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

ASSESSMENT AND IMPROVEMENT OF SOFTWARE SYSTEMS BY APPLYING DSM

Han van Roosmalen
D2Groep B.V.

Keywords: DSM, Software Architecture, Assessment, Improvement

1 INTRODUCTION

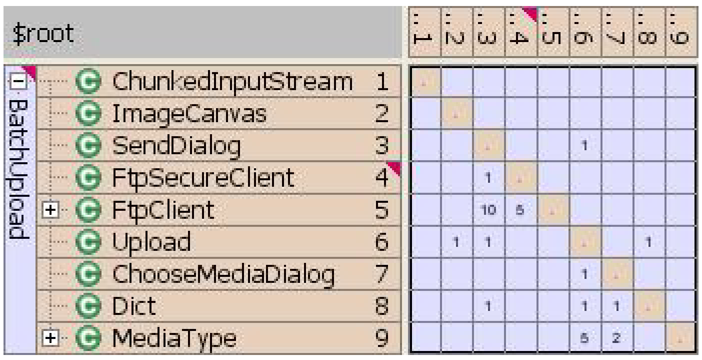
The application of DSM proves to be a very valuable tool for providing oversight and detailed insight in existing software systems. Based on the information shown in the DSM, managers and software and/or system architects obtain up-to-date information about the current state of their system. Since software DSMs provide technology-independent insight, more stakeholders can become informed easily.

Insight in the architectural structure is extremely valuable to any organisation because it is the real basis of the quality of their software system(s) and therewith the potential future of the system under development. Although most software systems are developed with valid intentions and might be documented properly, software architectures tend to erode overtime. With each new release, functionality is added incorrectly sometimes by new developers unaware of the initial architecture or due to extreme time-pressure. This will have a negative long-term effect on the overall quality characteristics of the system and it becomes less understandable.

For this and other reasons, applying DSM can play a special role when assessing and improving software systems.

2 ASSESSMENT

After more than one year of assessing different kind of systems, such as e-business, administrative and technical (embedded) systems, it is clear that applying DSM consistently provides benefits to the systems' stakeholders. In a short amount of time, the system complexity is revealed and valuable information is provided to the architect and the developers. After modelling the intended software architecture in the DSM, insight to the other stakeholders improves dramatically. The actual system complexity and adherence to the predefined software architecture is now demonstrated.



$root			1	2	3	4	5	6	7	8	9
BatchUpload	ChunkedInputStream	1									
	ImageCanvas	2									
	SendDialog	3						1			
	FtpSecureClient	4			1						
	FtpClient	5			10	5					
	Upload	6		1	1					1	
	ChooseMediaDialog	7						1			
	Dict	8			1			1	1		
	MediaType	9						5	2		

Figure 1. Example of an automatically generated DSM.

As an example figure 1 shows a DSM after the application code is imported. Even this relative simple system shows already two breakages of the hierarchy and modularity.

The DSM can now be utilised to perform assessments with a multitude of goals, such as:

- assess the quality of a third party component;
- assess the quality (and therewith the actual value) of a software component as produced by a supplier;

]

- determine ways to improve the system structure, e.g. amongst others by simulation of a possible refactoring approach,

or perform more concrete actions such as determine the required steps to perform a platform port or to decrease the memory footprint by identifying the actual used software parts.

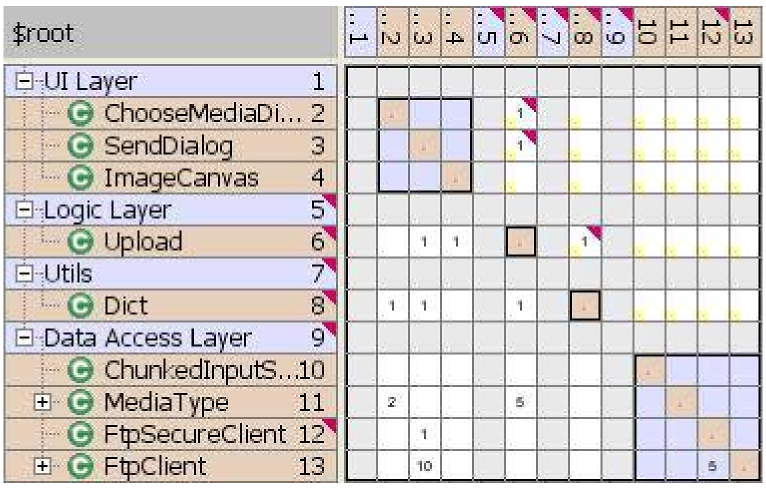
Each assessment consists of the following steps:

1. Determine which software modules (internal and well as external) make up the system and thus need to be investigated.
2. Load those modules into Lattix LDM and build the DSM.
3. Obtain a first draft overview of the structure.
4. Model the (intended) software architecture in the DSM.
5. With help of the systems architect and/or the architectural documents, determine correctness of the architectural structure.
6. Record deviations and ways to improve the structure of the systems and/or underlying components.

Possible assessment outcomes range from demonstrating the high quality of an excellent system decomposition to stopping any further development effort on a poor quality system. The minimum result of an assessment based on DSM is always an improved insight of the system at hand.

3 IMPROVEMENT

If the assessment outcome is sufficiently positive, the DSM is now used to guide the development team to a better architecture for the next release(s). First activities to be done will be in the area of improving the dependencies between the various software parts of the system. An improved architecture is demonstrated easily when dependencies are moved to the lower triangular area of the DSM.



$root		1	2	3	4	5	6	7	8	9	10	11	12	13
⊟ UI Layer	1													
ChooseMediaDi...	2						1							
SendDialog	3						1							
ImageCanvas	4													
⊟ Logic Layer	5													
Upload	6			1	1				1					
⊟ Utils	7													
Dict	8		1	1			1							
⊟ Data Access Layer	9													
ChunkedInputS...	10													
⊞ MediaType	11		2				5							
FtpSecureClient	12			1										
⊞ FtpClient	13			10									5	

Figure 2. Modeling of the intended architecture.

To continue with the previous example the system as presented in Figure 1, the model of a simple high level architectural layering is added. Parts that belong to such layer are moved to it. In this case as illustrated in Figure 2, as a result three inconsistencies appear in the upper triangle. Here software parts (Java classes in this example) in a lower layer call classes in the layer above and thereby break the intended layered architecture.

