An Analysis Of Modularity In Aspect Oriented Design

(paper submitted for AOSD'04, under review)

Cristina Videria Lopes and Sushil Krishna Bajracharya
(Authors)

ICS 229 Presentation, Fall 2004

Sushil K Bajracharya
sbajrach@ics.uci.edu

Donald Bren School of Information and Computer Sciences
Department of Informatics
University of California, Irvine

Background

- A fundamental theory of value based design
  - [BC00] C. Y. Baldwin and K. B. Clark. Design Rules vol I, The Power of Modularity. MIT Press, 2000.
- ***Thesis:*** Design is a value seeking process
  - "Modularization of a system can generate tremendous amounts of value in an industry, given that this strategy creates valuable options for module improvement." [Mac+]
- Basic ***components*** of the theory
  - Six modular Operators for expressing design evolution
  - Modeling designs using DSMs (Dependency Structure Matrices) and hierarchy diagrams
  - Net Options Value as a mathematical model for valuing designs

[Mac+] Alan MacCormack, John Rusnak, and Carliss Baldwin. Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code.

Background (2)

- Applications of Baldwin and Clark's theory
  - System Design (IBM 360 family, UNIX) [BC00]
  - KWIC [Sul+01]
  - Aspects [our paper]

[Sul+01] K. J. Sullivan, W. G. Griswold, Y. Cai, and B. Hallen. The structure and value of modularity in software design. ACM SIGSOFT 2001

Terminology

- *Design*: abstract description of functionality and structure of an artifact.
- *Medium* for expressing design: A designer expresses the basic structure and configuration of design elements with a *medium* (s)he chooses to work with.
- *Design parameters*: attributes of the artifact that govern the variation in design. Choosing new values for parameters give new design options.
- *Hierarchies*: formed by partial ordering of parameter dependencies (for e.g. 'uses' relationships).
  - [Par02] D. L. Parnas. On a "Buzzword": Hierarchical structure. In *Software pioneers: contributions to software engineering*, Springer-Verlag New York, Inc., 2002.
- *Abstraction*: hides the complexity of the element.
- *Information hiding*: design for change.
  - [Par72] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 15(12):1053-1058, 1972.

Terminology (2)

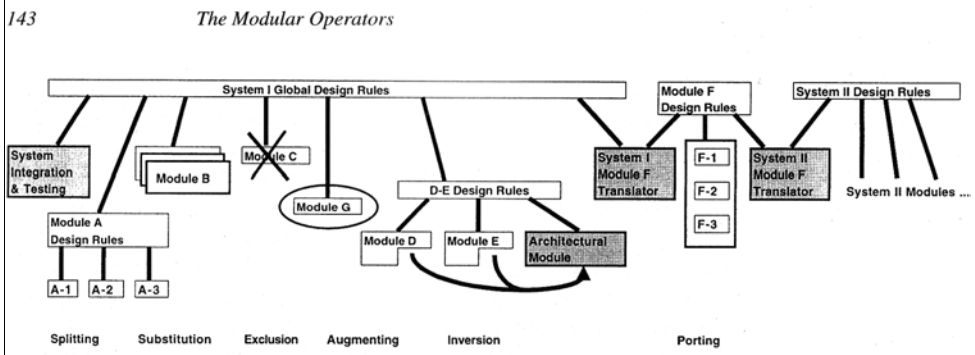
- *Modules*: strongly connected structural elements grouped together as a module
  - increases the range of manageable complexity
  - allows concurrent work
  - accommodates uncertainty
- *Design rules* and *Interface*:
  - decisions common to modules, that are unlikely to change are factored out as design rules
  - design rules constitute the interfaces to connect modules with each other
- *Architecture*: provides a framework that allows for both independence of structure and integration of function

