9TH INTERNATIONAL DESIGN STRUCTURE MATRIX CONFERENCE, DSM'07
16 – 18 OCTOBER 2007, MUNICH, GERMANY

DYNAMIC, DSM-BASED ANALYSIS OF SOFTWARE PRODUCT ARCHITECTURES

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, complexity, organizational structure, software development

1 INTRODUCTION

We explore how the architecture of a product evolves over several generations. We propose a theoretical framework and research approach to study the dynamics of complex product architectures. We illustrate our approach 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. The IEEE defines product architecture as "the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution" [1]. In the software domain, architecting involves organizing or structuring the code into modules and layers with the appropriate set of dependencies between them [2, 3].

This paper reports results from [4], in which we provide a theoretical framework, a basic set of metrics, and a research approach for exploring the dynamics of complex software architectures. Then, based on empirical evidence from a case study of an open source project, we uncover several patterns and insights regarding the dynamics of software architectures and their relationships to organizational dynamics. These findings indicate several promising avenues for future research.

To explore the dynamics of complex software architectures, we structure our research approach in three steps:

1. Capture the evolution of software architecture properties.
2. Capture the evolution of organizational attributes.
3. Compare the dynamics of product architectures and organizational attributes.

This abstract provides only a brief introduction to our approach. Our presentation will include metrics and results. Additional discussion is also available in [4].

2 REPRESENTING SOFTWARE ARCHITECTURE

To measure the complexity associated with software architectures, 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. A tree representation indicates module membership and layering, whereas a product DSM captures the interactions between components both within and across modules.

Figure 1 shows the tree representation of one of the versions of the software product we study in this paper, Ant 1.3. The tree representation shows how the 126 components comprising this version of the product are organized into eight modules and three layers.

In the software domain, a DSM representation has been used to capture the interactions between "class functions" that comprise software applications [5-7]. Typically, the rows and columns in a product DSM are ordered so as to maximize the density of clusters of components along the diagonal, so that clusters (modules) encapsulate the majority of interfaces. This approach, called clustering [8], is generally recommended for hardware products because of the highly symmetric nature of many spatial and structural design dependencies between physical components [9]. However, when analyzing the architecture of software products, we instead use the clusters defined by the system architects and partition (triangularize) the DSM, also called sequencing [8], to uncover the dependencies that define the truly coupled components.

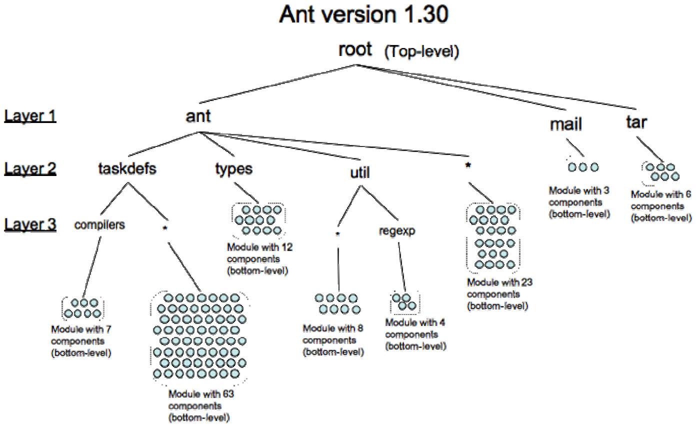

Ant version 1.30
root (Top-level)
Layer 1
ant
mail
tar
Layer 2
taskdefs
types
util
*
Layer 3
compilers
*
regexp
*
Module with 7 components (bottom-level)
Module with 63 components (bottom-level)
Module with 12 components (bottom-level)
Module with 8 components (bottom-level)
Module with 4 components (bottom-level)
Module with 23 components (bottom-level)
Module with 3 components (bottom-level)
Module with 6 components (bottom-level)

Figure 1. Tree diagram of Ant version 1.3 (©2007 ASME)

Those familiar with DSM techniques will notice two innovations here. First, we are applying a sequencing algorithm to a component-based DSM, a combination which did not exist [8] prior to the work by Sangal et al. [6]. Second, we reverse the typical order of dependency in the DSM. Traditionally, a DSM using the convention where the components labeling the columns depend on the components labeling the rows would show feedback below the diagonal. This is done because, as is conventional in software, the "higher level" components are said to depend on the "lower level" ones for functionality, and, unlike other time-based DSM applications to date, all of the components indeed exist simultaneously.

In a complex software product with several layers, like in **Error! Reference source not found.**, we partition the DSM layer by layer so that modules within the same layer are arranged so as to minimize super-diagonal marks. (To sequence within each layer, we use the algorithm originally proposed by Steward [10].) **Error! Reference source not found.** shows a DSM representation of Ant 1.3. The DSM shown is a 126x126 matrix with 476 off-diagonal marks representing the "calls" between the 126 "classes" that comprise Ant 1.30. The DSM is sequenced by layer so that feedback marks above the diagonal are minimized both within and across modules. This DSM has 12 marks above the diagonal, six of them in layer 2 within module ("ant"—"*") and six of them across modules (four within layer 2 and two within layer 3. Note that the branches of the tree in **Error! Reference source not found.** are arranged to correspond to the sequenced DSM. The branches on the left of the tree depend on the branches on the right. Figure 3 provides a condensed DSM.

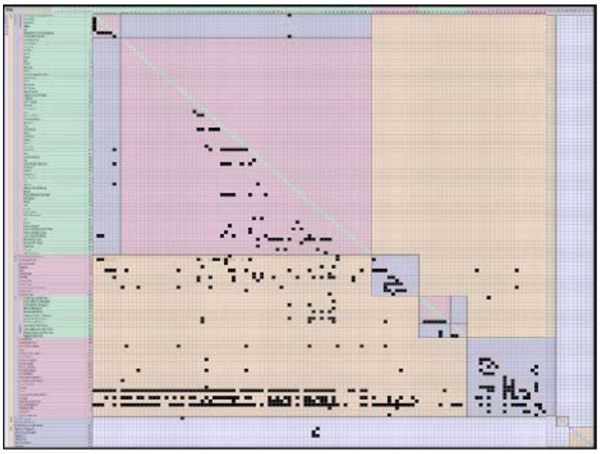
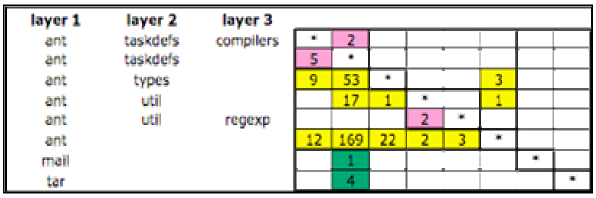


Figure 2. Complete DSM for Ant version 1.3 (©2007 ASME)



layer 1	layer 2	layer 3								
ant	taskdefs	compilers	*	2						
ant	taskdefs		5	*						
ant	types		9	53	*			3		
ant	util			17	1	*		1		
ant	util	regexp				2	*			
ant			12	169	22	2	3	*		
mail				1					*	
tar				4						*

Figure 3. Condensed DSM for Ant version 1.3 (©2007 ASME)

3 APPROACH, METRICS, AND RESULTS

In our presentation and in [4] we provide further discussion of our approach, static and dynamic complexity metrics, and results for seven generations of the Apache Ant application. Our analysis suggests that the architecture of a new product does not magically emerge in the first version. Rather, establishing the architecture of the product is a dynamic process that goes through distinct phases which require different managerial competences.

REFERENCES

[1] IEEE. IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. (Institute of Electrical and Electronics Engineers Standards Association, 2000).

[2] Parnas, D.L. On the Criteria To Be Used in Decomposing Systems into Modules. *Communications of the ACM*, 1972, 15(12), 1053-1058.

[3] Shaw, M. and Garlan, D. *Software Architecture: Perspectives on an Emerging Discipline*. (Prentice Hall, Upper Saddle River, NJ, 1996).

[4] 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).