An improved hierarchy/structure will automatically provide a better understanding for all stakeholders involved, and thereby decrease maintenance cycles and increase robustness. How this can be achieved is first established in the DSM without making any changes to the code. If the structure does improve as intended, the proposed changes can then be handed over to the developer and implemented. The result of this action is then updated in the DSM and verified for correctness, making the DSM an even more valuable tool for the system architect who now can track changes at on a more abstract level. The architect can even allow and disallow predefined dependencies between software parts at different levels of abstraction. In the DSM in Figure 2, this is indicated by the small coloured triangles, where red indicates a violation of the design rule.

4 AUTOMATED CONTINUOUS ARCHITECTURE IMPROVEMENT

Of course the assessment and improvement processes can be combined as depicted in Figure 3. The workflow is composed of the following steps:

- Step 1: Loading compiled source code into the DSM modelling tool
- Step 2: Store architecture intend as design rules into a repository
- Step 3 and 4: During the following compilation and build cycles verify adherence to design rules
- Step 5: Generation of reports
- Step 6: Verify correctness and adapt source code where required

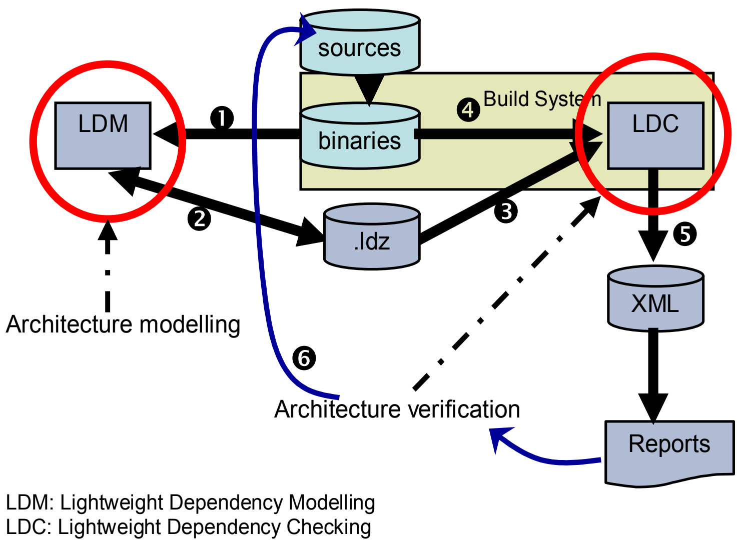

sources
binaries
Build System
LDM
LDC
.ldz
XML
Reports
Architecture modelling
Architecture verification
1
2
3
4
5
6
LDM: Lightweight Dependency Modelling
LDC: Lightweight Dependency Checking

Figure 3. Automatic Continuous Improvement of Architecture.

Steps 1, 2 and 6 are not automated and require interference of the architect and/or developers. The other steps are automatically executed during a recompilation of the source code.

5 CONCLUSION

Problems with the architecture of software system can result in:

- release date slippage,
- an increase of the number of people involved in the software development and testing,
- complaints of software developers who build systems on top of other systems.

When these kinds of problems occur, it becomes necessary to revaluate the system's structure and its internal and external dependencies. Assessment of the system by means of DSM proves to be a very effective option. To help the development team, the intended architecture can be modelled as a guidance tool for further development.

Contact: Han van Roosmalen
D2Groep BV.
Hakgriend 42
3371 KA Hardinxveld-Giessendam
The Netherlands
+31.184.621.232
han.van.roosmalen@d2groep.nl
http://www.d2groep.nl

Capitalize on Complexity
In cooperation with BMW Group

D2 GROEP ...dedicated to you
DLM D2EE D2IM D2SA

9th International DSM Conference

Assessment and Improvement of Software Systems by applying DSM

Ing. Han van Roosmalen

D2Groep
Dedicated to Software Architecture
The Netherlands

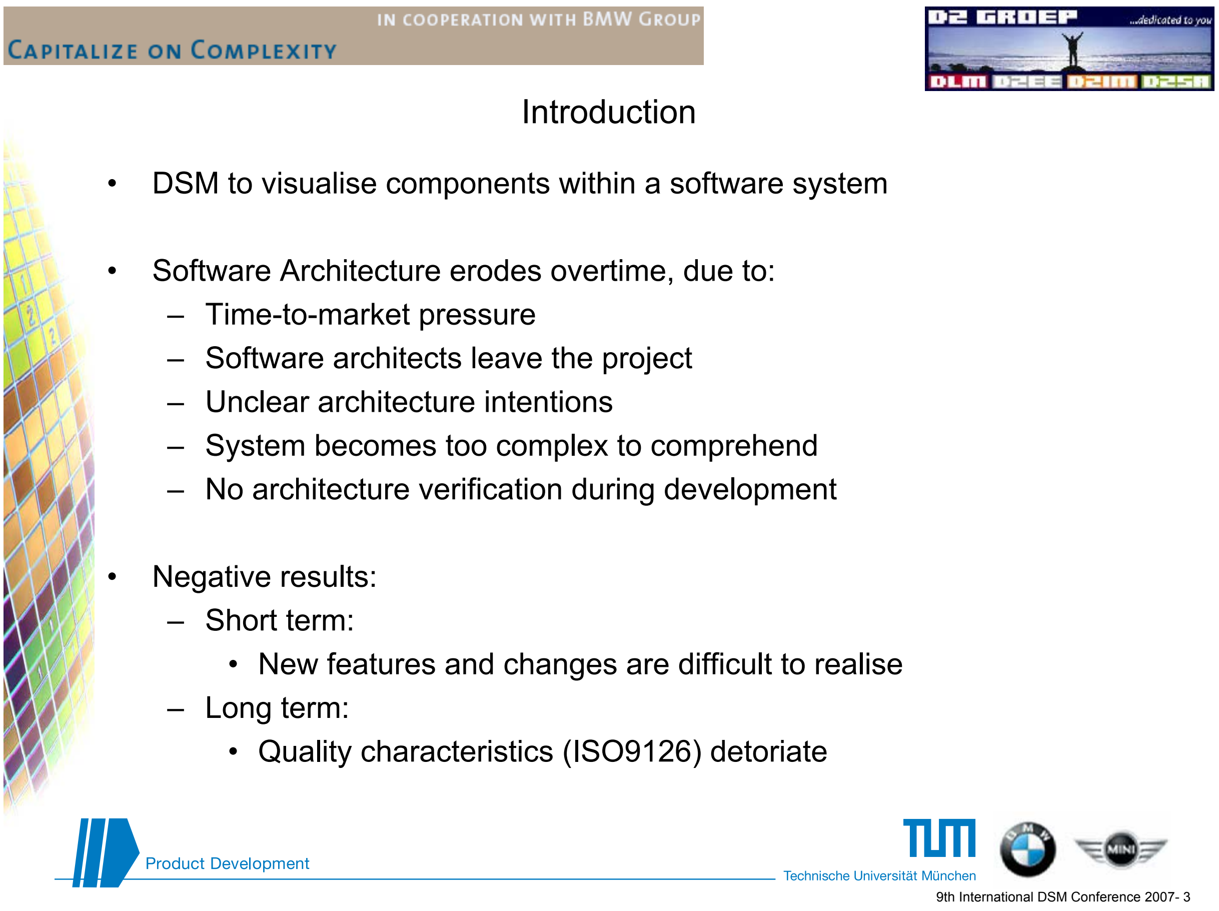
Product Development
Technische Universität München