Outline

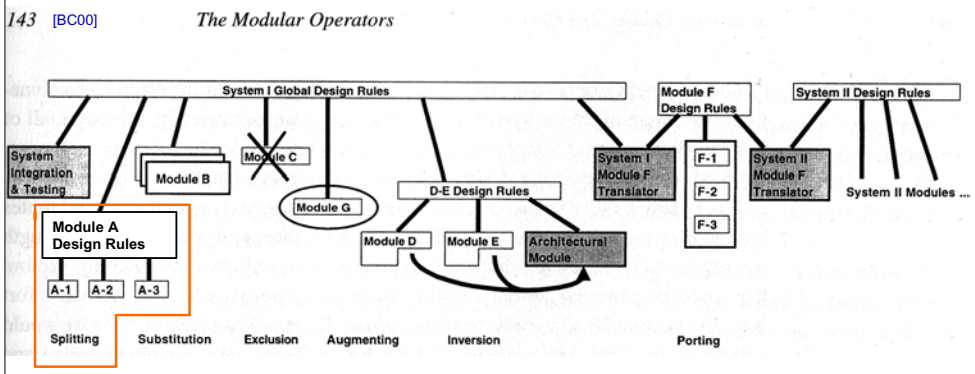
- Outline of the paper
  - Tracing design evolution of a web services application
    - Introducing aspects (aspect oriented modularization)
  - Heuristics and assumptions for analysis
  - Evaluating design changes
  - Conclusions

Modular Operators

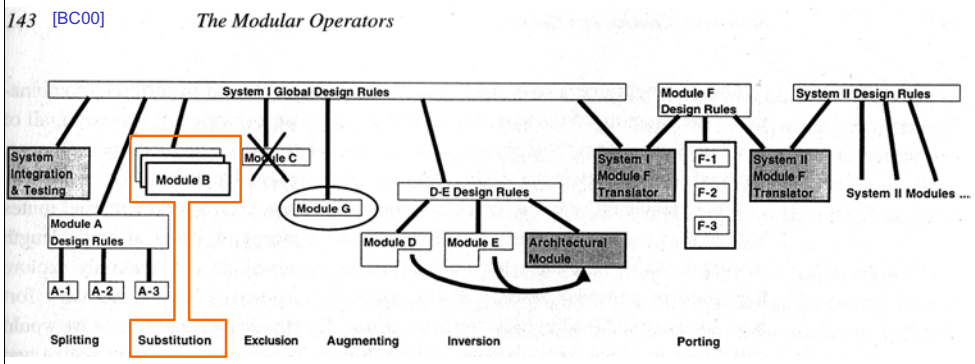
- Splitting, Substitution, Augmentation, Exclusion, Inversion, and Porting [BC00]



