Advanced Topics of Software Engineering (ASE)

Chapter 2. From requirements to system design

Prof. Dr. Florian Matthes, Prof. Dr. Alexander Pretschner

Chair of Software Engineering for Business Information Systems (sebis)
Faculty of Informatics
Technische Universität München
wwwmatthes.in.tum.de

From requirements to system design

Dependency structure matrix (DSM)

A two-dimensional matrix representing the structural or functional interrelationships of objects, tasks or teams

- Support for modelling dependency in large software architectures
- Dependencies between different parts of the software can be presented in a matrix form

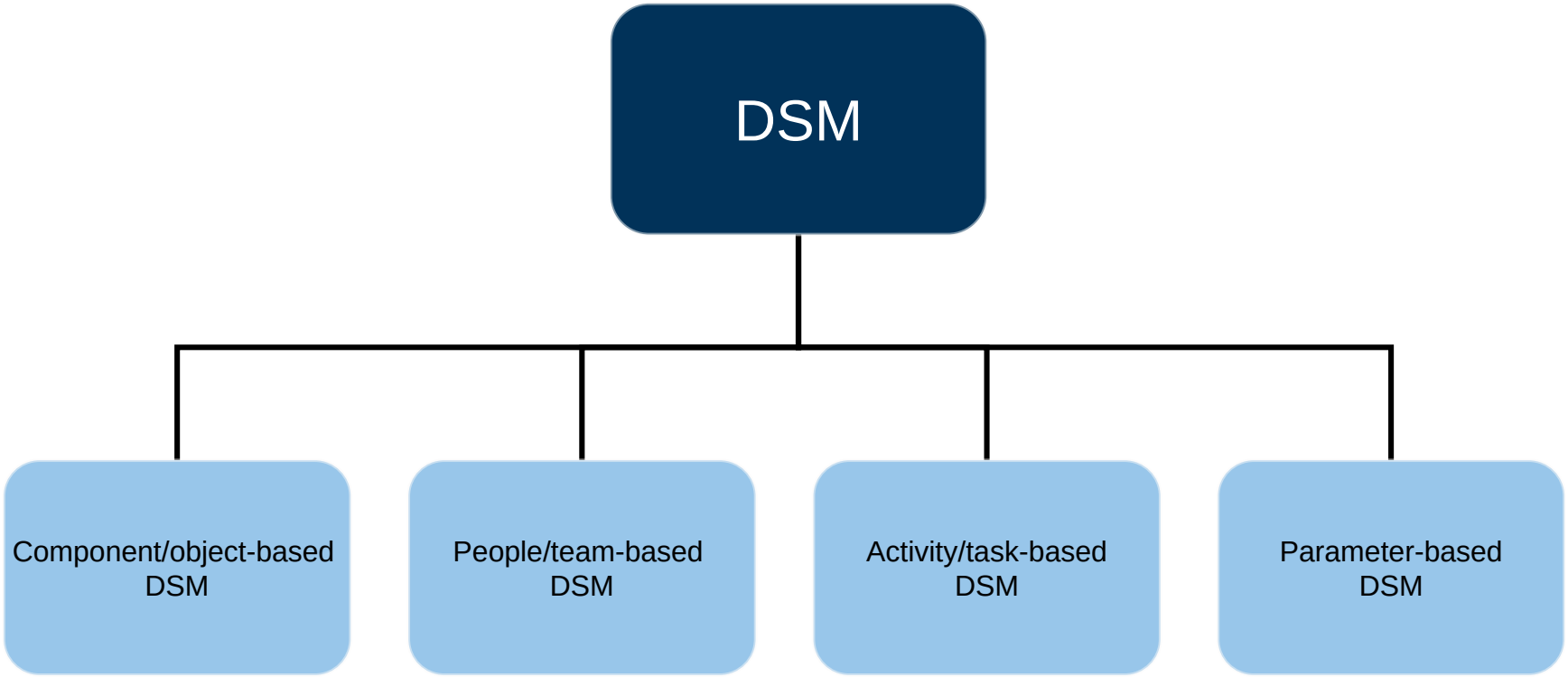
- In a DSM the items being analyzed make up the rows and columns of a two-dimensional matrix