Capitalize on Complexity
In cooperation with BMW Group

D2 GROEP ...dedicated to you
DLM D2EE D2IM D2SA

Index

Product Development
Technische Universität München

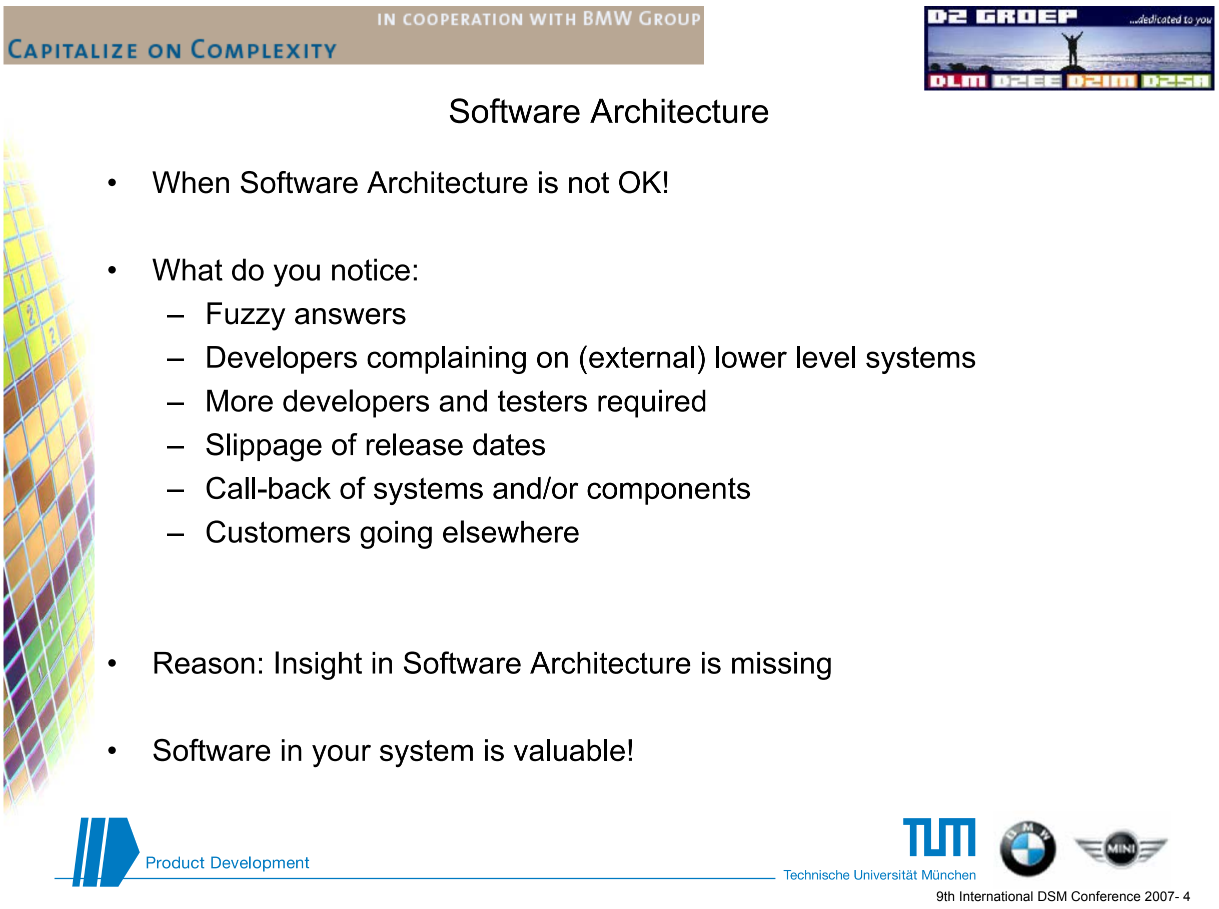


IN COOPERATION WITH BMW GROUP
CAPITALIZE ON COMPLEXITY

Introduction

- DSM to visualise components within a software system
- Software Architecture erodes overtime, due to:
  - Time-to-market pressure
  - Software architects leave the project
  - Unclear architecture intentions
  - System becomes too complex to comprehend
  - No architecture verification during development
- Negative results:
  - Short term:
    - New features and changes are difficult to realise
  - Long term:
    - Quality characteristics (ISO9126) detoriate

Product Development
Technische Universität München
9th International DSM Conference 2007- 3



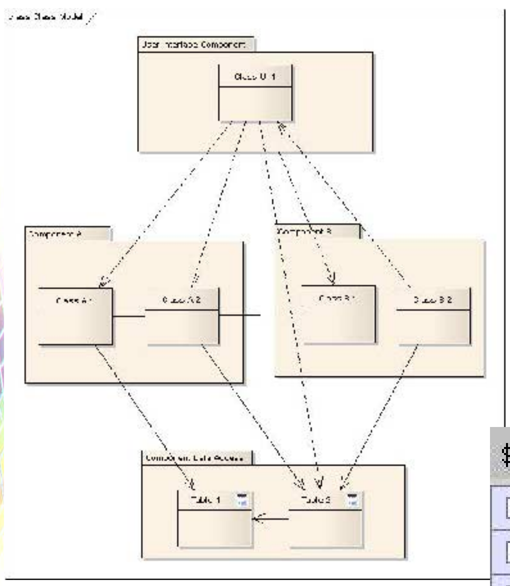
IN COOPERATION WITH BMW GROUP
CAPITALIZE ON COMPLEXITY

Software Architecture

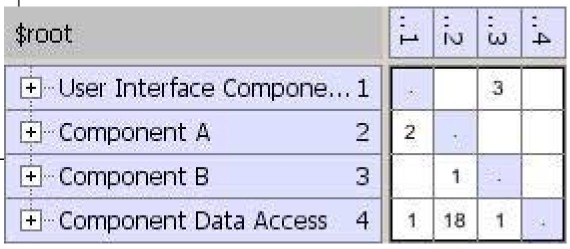
- When Software Architecture is not OK!
- What do you notice:
  - Fuzzy answers
  - Developers complaining on (external) lower level systems
  - More developers and testers required
  - Slippage of release dates
  - Call-back of systems and/or components
  - Customers going elsewhere
- Reason: Insight in Software Architecture is missing
- Software in your system is valuable!