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

[6] 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).

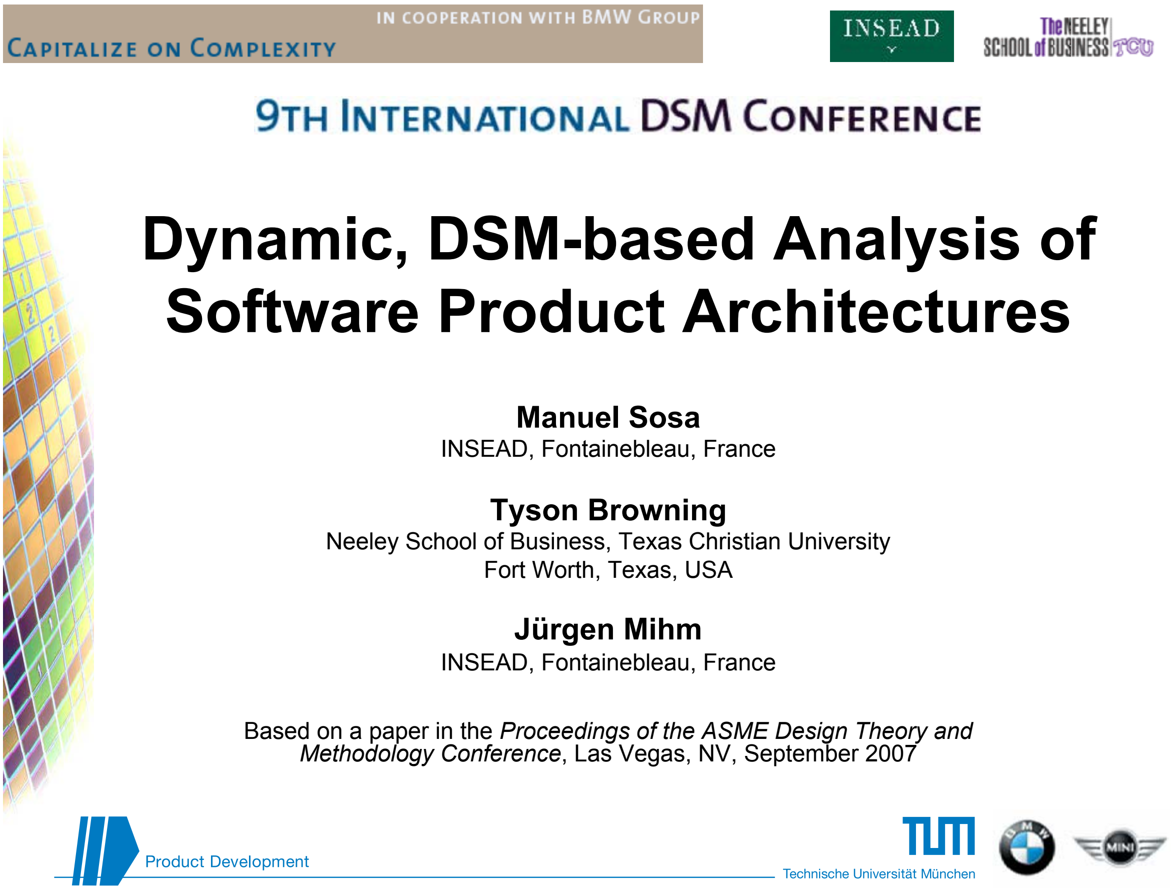
[7] Sullivan, K.J., Griswold, W.G., Cai, Y. and Hallen, B. The Structure and Value of Modularity in Software Design. *ACM SIGSOFT Software Engineering Notes*, 2001, 26(5), 99-108.

[8] Browning, T.R. Applying the Design Structure Matrix to System Decomposition and Integration Problems: A Review and New Directions. *IEEE Transactions on Engineering Management*, 2001, 48(3), 292-306.

[9] Sosa, M.E., Eppinger, S.D. and Rowles, C.M. A Network Approach to Define Modularity of Components in Product Design. *Journal of Mechanical Design*, 2007(forthcoming).

[10] Steward, D.V. The Design Structure System: A Method for Managing the Design of Complex Systems. *IEEE Transactions on Engineering Management*, 1981, 28(3), 71-74.

Contact: Tyson R. Browning
Neeley School of Business, Texas Christian University
Department of Information Systems and Supply Chain Management
Fort Worth, Texas
USA
817-257-5069
t.browning@tcu.edu
www.tysonbrowning.com



CAPITALIZE ON COMPLEXITY
IN COOPERATION WITH BMW GROUP

9TH INTERNATIONAL DSM CONFERENCE

Dynamic, DSM-based Analysis of Software Product Architectures

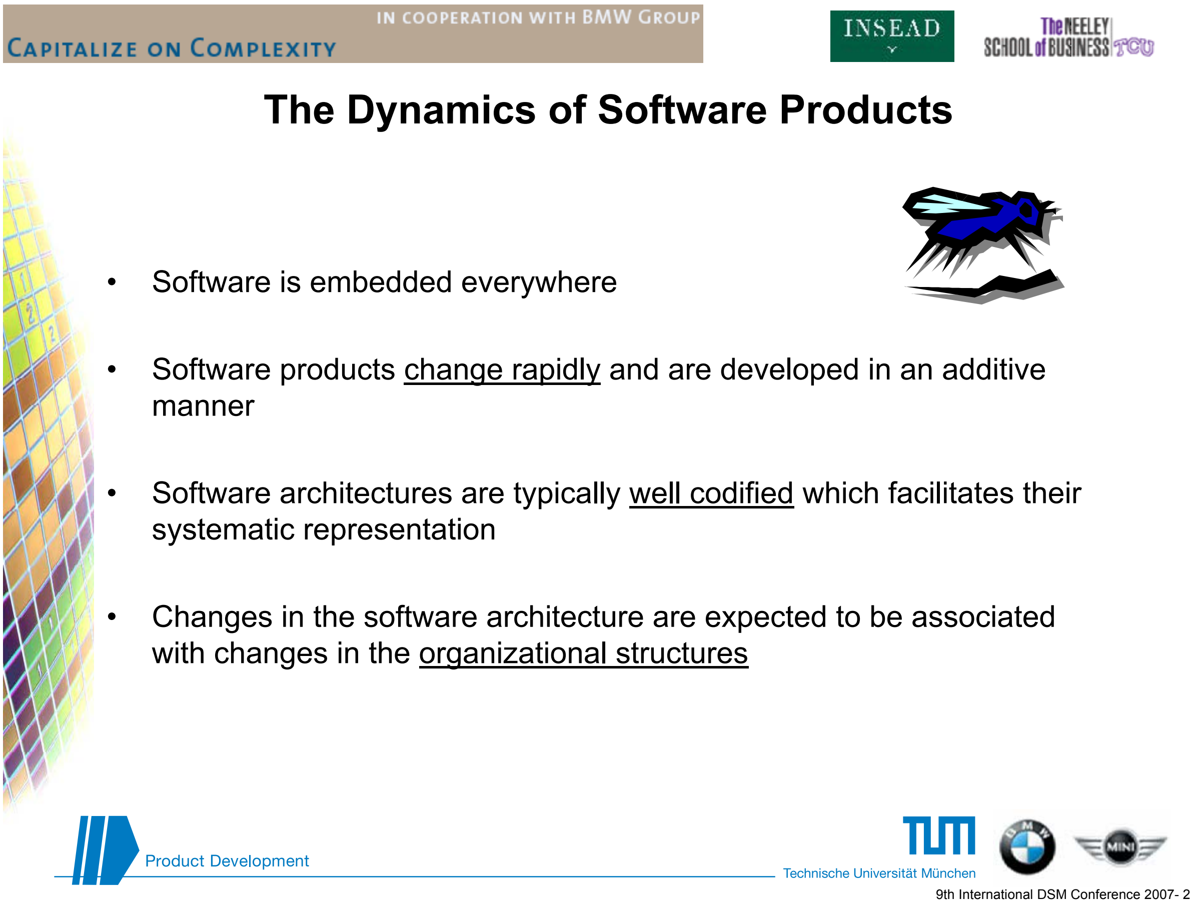
Manuel Sosa
INSEAD, Fontainebleau, France