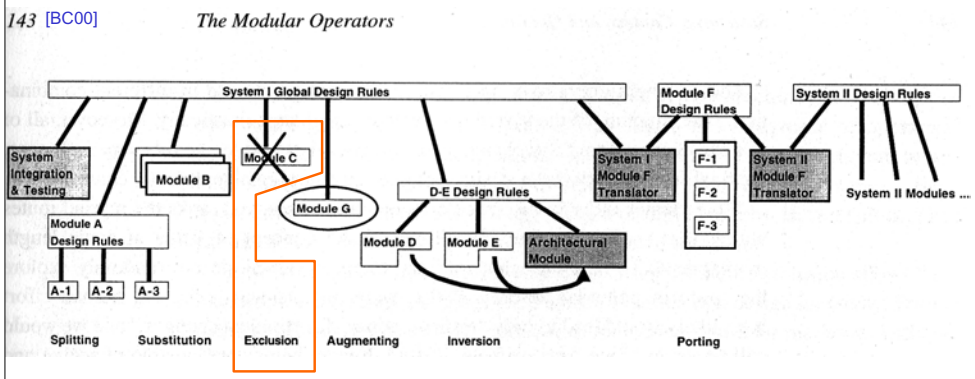
Splitting



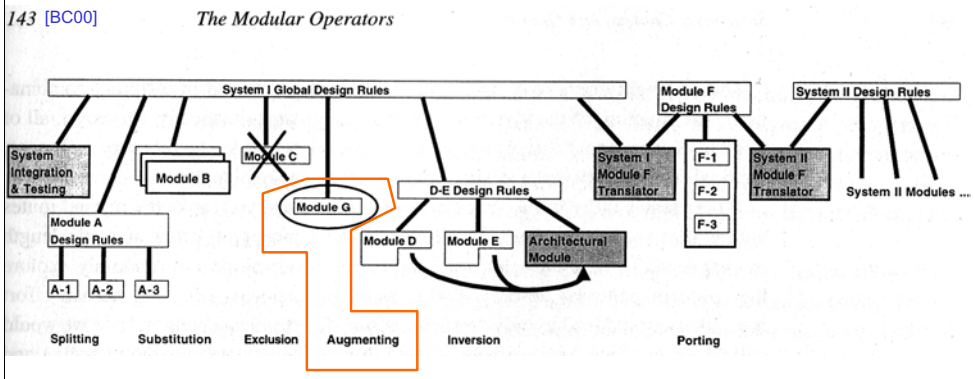
Substitution



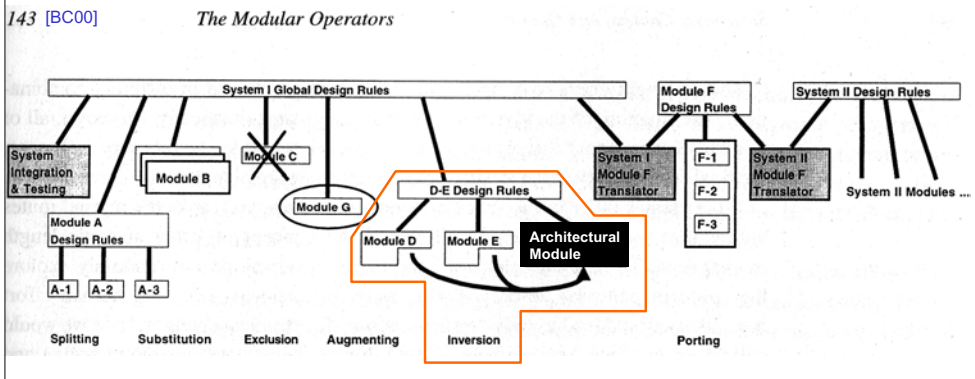
Exclusion



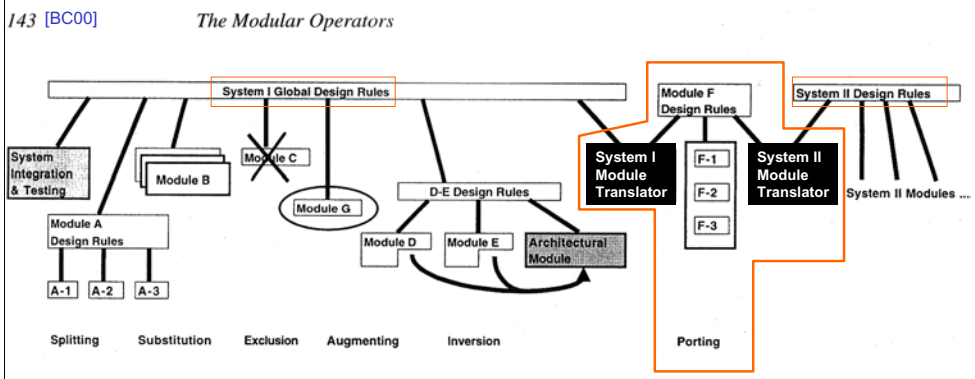
Augmenting



Inversion

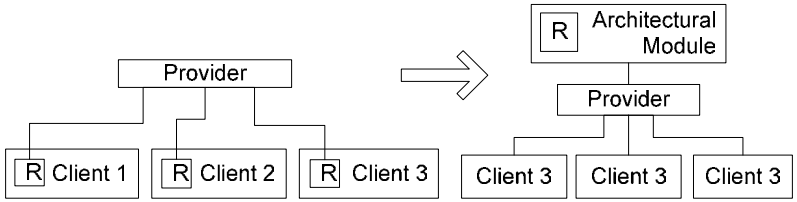


Porting



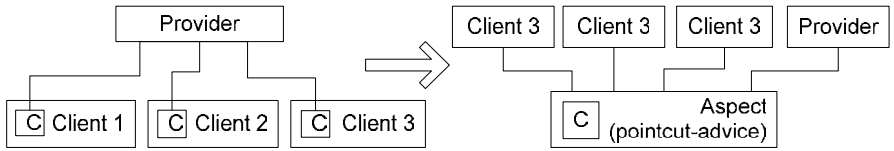
Aspect Oriented Modularization

- Aspect Oriented Modularization as a variant of Inversion
- The effect of Inversion
  - Factors out redundant parameters
  - Adds architectural modules
  - Changes the hierarchy



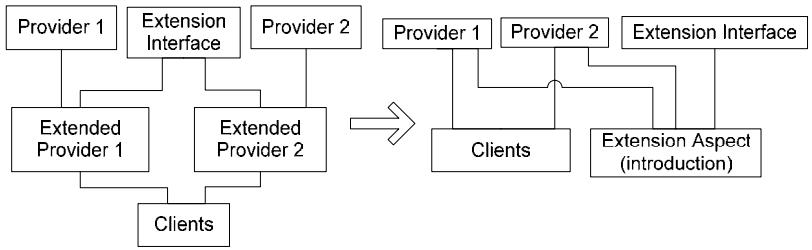
Aspect Oriented Modularization (2)