Product Development
Technische Universität München
9th International DSM Conference 2007- 4

Insight in Software Architecture

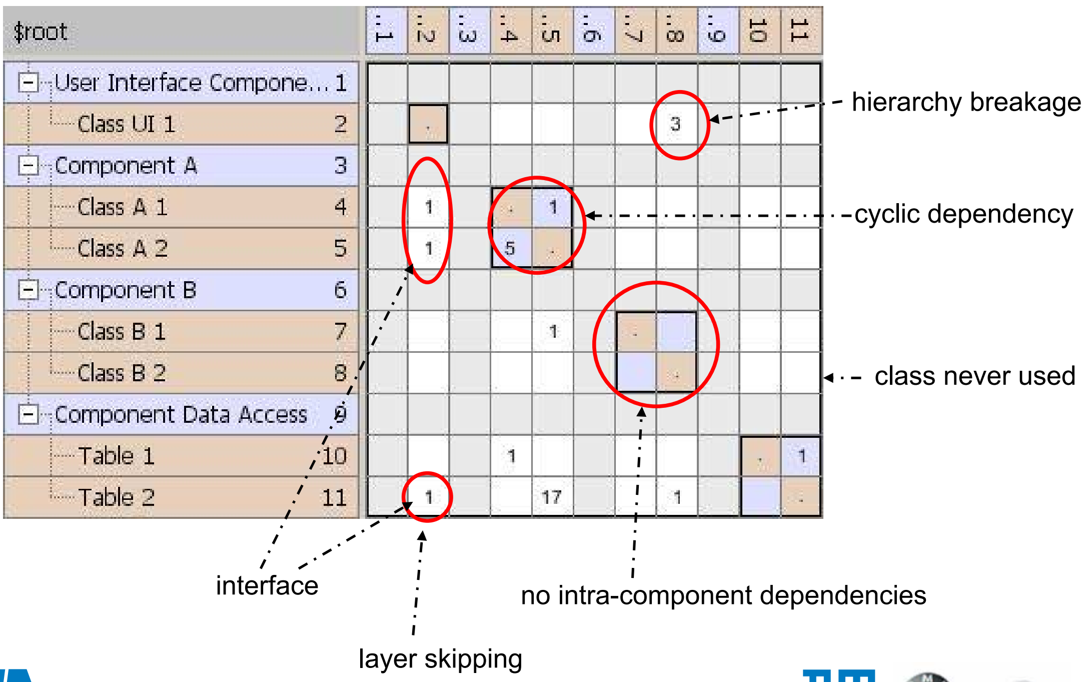


- Static analysis of high-level system structure
  - Decomposition/Structure
  - Abstraction
  - Layering
- For software architecture it is more logical to position the lower level components to the lower left triangle, since this makes hierarchy discovery easier

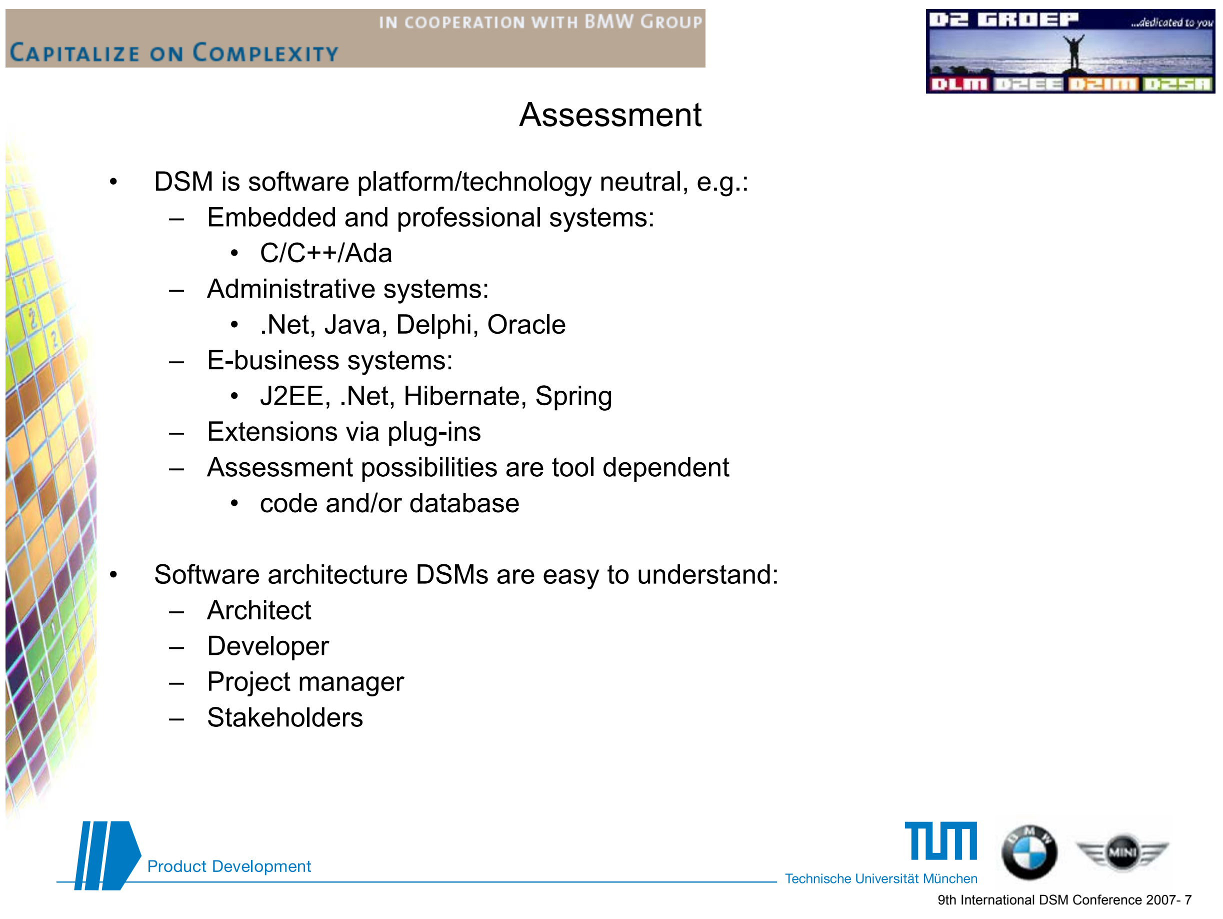


$root		1	2	3	4
User Interface Compone...	1	.		3	
Component A	2	2	.		
Component B	3		1	.	
Component Data Access	4	1	18	1	.