Tyson Browning
Neeley School of Business, Texas Christian University
Fort Worth, Texas, USA

Jürgen Mihm
INSEAD, Fontainebleau, France

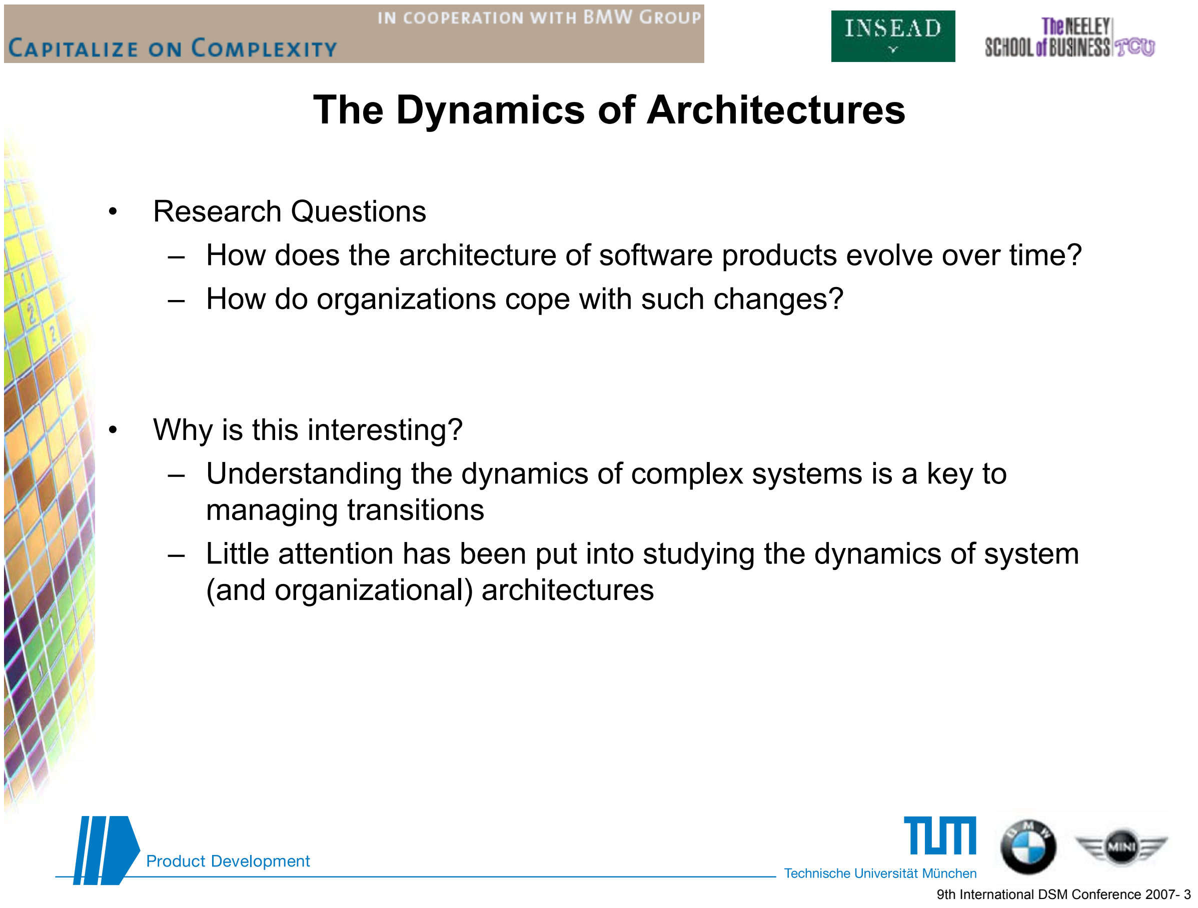
Based on a paper in the *Proceedings of the ASME Design Theory and Methodology Conference*, Las Vegas, NV, September 2007

Product Development
Technische Universität München



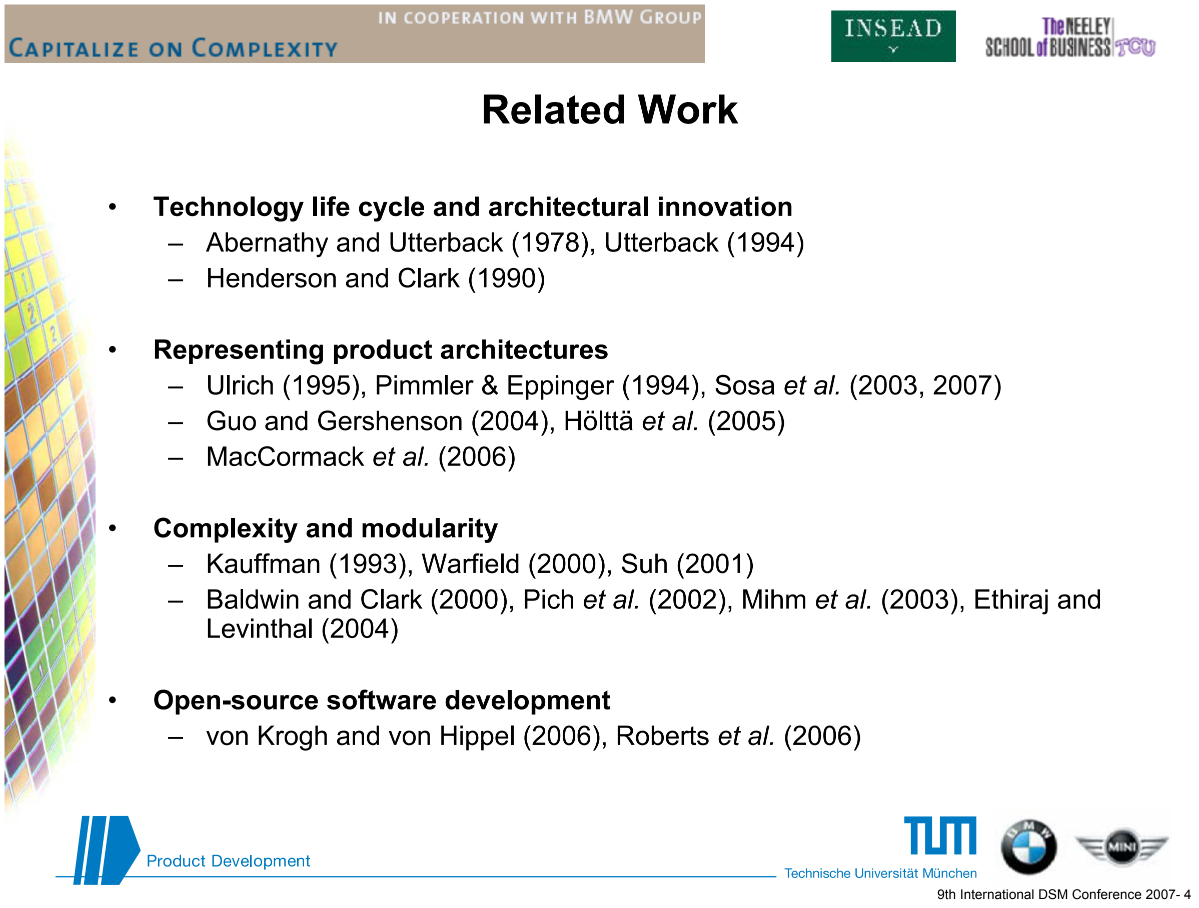
The Dynamics of Software Products

- Software is embedded everywhere
- Software products change rapidly and are developed in an additive manner
- Software architectures are typically well codified which facilitates their systematic representation
- Changes in the software architecture are expected to be associated with changes in the organizational structures