- Effect of aspects with pointcut-advice mechanism
  - Removes scattered code
  - Adds ‘Aspects’
  - Changes the hierarchy



Aspect Oriented Modularization (3)

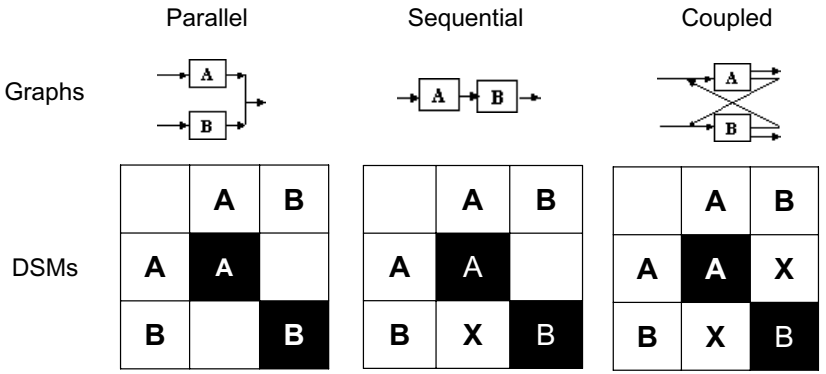
- Effect of aspects with ‘introductions’
  - Modifies the hierarchy
  - Factors out common parameters
  - Is this inversion ?



Aspect Oriented Modularization (4)

- What is similar to Inversion?
  - Captures common parameters and moves them to a single module
  - Changes the levels of the parameters in the hierarchy
- What is different from Inversion?
  - Introduces aspects that depend on existing module
  - Whereas, inversion introduces modules on which existing modules are dependent

DSMs – Dependency Matrix of Design Parameters

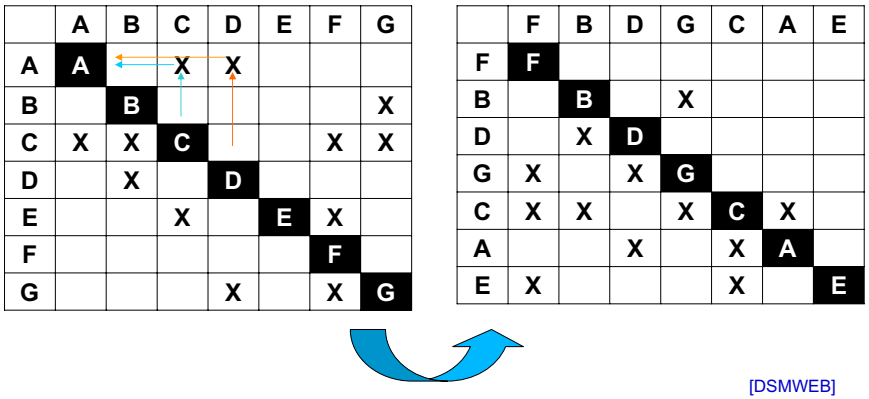


Three configurations of design parameters that characterize a system

[DSMWEB] www.dsmweb.org

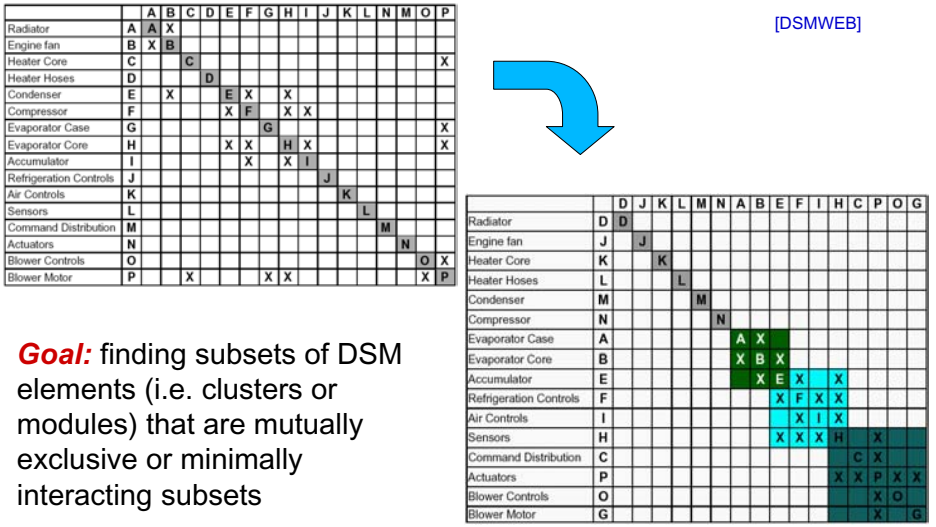
Partitioning a DSM

-Transforming the DSM into a nearly lower triangular form.
-Goal: define a task order, reduce feedback information flows