Insight in Software Architecture



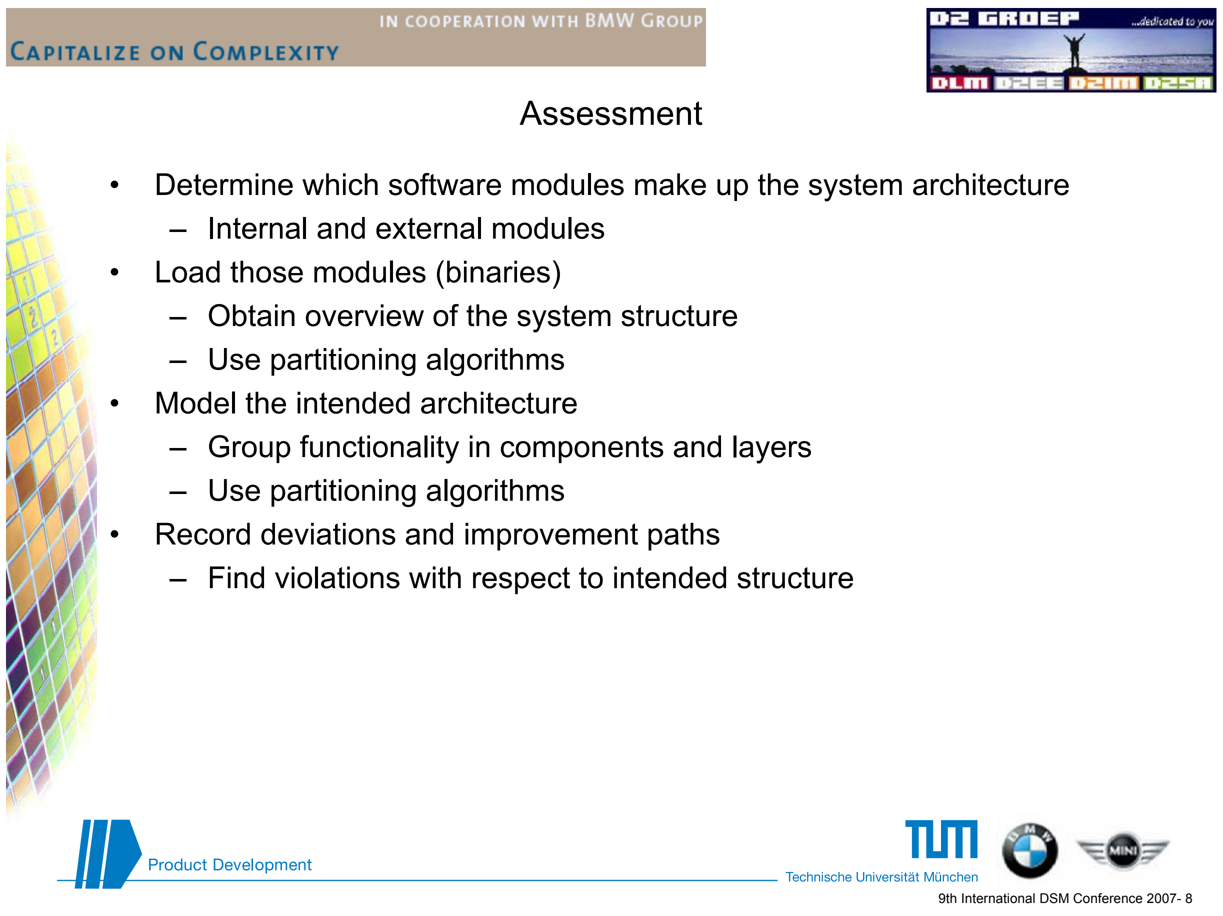
$root		1	2	3	4	5	6	7	8	9	10	11
User Interface Compone...	1											
Class UI 1	2		.						3			
Component A	3											
Class A 1	4		1		.	1						
Class A 2	5		1		5	.						
Component B	6											
Class B 1	7					1		.				
Class B 2	8								.			
Component Data Access	9											
Table 1	10				1						.	1
Table 2	11		1			17			1			.



Assessment

- DSM is software platform/technology neutral, e.g.:
  - Embedded and professional systems:
    - C/C++/Ada
  - Administrative systems:
    - .Net, Java, Delphi, Oracle
  - E-business systems:
    - J2EE, .Net, Hibernate, Spring
  - Extensions via plug-ins
  - Assessment possibilities are tool dependent
    - code and/or database

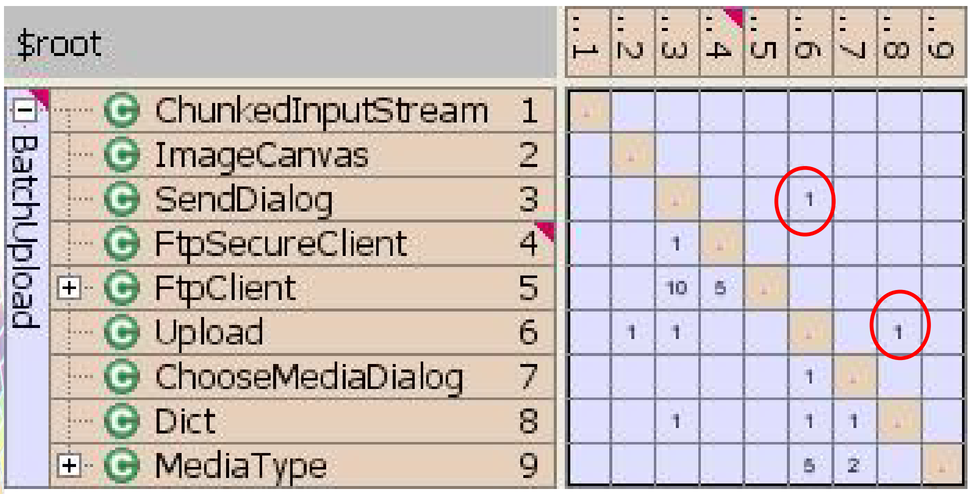
- Software architecture DSMs are easy to understand:
  - Architect
  - Developer
  - Project manager
  - Stakeholders