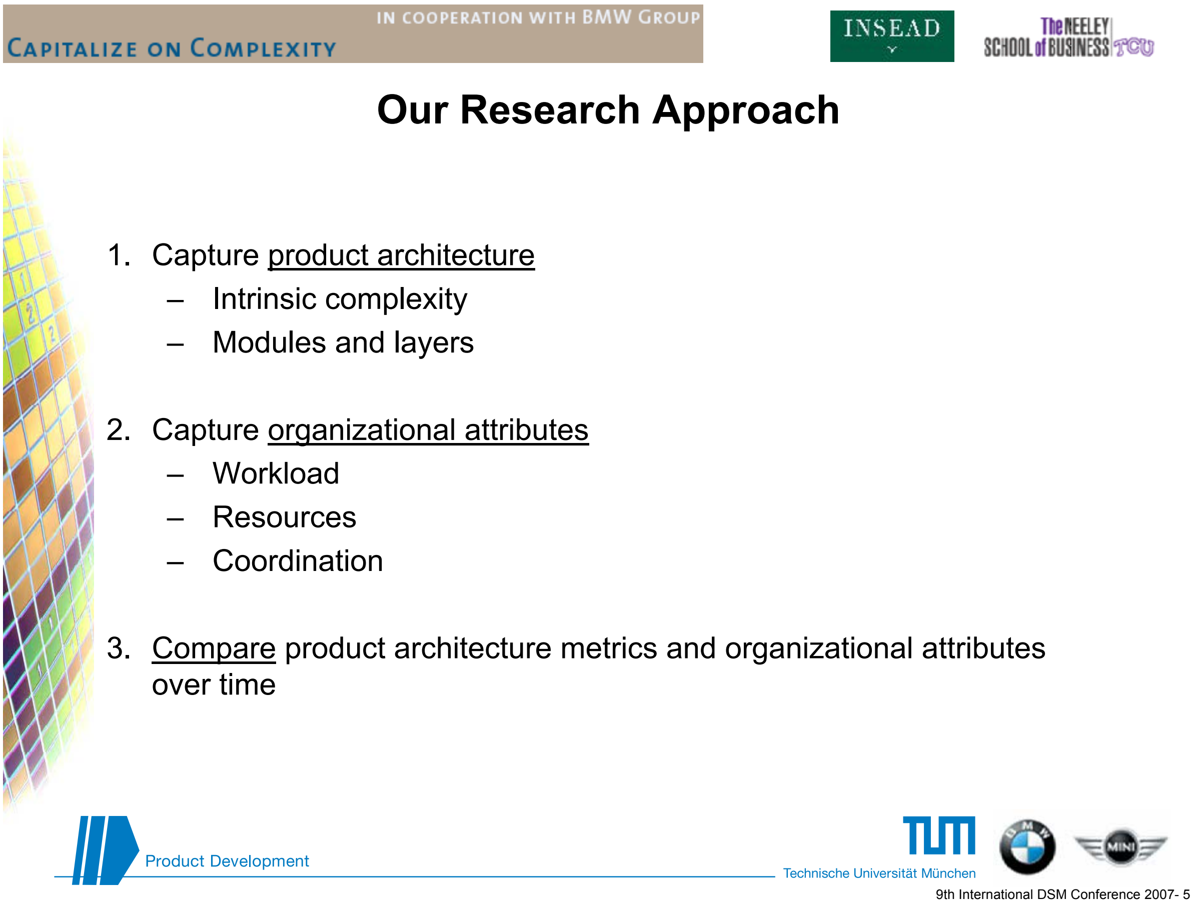
The Dynamics of Architectures

- Research Questions
  - How does the architecture of software products evolve over time?
  - How do organizations cope with such changes?
- Why is this interesting?
  - Understanding the dynamics of complex systems is a key to managing transitions
  - Little attention has been put into studying the dynamics of system (and organizational) architectures



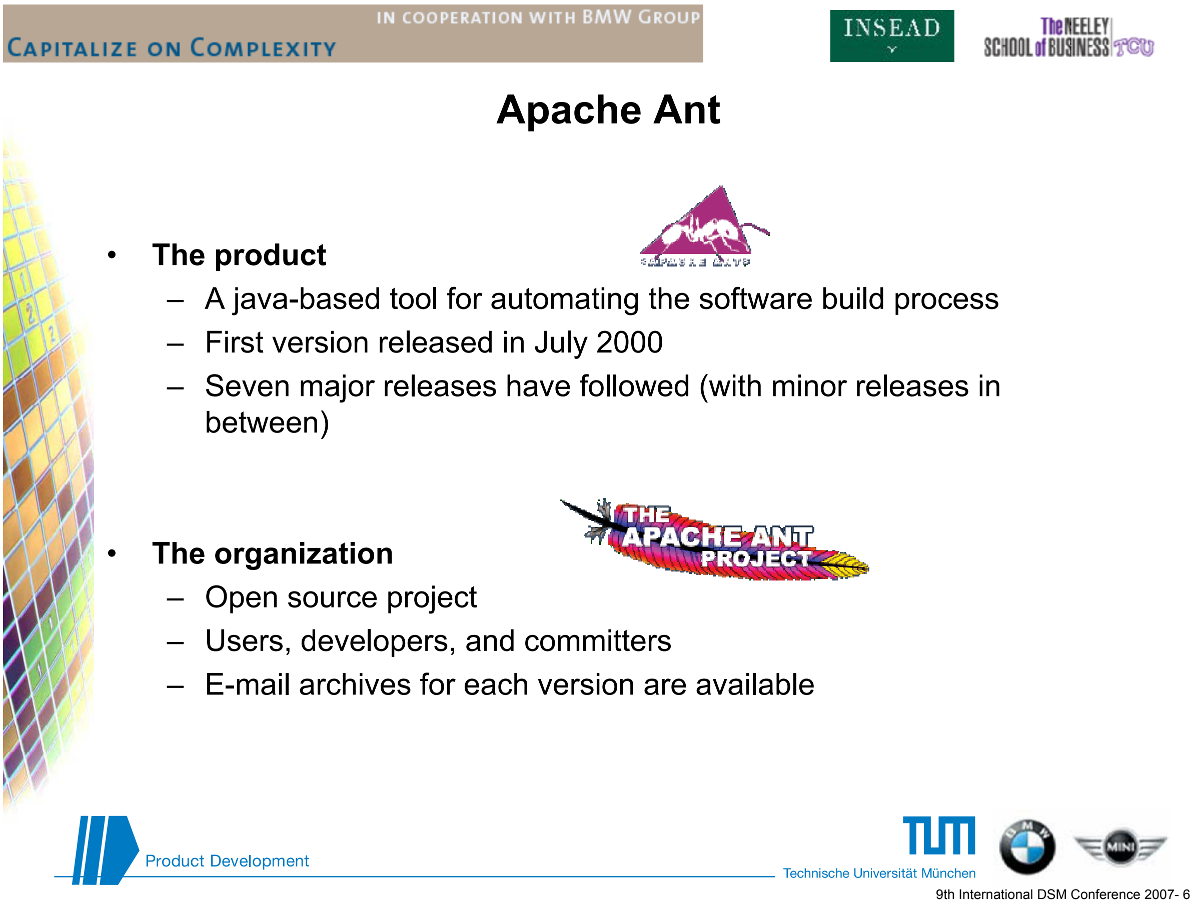
Related Work

- **Technology life cycle and architectural innovation**
  - Abernathy and Utterback (1978), Utterback (1994)
  - Henderson and Clark (1990)
- **Representing product architectures**
  - Ulrich (1995), Pimmler & Eppinger (1994), Sosa *et al.* (2003, 2007)
  - Guo and Gershenson (2004), Hölttä *et al.* (2005)
  - MacCormack *et al.* (2006)
- **Complexity and modularity**
  - Kauffman (1993), Warfield (2000), Suh (2001)
  - Baldwin and Clark (2000), Pich *et al.* (2002), Mihm *et al.* (2003), Ethiraj and Levinthal (2004)
- **Open-source software development**
  - von Krogh and von Hippel (2006), Roberts *et al.* (2006)