[DSMWEB]

Clustering in DSM



[DSMWEB]

Goal: finding subsets of DSM elements (i.e. clusters or modules) that are mutually exclusive or minimally interacting subsets

Categories of Design Parameters

- Simple clustering of design parameters into following categories
  - External Parameters
    - Libraries, Frameworks, External Services.
      - External Parameter is a particular category of Environment Parameter introduced in [Sul+01]
  - Design Rules
    - Parameters that are less likely to be changed.
    - Serve as interface between modules
  - Application Modules
    - Perform application specific tasks
  - Subsidiary Modules
    - Contribute to subsidiary functionalities (non-functional requirements)
  - Application Controller
    - modules that use the design rules as interface to access the functionalities provided by application modules

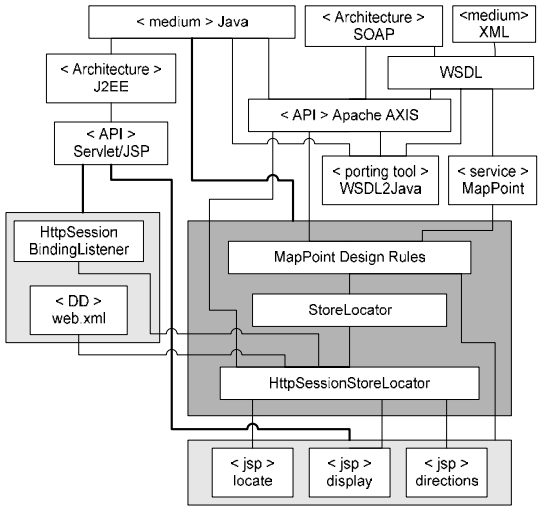
Example Application

- Tracing design evolution of an example application *WineryLocator* starting from an existing application *StoreLocator*.
- *StoreLocator* locates coffee stores in USA
- *WineryLocator* locates wineries in California
- Design Changes:
  - *StoreLocator*
  - *WineryLocator,* First version (Splitting and Substitution on *StoreLocator*)
  - *WineryLocator* with logging feature (Augmentation)
  - *WineryLocator*, Hiding design rules from external parameters (Refining Inversion)
  - *WineryLocator* with Aspect-oriented modularization

Requirements

- ***StoreLocator***
  - Functional
    - Finding accurate locations (starting point)
    - Getting List of coffee store
    - Generating navigable maps and driving directions
  - Non-Functional
    - Providing authentication mechanism for accessing MapPoint webservices
- ***WineryLocator***
  - Functional
    - Finding accurate locations (starting point)
    - Getting list of wineries according to users preference
    - Generating wineries tour of all wineries
    - Generating directions and maps for the tour
  - Non-Functional
    - Authentication
    - Logging access to WebServices

StoreLocator – Hierarchy diagram



StoreLocator - DSM

			1	2	3	4	5	6	7	8	9	10	11
< EP >	< service > MapPoint	1	*										
	< API > Apache AXIS	2		*									
	< API > Servlet	3			*								
	HttpSessionBindingListener	4			X	*							
<DR>	MapPoint Design Rules	5	X	X			*						
<AM>	StoreLocator	6					X	*					
	HttpSessionStoreLocator	7		X		X		X	*				X
< AC >	< jsp > locate	8			X		X		X	*	X		
	< jsp > display	9			X		X		X	X	*	X	
	< jsp > directions	10			X		X		X	X		*	
	< DD > web.xml	11	X	X	X								*