Assessment

- Determine which software modules make up the system architecture
  - Internal and external modules
- Load those modules (binaries)
  - Obtain overview of the system structure
  - Use partitioning algorithms
- Model the intended architecture
  - Group functionality in components and layers
  - Use partitioning algorithms
- Record deviations and improvement paths
  - Find violations with respect to intended structure

Assessment Example 1

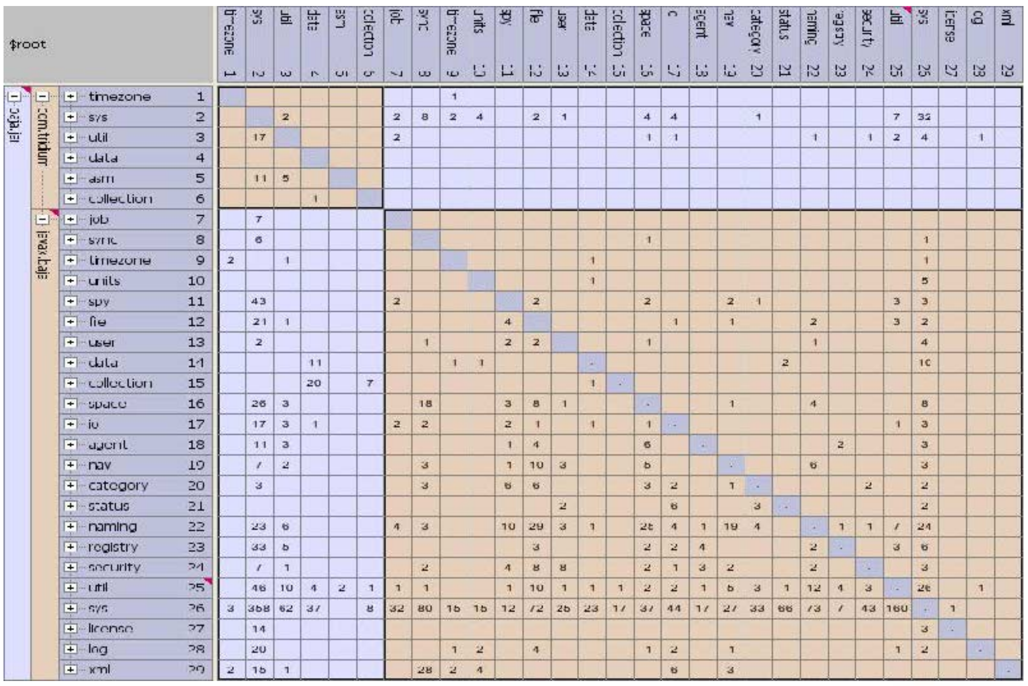


$root			1	2	3	4	5	6	7	8	9
BatchUpload	ChunkedInputStream	1									
	ImageCanvas	2									
	SendDialog	3						1			
	FtpSecureClient	4			1						
	FtpClient	5			10	5					
	Upload	6		1	1					1	
	ChooseMediaDialog	7						1			
	Dict	8			1			1	1		
	MediaType	9						5	2		

Partitioning based on strength provides first cut of realised architecture and shows 2 hierarchy violations

Code read > 30 minutes vs. DSM < 2 minutes

Assessment Example 2



$root				timezone 1	sys 2	util 3	data 4	asm 5	collection 6	job 7	sync 8	timezone 9	units 10	spy 11	file 12	user 13	data 14	collection 15	space 16	io 17	agent 18	nav 19	category 20	status 21	naming 22	registry 23	security 24	util 25	sys 26	license 27	log 28	xml 29
baja.jar	com.tridium	timezone	1									1																				
		sys	2			2				2	8	2	4		2	1			4	4			1					7	32			
		util	3		17					2									1	1					1		1	2	4		1	
		data	4																													
		asm	5		11	5																										
		collection	6				1																									
	javax.baja	job	7		7																											
		sync	8		6														1										1			
		timezone	9	2		1											1												1			
		units	10														1												5			
		spy	11		43					2					2				2			2	1					3	3			
		file	12		21	1								4						1		1			2			3	2			
		user	13		2						1			2	2				1						1				4			
		data	14				11					1	1											2					10			
		collection	15				20		7								1															
		space	16		26	3					18			3	8	1						1			4				8			
		io	17		17	3	1			2	2			2	1		1		1									1	3			
		agent	18		11	3								1	4				6							2			3			
		nav	19		7	2					3			1	10	3			6						6				3			
		category	20		3						3			6	6				3	2		1					2		2			
		status	21													2				6			3						2			
		naming	22		23	6				4	3			10	29	3	1		26	4	1	19	4			1	1	7	24			
		registry	23		33	6									3				2	2	4				2			3	6			
		security	24		7	1					2			4	8	8			2	1	3	2			2				3			
		util	25		46	10	4	2	1	1	1			1	10	1	1	1	2	2	1	6	3	1	12	4	3		26		1	
		sys	26	3	368	62	37		8	32	80	15	15	12	72	26	23	17	37	44	17	27	33	66	73	7	43	160		1		
		license	27		14																								3			
		log	28		20							1	2		4				1	2		1						1	2			
		xml	29	2	15	1					28	2	4							6		3										

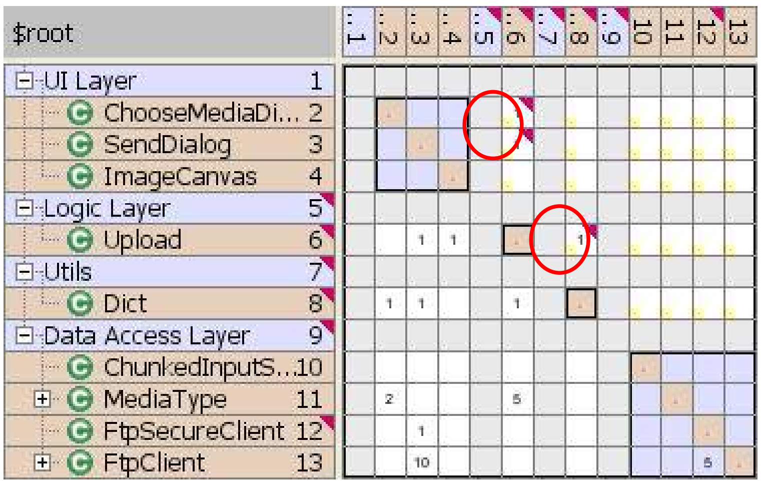
Assessment

- Results of an assessment:
  - Insight in the current state of the actual software
    - Not the colourful diagrams of the “how-it-should-be” architecture
  - Possible hot spots and areas of concern