Our Research Approach

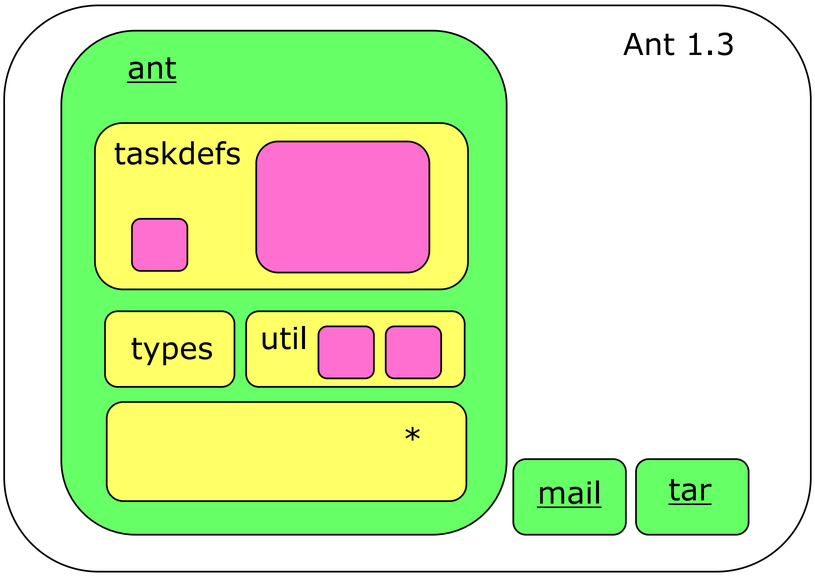
1. Capture product architecture
   - Intrinsic complexity
   - Modules and layers
2. Capture organizational attributes
   - Workload
   - Resources
   - Coordination
3. Compare product architecture metrics and organizational attributes over time



Apache Ant

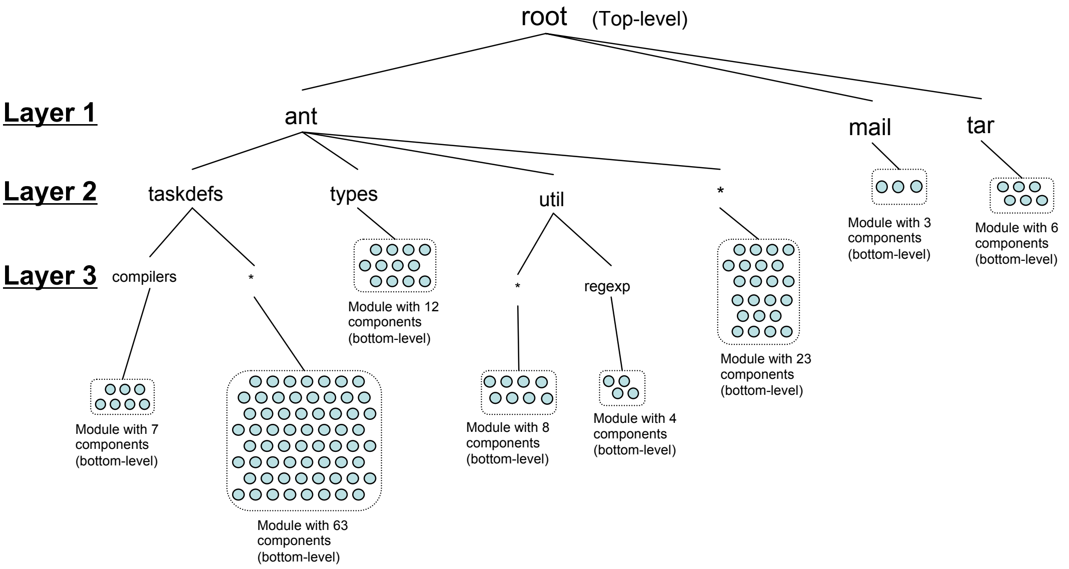
- **The product**
  - A java-based tool for automating the software build process
  - First version released in July 2000
  - Seven major releases have followed (with minor releases in between)
- **The organization**
  - Open source project
  - Users, developers, and committers
  - E-mail archives for each version are available

Ant 1.3

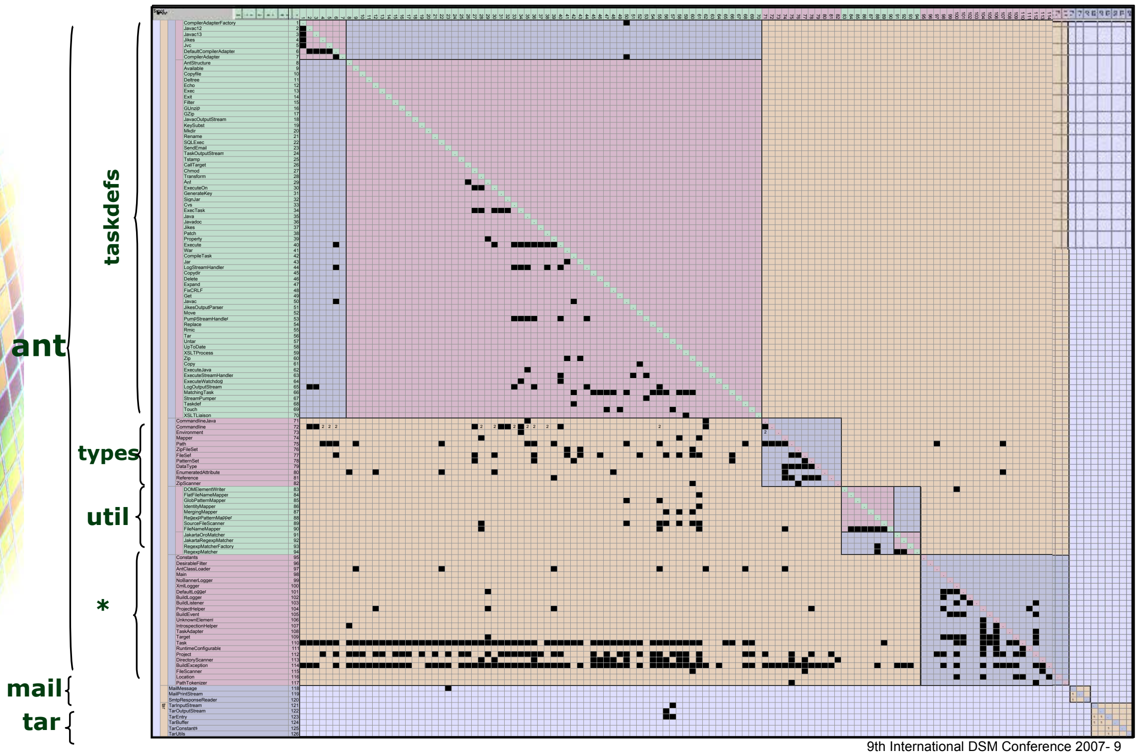


Tree Representation: Layers and Modules

Ant version 1.3


root (Top-level)
Layer 1
Layer 2
Layer 3
ant
mail
tar
taskdefs
types
util
*
compilers
*
*
regexp
Module with 3 components (bottom-level)
Module with 6 components (bottom-level)
Module with 12 components (bottom-level)
Module with 23 components (bottom-level)
Module with 7 components (bottom-level)
Module with 63 components (bottom-level)
Module with 8 components (bottom-level)
Module with 4 components (bottom-level)

Matrix Representation: Layers, Modules, and Interfaces



Architectural Metrics

- Intrinsic complexity

  Number of elements and their interactions

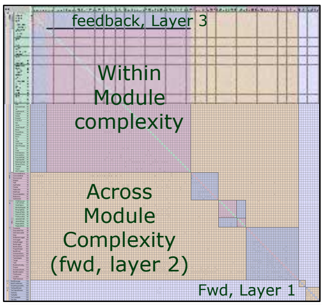
  $C_1 = n{*}k$

- The effects of modules and layers

  Avg complexity **within** module

  Avg complexity **across** modules (per layer)

$$\textit{cross-module complexity} = m_{\substack{\textit{feedforward}\\ \textit{interactions}}}$$



Look inside the architecture!