An entry in the matrix indicates that the item on the corresponding column depends on the item on the corresponding row

- Also known as (depending on the context of application)
  - design structure matrix
  - dependency structure method
  - dependency source matrix
  - problem solving matrix (PSM)
  - incidence matrix
  - N2matrix
  - interaction matrix
  - dependency map
  - design precedence matrix

["Using dependency models to manage complex software architecture." Sangal et al. (2005)]

Categorization of DSM



A simple DSM

	A	B	C	D
A	-		X	X
B		-	X	
C	X		-	X
D				

- A depends on C
- C depends on A and B
- D depends on A and C
- B does not depend on any of them

Column depends on rows

- This DSM is binary
  - Shows if there is dependency between two items
- All dependencies are of equal weight
  - The crosses (X) can be replaced with numbers that show the exact dependency weights

Block triangular DSM after partitioning

	D	A	C	B
D	-			
A	X	-	X	
C	X	X	-	
B			X	

- A and C are mutually dependent
- A and C can be regarded as a single *composite* entity (component/task)
- This transformation is known as ***partitioning***
- A DSM which has been rearranged so that all dependencies either fall below the diagonal or within groups, is said to be in ***block triangular*** form

Lower triangular DSM

	D	A-C	B
D	-		
A-C	X	-	
B		X	-

- Ensures dependency relations are acyclic
  - No entries above the diagonal
- By considering A and C as a single composite task, the cycle can be eliminated

Hierarchical DSM

		D	A	C	B
D		-			
A-C	A	X	-	X	
	C	X	X	-	
B				X	

- Introduces hierarchical structure
- Identities of the basic entities (components/tasks) can be retained
- The grouping of A and C is shown by their indentation

Detecting patterns in DSM

Layered software systems

$root			1	2	3	4	5
org.example	application	1	-				
	model	2	X	-			
	domain	3	X	X	-		
	framework	4	X	X	X	-	
	util	5	X	X	X	X	-

["Using dependency models to manage complex software architecture." Sangal et al. (2005)]

Detecting patterns in DSM

Strictly layered software systems

$root			1	2	3	4	5
org.example	application	1	-				
	model	2	X	-			
	domain	3		X	-		
	framework	4			X	-	
	util	5				X	-

["Using dependency models to manage complex software architecture." Sangal et al. (2005)]

Detecting patterns in DSM

Layered system example - *JUnit*

			1	2	3	4	5	6
junit	awtui	1	-					
	swingui	2		-				
	textui	3			-			
	extensions	4		1		-		
	runner	5	3	8	4		-	
	framework	6	5	7	6	6	5	-

- Junit version 3.8.1
- Layered system with clean separation of the user interface layers from the underlying core logic
- Notice the numbers: they indicate the number of dependencies

["Using dependency models to manage complex software architecture." Sangal et al. (2005)]

Detecting antipatterns in DSM

Too many dependencies?

$root			1	2	3	4	5	6
org.example	artifact-test	1	-					
	core	2		-				
	project	3		X	-			
	artifact-manager	4		X	X	-		
	reporting-api	5		X			-	
	model	6		X	X		X	-

Detecting antipattern in DSM

Change propagator

$root			1	2	3	4	5	6	7	8	9	10
org.example	actions	1	-								X	
	events	2	X	-		X		X			X	
	project	defaultWorkspaceManager 3			-		X				X	
		defaultworkspacecontext 4				-					X	
		partitioner 5					-			X	X	
		projectorLoader 6						-			X	
		projectView 7						X	-		X	
		projectUpdater 8								-	X	
		project 9	X	X	X	X	X	X	X	X	-	
	services	10						X				-