Improvement

- Relocate software parts (classes/functions) to improve system structure concerning:
  - Violations in (strictly) layering
  - Violations in hierarchy
  - Violations in functionality to component mapping
- Simulate improvements without any code changes:
  - Change DSM structure first
  - Change code structure afterwards
- Verify improvements:
  - Run structure verification checker directly after system (re)build
  - Receive detailed reports on current violations

Improvement Example

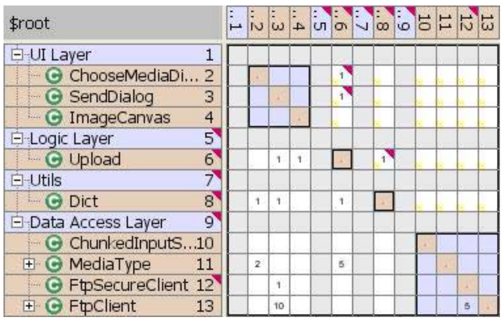


$root		:1	:2	:3	:4	:5	:6	:7	:8	:9	10	11	12	13
UI Layer	1													
ChooseMediaDi...	2						1							
SendDialog	3						1							
ImageCanvas	4													
Logic Layer	5													
Upload	6			1	1				1					
Utils	7													
Dict	8		1	1			1							
Data Access Layer	9													
ChunkedInputS...	10													
MediaType	11		2				5							
FtpSecureClient	12			1										
FtpClient	13			10									5	

More imperfections show up after layering and componentisation of intended architecture

Improvement

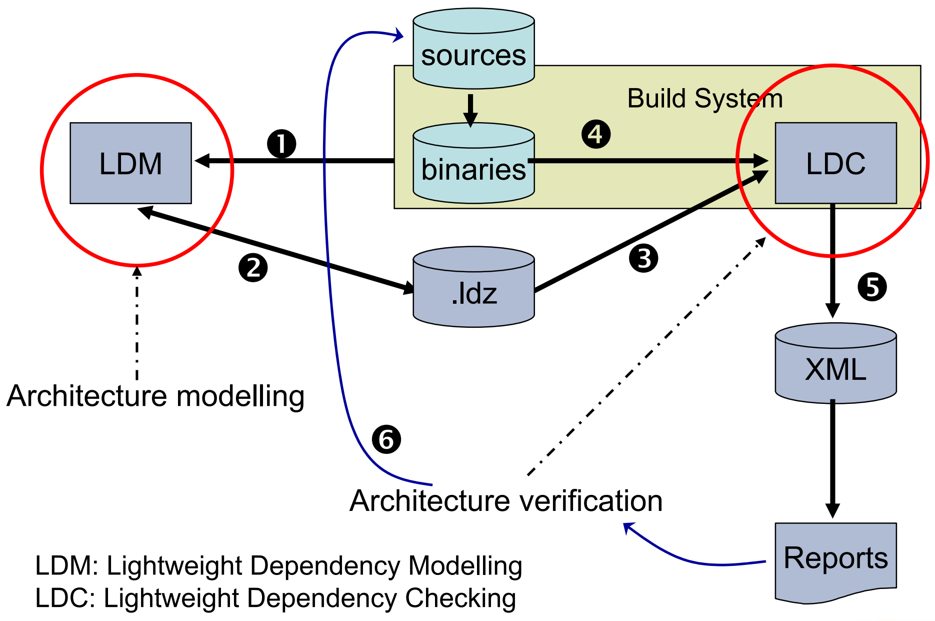
- Steps:
  - Add architecture intend to DSM model:
    - Architecture diagrams
    - Architect, architecture team
  - Make violations visible
  - Set-up design rules for automatic verification during software compilation and build



$root		:1	:2	:3	:4	:5	:6	:7	:8	:9	10	11	12	13
UI Layer	1													
ChooseMediaDi...	2						1							
SendDialog	3						1							
ImageCanvas	4													
Logic Layer	5													
Upload	6			1	1				1					
Utils	7													
Dict	8		1	1			1							
Data Access Layer	9													
ChunkedInputS...	10													
MediaType	11		2				5							
FtpSecureClient	12			1										
FtpClient	13			10									5	

Improvement

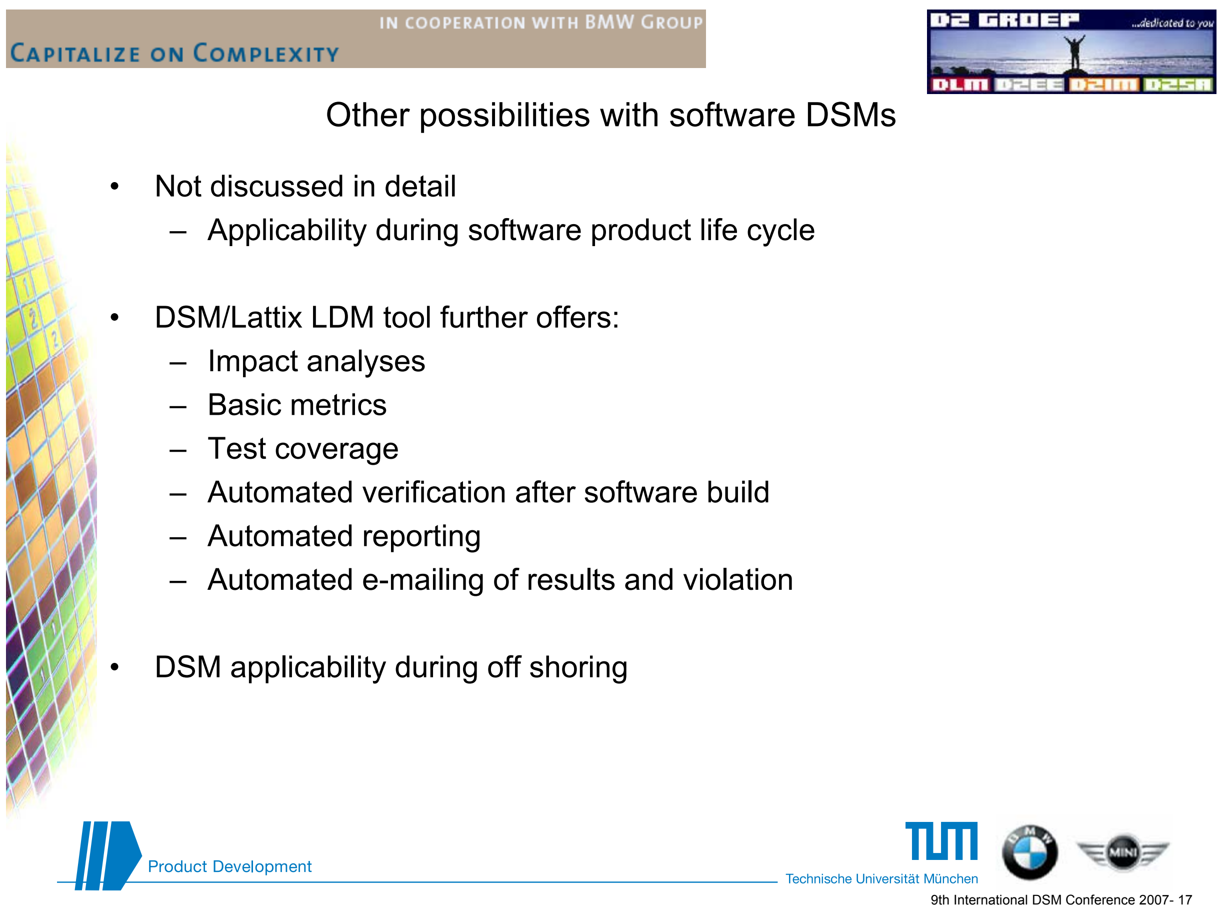
- Automated Continuous Architecture Improvement