- Dynamic architectural changes between versions
  - Total number of modules in version x, (N_x)
  - Proportion of modules **added** in version x, ($D_{a,x}$)
  - Proportion of modules **eliminated** in version x, ($D_{e,x}$)

$$D_{a,x} \quad \frac{A_x}{N_x} \qquad\qquad D_{e,x} \quad \frac{E_x}{N_x}$$

Aggregated DSM Representation (Ant 1.3)

layer 1	layer 2	layer 3								
ant	taskdefs	compilers	*	2						
ant	taskdefs		5	*						
ant	types		9	53	*			3		
ant	util			17	1	*		1		
ant	util	regexp				2	*			
ant			12	169	22	2	3	*		
mail				1					*	
tar				4						*

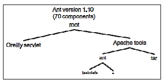
🟩 Interfaces at layer 1

🟨 Interfaces at layer 2

🟥 Interfaces at layer 3

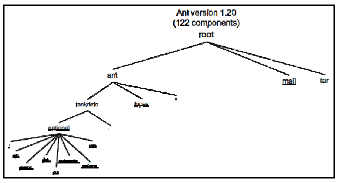
Product Architecture Dynamics

Ant 1.1



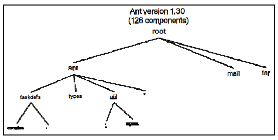
layer 1	layer 2	layer 3				
oreilly servlet			*			
apache tools	ant	taskdefs		*		
apache tools	ant			124	*	
apache tools	tar			4		*

Ant 1.2



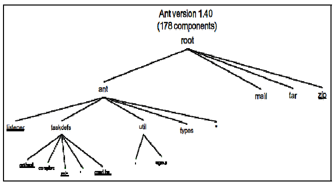
layer 1	layer 2	layer 3	layer 4													
ant	taskdefs	optional		*												
ant	taskdefs	optional	ejb		*											
ant	taskdefs	optional	javacc			*										
ant	taskdefs	optional	jlink				*									
ant	taskdefs	optional	jsp					*								
ant	taskdefs	optional	metamata						*							
ant	taskdefs	optional	perforce							*						
ant	taskdefs	optional	vss								*					
ant	taskdefs			4	2	3	1	2	2	1	2	*				
ant	types			3	1	6	1	1	3		3	40	*	3		
ant				12	9	5	2	4	3	1	6	166	17	*		
mail												1			*	
tar												4				*

Ant 1.3



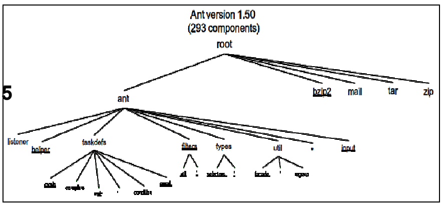
layer 1	layer 2	layer 3								
ant	taskdefs	compilers	*	2						
ant	taskdefs		5	*						
ant	types		9	53	*			3		
ant	util			17	1	*		1		
ant	util	regexp				2	*			
ant			12	169	22	2	3	*		
mail				1					*	
tar				4						*

Ant 1.4

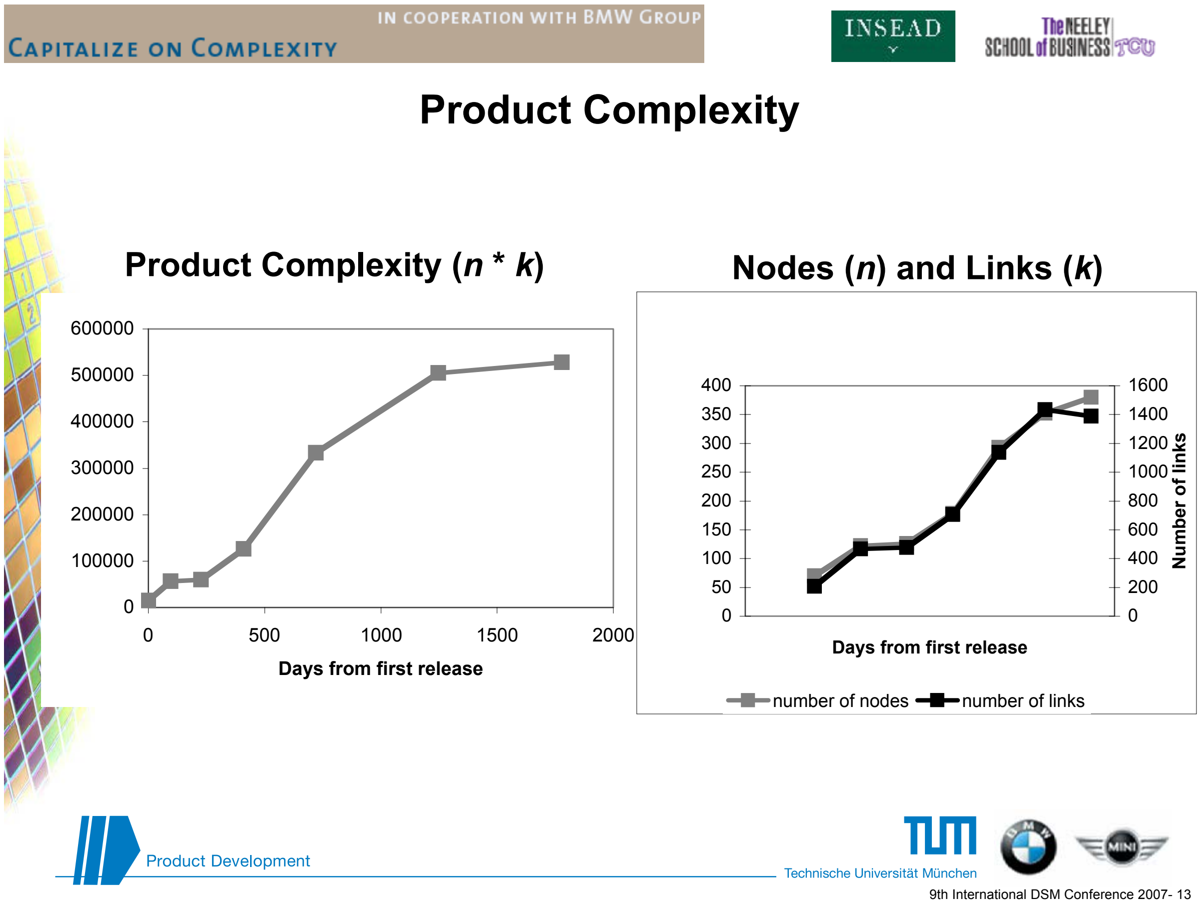


layer 1	layer 2	layer 3													
ant	listener		*												
ant	taskdefs	optional		*											
ant	taskdefs	compilers			*		2								
ant	taskdefs	rmic				*	2								
ant	taskdefs			1	5	2	*								
ant	taskdefs	condition					4	*							
ant	util					1	20		*		1	2			
ant	util	regexp							2	*					
ant	types				15	6	68		1		*	5			
ant			4		19	10	248	4	4	3	30	*			
mail							1						*		
tar							4							*	
zip							2								*

Ant 1.5



layer 1	layer 2	layer 3																						
ant	listener		*																				
ant	helper			*														1					
ant	taskdefs	cvslib			*																		
ant	taskdefs	compilers				*		2															
ant	taskdefs	rmic					*	2															
ant	taskdefs				4	6	3	*															
ant	taskdefs	condition						12	*				2			3		1					
ant	taskdefs	email						1		*													
ant	filters	util						2			*					1							
ant	filters							1			2	*											
ant	types	selectors						1					*	2				3					
ant	types				2	14	5	92		2	3	11	16	*	1	1		5					
ant	util	facade						4							*								
ant	util		1	1	1	3	2	49	1		1	1	4	5		*		12					
ant	util	regexp										1		2		2	*						
ant			4	12	11	19	11	294	17	8	3	1	13	45		6	4	*	2				
ant	input							3										2	*				
bzip2								4												*			
mail			1							1											*		
tar								4														*	
zip								2															*