Component 9 (Project) is the change propagator

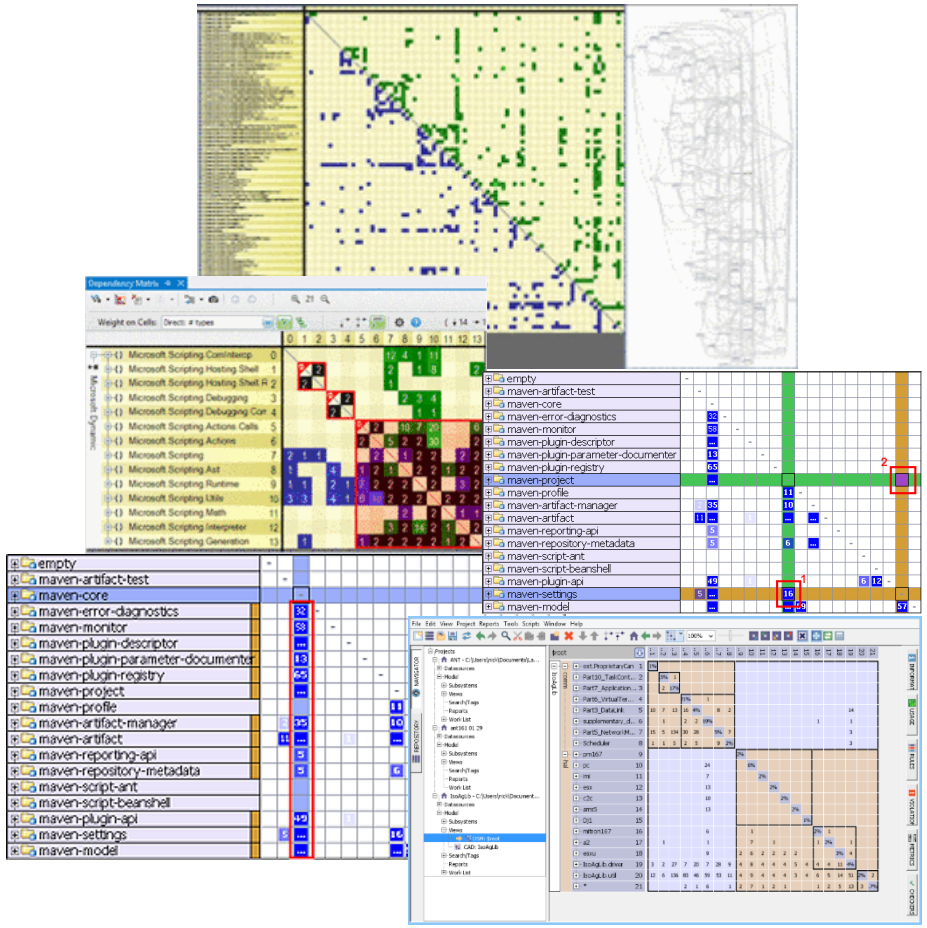
Imagine a change in component 10 (services)

["Using dependency models to manage complex software architecture." Sangal et al. (2005)]

Tool support for DSM

Plenty of tools and plugins for DSM analysis and visualization

- Eclipse
  - DSM-viewer
  - Eclipse-DSM-viewer
  - jDSM
- IntelliJ IDEA
  - DSM Analysis
- NDepend
  - Institute of Product Development - **TUM**
- Lattix dependency manager - lattix.com



[For more information - http://www.dsmweb.org/en/dsm-tools/commercial-tools.html]

Benefits of DSM

- The matrix representation scales better than box-and-line diagrams
- Graphs are more intuitive but when the numbers of nodes and edges grow, they becomes very complex
- Helps to better understand the information flows
- The partitioning algorithms provide an automatic mechanism for architectural discovery in a large code base
  - possibility to spot structural patterns at a glance
- Efficient cycle detection in DSM
- Integration of dependency rules
  - C1 can-use C2
  - C1 cannot-use C2

Challenges in DSM

- A DSM is only as good as the knowledge that goes into it.

- Unknown interdependencies can exist
  - one of the primary sources of uncertainty
  - can emerge unexpectedly

- DSM is less intuitive as compared to a graph

["Design rules: The power of modularity." Baldwin, C.Y. and Clark, K.B. (2000)]