LDM: Lightweight Dependency Modelling
LDC: Lightweight Dependency Checking

Results of Improvement

- Improved insight and structure for:
  - Architect
  - Developer
  - Tester
- Better architecture:
  - Easier/faster to add new features
    - Release dates can be set more realistic
  - Reduction of test time
  - Less/no erosion over software product life-cycle
- Some encountered side effects:
  - Find external libraries easily (license concerns)
  - Architect knows something is not perfect, but cannot pinpoint problem (build-in obfuscation)
  - Discussions that matter based on output (team communication)


IN COOPERATION WITH BMW GROUP

CAPITALIZE ON COMPLEXITY

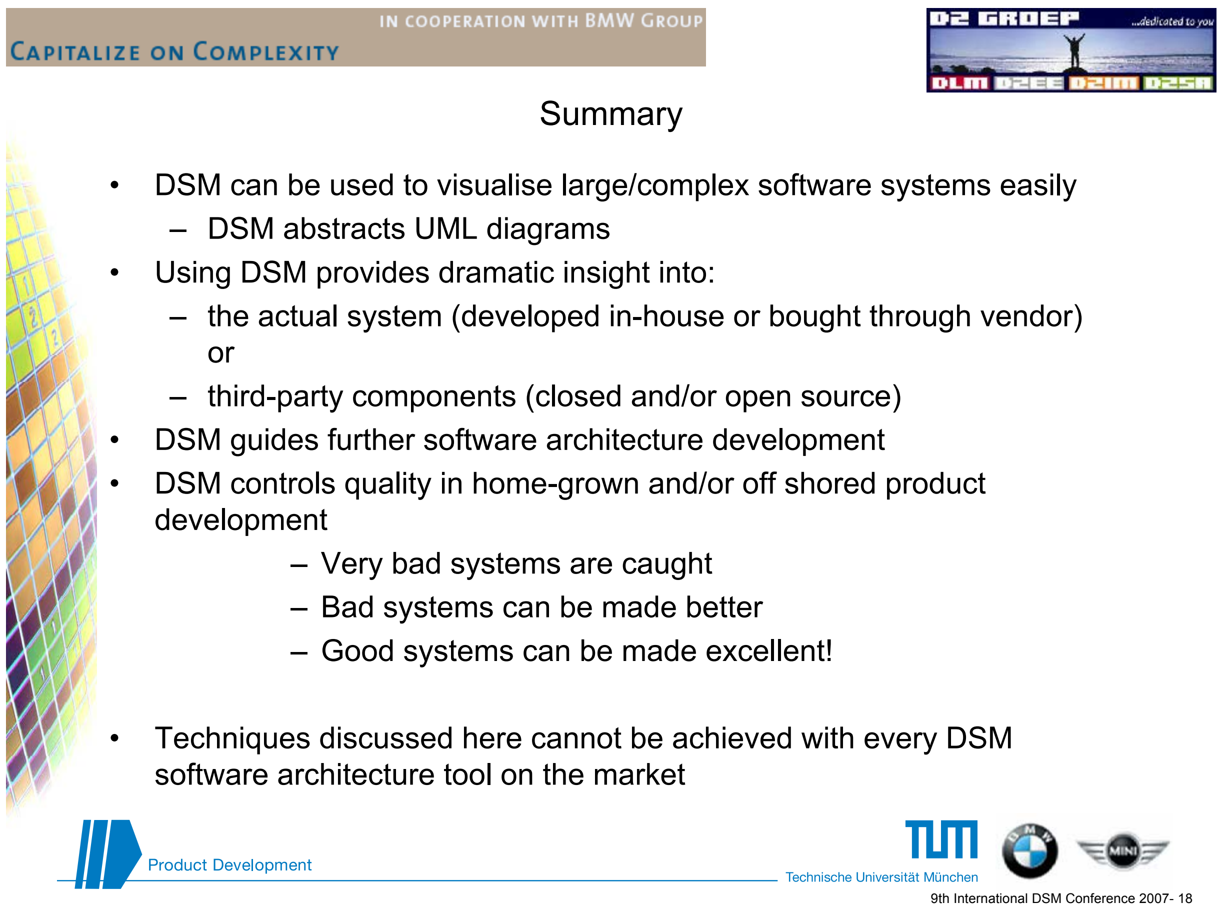
Other possibilities with software DSMs

- Not discussed in detail
  - Applicability during software product life cycle
- DSM/Lattix LDM tool further offers:
  - Impact analyses
  - Basic metrics
  - Test coverage
  - Automated verification after software build
  - Automated reporting
  - Automated e-mailing of results and violation
- DSM applicability during off shoring

Product Development

Technische Universität München

9th International DSM Conference 2007- 17


IN COOPERATION WITH BMW GROUP

CAPITALIZE ON COMPLEXITY

Summary

- DSM can be used to visualise large/complex software systems easily
  - DSM abstracts UML diagrams
- Using DSM provides dramatic insight into:
  - the actual system (developed in-house or bought through vendor) or
  - third-party components (closed and/or open source)
- DSM guides further software architecture development
- DSM controls quality in home-grown and/or off shored product development
  - Very bad systems are caught
  - Bad systems can be made better
  - Good systems can be made excellent!
- Techniques discussed here cannot be achieved with every DSM software architecture tool on the market

Product Development

Technische Universität München

9th International DSM Conference 2007- 18