CAPITALIZE ON COMPLEXITY

IN COOPERATION WITH BMW GROUP

INSEAD

The NEELEY SCHOOL of BUSINESS TCU

Product Complexity

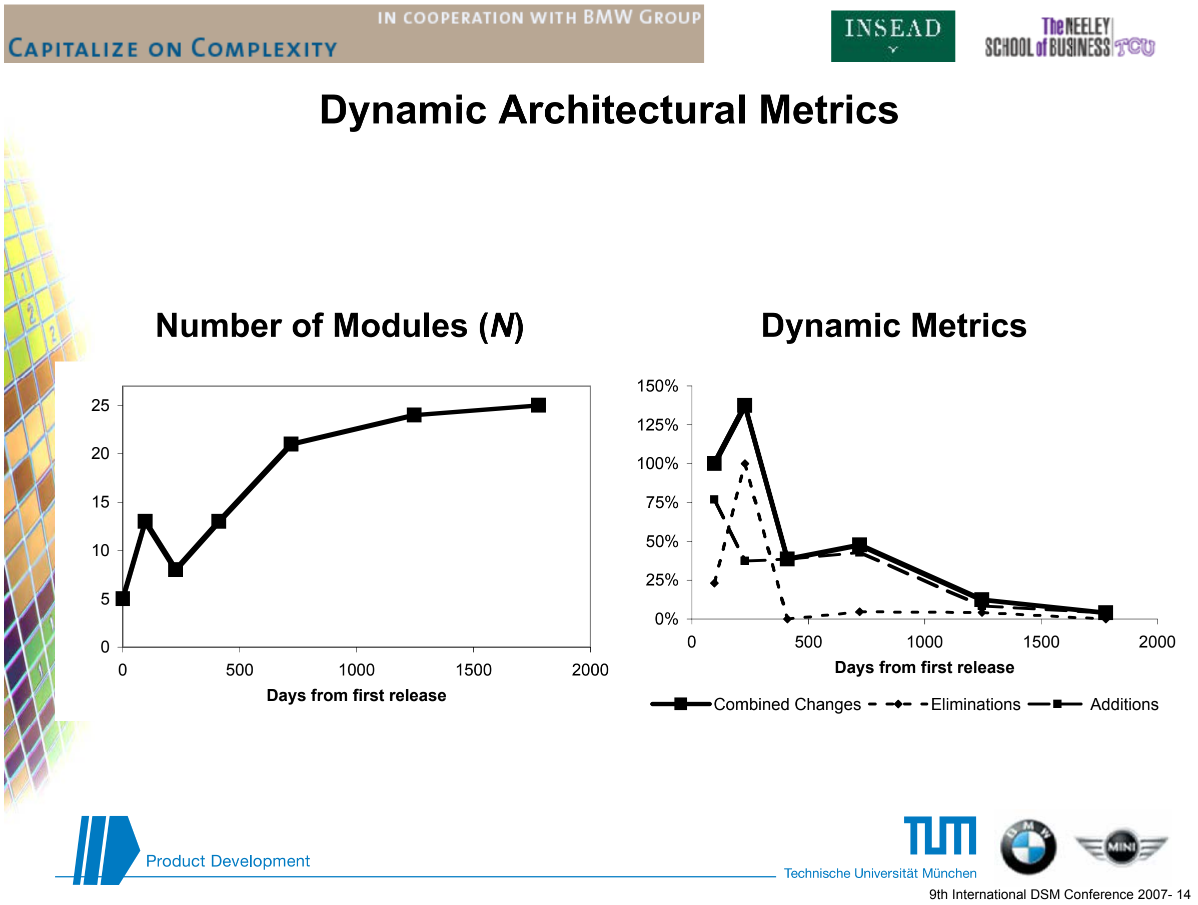
Product Complexity (*n* * *k*)

Nodes (*n*) and Links (*k*)

Product Development

Technische Universität München

9th International DSM Conference 2007- 13



CAPITALIZE ON COMPLEXITY

IN COOPERATION WITH BMW GROUP

INSEAD

The NEELEY SCHOOL of BUSINESS TCU

Dynamic Architectural Metrics

Number of Modules (*N*)

Dynamic Metrics

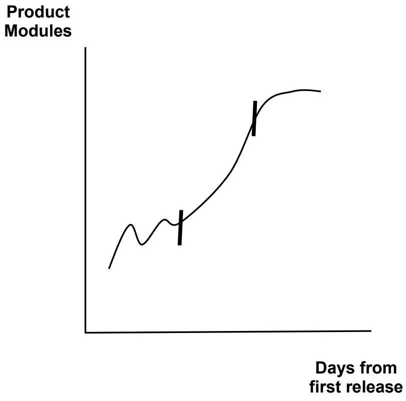
Product Development

Technische Universität München

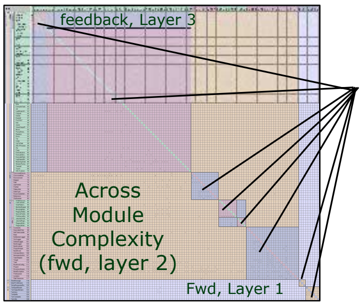
9th International DSM Conference 2007- 14

A Model of Architectural Evolution

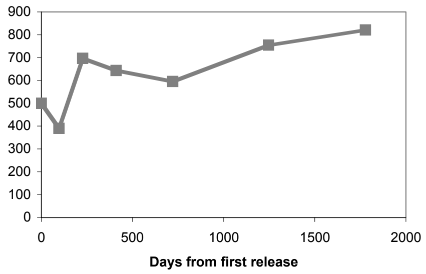
- *Formation* phase
  - Searching for the “optimal” architecture
    - Ant 1.1, 1.2, 1.3
    - “Dominant” architecture establishes in Ant 1.3, 1.4
- *Growth* phase
  - Product grows rapidly
    - Ant 1.4, 1.5, 1.6
- *Saturation* phase
  - Limits to growth appear
    - Ant 1.6, 1.65



