- Iterative problem solving associated with coupling in the product architecture is likely to propagate changes iteratively and consume more resources than expected

FOLLOW THE FLOW OF INFORMATION!
MANAGE COMPLEX SYSTEMS

The Effects of Connectivity

- The more interconnected the elements in a system the less modular it is and the easier changes propagate through it (Clarkson et al. 2004, Sosa et al. 2007)
- Excessive interconnectedness is redundancy in the structure of the source code which demands for improvement actions (MacCormack et al. 2006)

Three Hypotheses

- *Coupling* in version *x* is positively associated with number of *bugs* in version *x+1*
- *Coupling* in version *x* is negatively associated with number of *bugs fixed* in version *x+1*
- *Connectedness* in version *x* is positively associated with the level of *improvements* included in version *x+1*

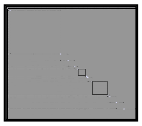
FOLLOW THE FLOW OF INFORMATION!
MANAGE COMPLEX SYSTEMS

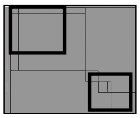
The Apache Software Foundation
http://www.apache.org/

- **The applications**
  - Open-source Java-based tools for automating software development
  - 20 applications with complete data about
    - Source code to capture the software architecture
    - Bug reports to capture the number of bugs and bug fixes
    - Release notes to capture the incremental and radical changes
  - On average, 7 versions per applications

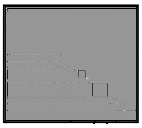


Architectural Measures

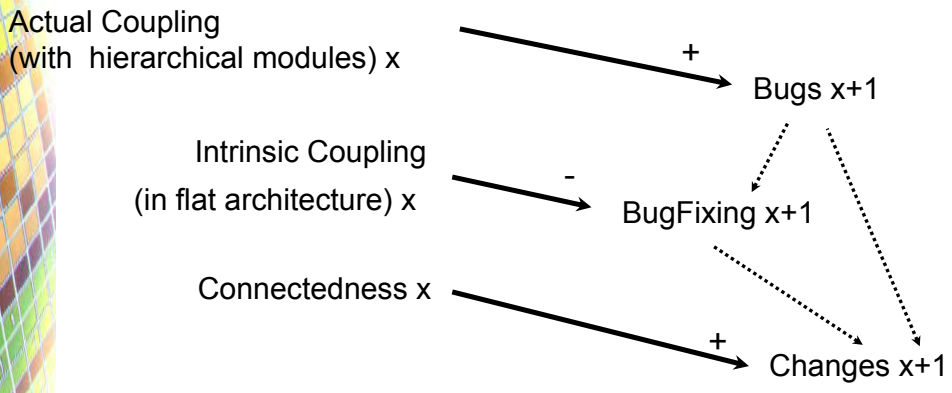




- Coupling
  - Number of groups of coupled components
  - Number of elements involved in design iterations
- Connectedness
  - Propagation cost = probability that component *i* can reach component *j*



Putting All Together



Results

	Bugs	BugFixing	Changes
Total number of bugs		0.817***	0.371
Total number of bugs fixed			-0.398
Age: Days from 1st release	0.054**	-0.006	-0.019**
Days to nxt release	0.131***	-0.034**	0.007
Newness	0.568**	-0.094	-0.080
# nominal modules	-2.700**	-0.557	-2.125***
N	-1.005**	-0.279***	0.646***
K	-0.117*	0.083***	-0.096***
Network Density	-4007***	-522	-1002!
# of coupled groups with hierarchies	**30.503*** **		
# of coupled comps with hierarchies	**1.605*** **		
# coupled groups flat		**2.618** **	
# of coupled comps flat		**-0.771*** **	
Connectedness: propagation cost			**418.6*** **
Age x Connectedness			**-0.139!**
Adj R^2	0.7159	0.9714	0.5698

Obs= 108 * < .1 ** <.05 ***<.001
All models include fixed effects for each of the 20 applications

FOLLOW THE FLOW OF INFORMATION!
MANAGE COMPLEX SYSTEMS

Conclusions

- Can we predict bugs, bug fixing, and changes for improvement of software systems by looking at its architecture?
  - Yes!

✓ **Actual coupling** of the architecture of a system **positively** influences its number of **bugs**

✓ **Intrinsic coupling** of the architecture of a system **negatively** influences the capability to **fix bugs**

✓ **The internal interconnectedness** of the architecture of a system **positively** influences the need to **continuously improve** it.