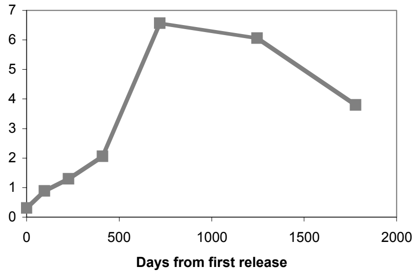
Looking Inside the Architecture

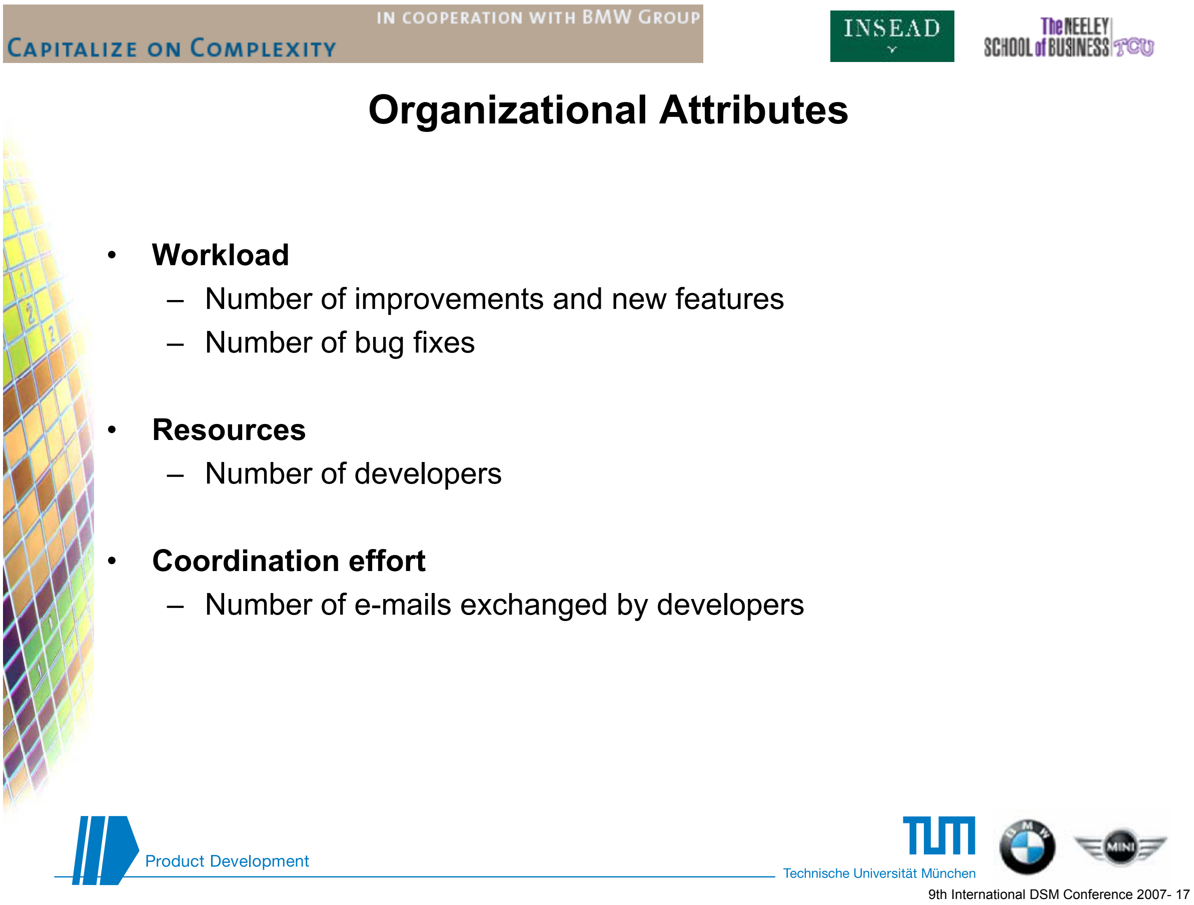


Within Module Complexity



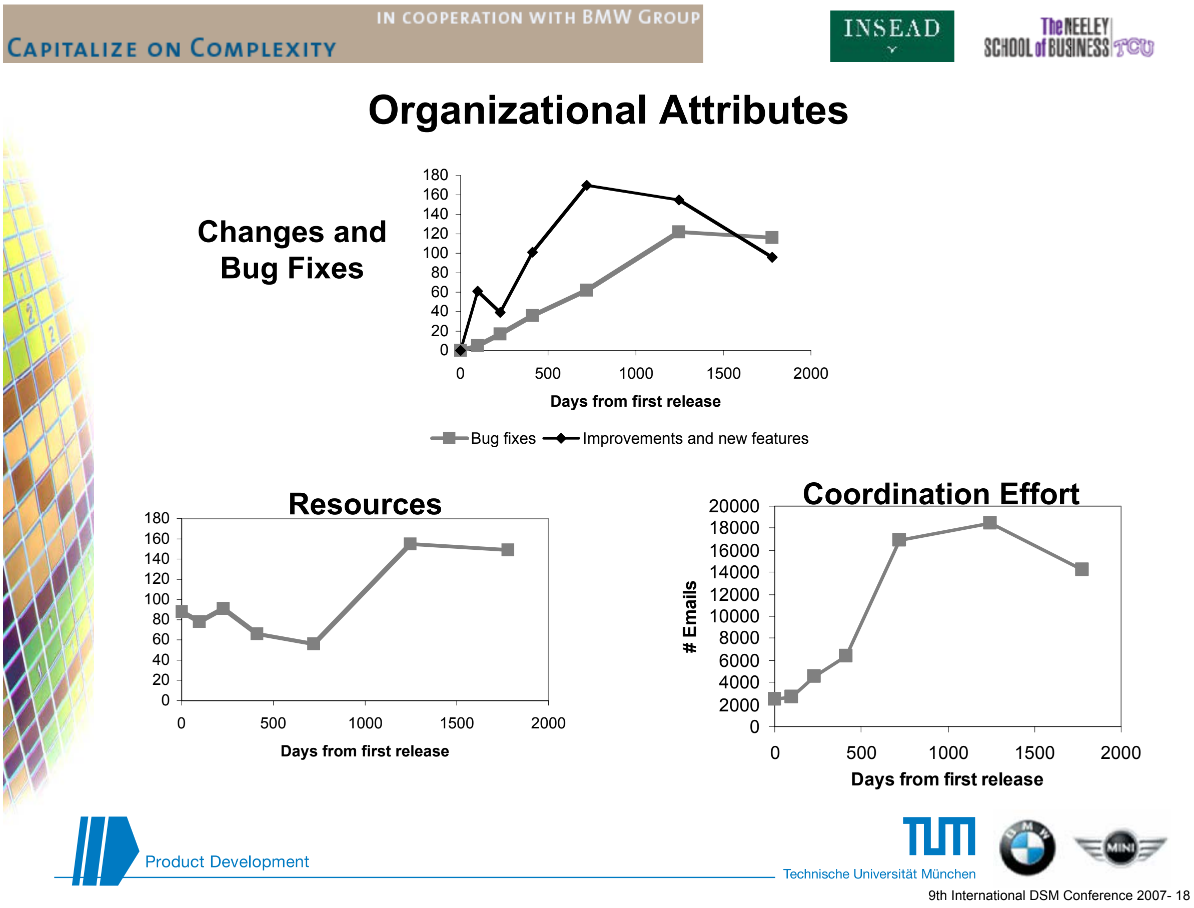
Across Module Complexity





Organizational Attributes

- **Workload**
  - Number of improvements and new features
  - Number of bug fixes
- **Resources**
  - Number of developers
- **Coordination effort**
  - Number of e-mails exchanged by developers



Organizational Attributes

Changes and Bug Fixes

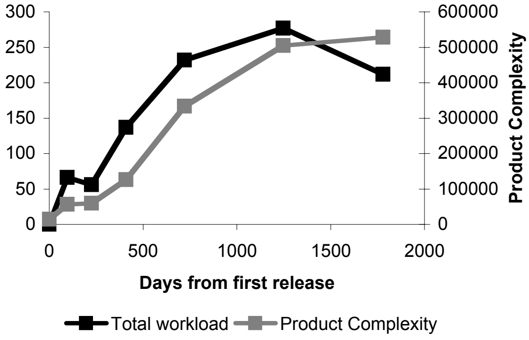
Resources

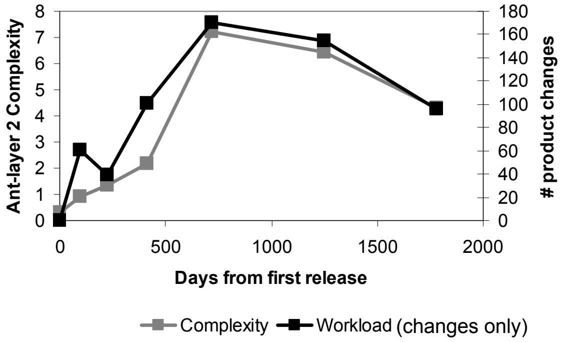
Coordination Effort

Complexity and Workload

- Total workload
- Product complexity

- Product changes
- Cross-module complexity





Conclusions

- We introduced a structured approach and simple metrics to explore the dynamics of software architectures
- The architecture of (software) products evolves through distinct phases instead of magically appearing
- By looking inside the architecture, we found evidence of the co-evolution of product and organizational structures