StoreLocator - DSM

			1	2	3	4	5	6	7	8	9	10	11
< EP >	< service > MapPoint	1	*										
	< API > Apache AXIS	2		*									
	< API > Servlet	3			*								
	HttpSessionBindingListener	4			X	*							
<DR>	MapPoint Design Rules	5	X	X			*						
<AM>	StoreLocator ⇐	6					X	*					
	HttpSessionStoreLocator ⇐	7		X		X		X	*				X
< AC >	< jsp > locate	8			X		X		X	*	X		
	< jsp > display	9			X		X		X	X	*	X	
	< jsp > directions	10			X		X		X	X		*	
	< DD > web.xml	11	X	X	X								*

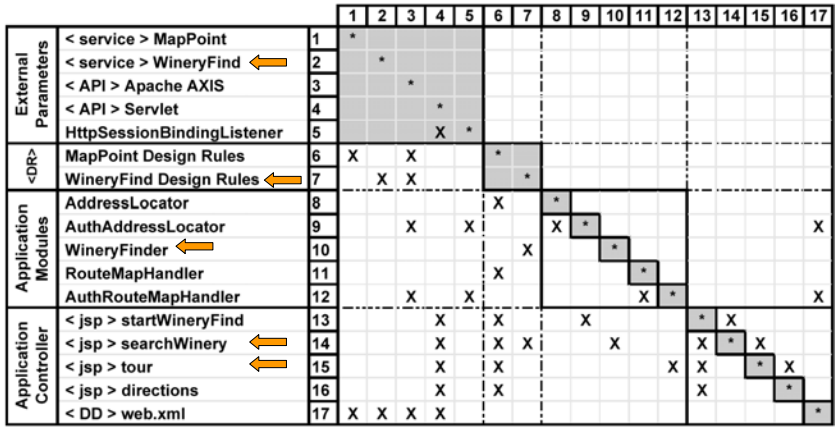
StoreLocator - DSM

			1	2	3	4	5	6	7	8	9	10	11
< EP >	< service > MapPoint	1	*										
	< API > Apache AXIS	2		*									
	< API > Servlet	3			*								
	HttpSessionBindingListener	4			X	*							
<DR>	MapPoint Design Rules	5	X	X			*						
<AM>	StoreLocator ⇐	6					X	*					
	HttpSessionStoreLocator ⇐	7		X		X		X	*				X
< AC >	< jsp > locate ⇦	8			X		X		X	*	X		
	< jsp > display ⇦	9			X		X		X	X	*	X	
	< jsp > directions ⇦	10			X		X		X	X		*	
	< DD > web.xml	11	X	X	X								*

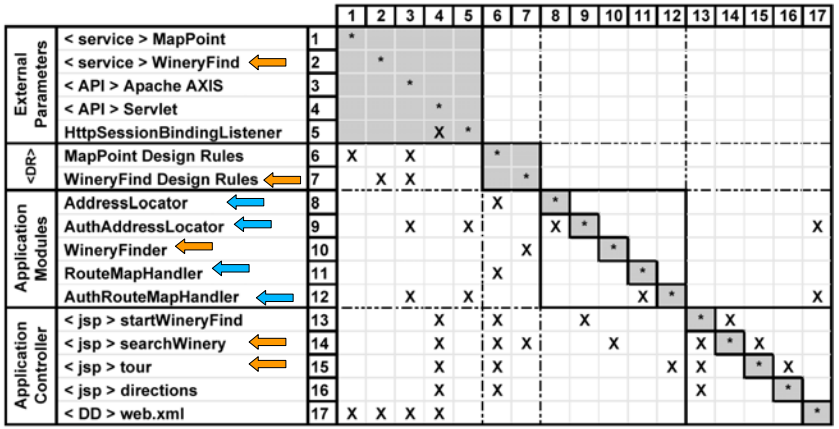
WineryLocator, first version

			1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
External Parameters	< service > MapPoint	1	*																
	< service > WineryFind	2		*															
	< API > Apache AXIS	3			*														
	< API > Servlet	4				*													
	HttpSessionBindingListener	5				X	*												
<DR>	MapPoint Design Rules	6	X		X			*											
	WineryFind Design Rules	7		X	X				*										
Application Modules	AddressLocator	8						X		*									
	AuthAddressLocator	9			X		X			X	*								X
	WineryFinder	10							X			*							
	RouteMapHandler	11						X					*						
	AuthRouteMapHandler	12			X		X						X	*					X
Application Controller	< jsp > startWineryFind	13				X		X			X				*	X			
	< jsp > searchWinery	14				X		X	X			X			X	*	X		
	< jsp > tour	15				X		X						X	X		*	X	
	< jsp > directions	16				X		X							X			*	
	< DD > web.xml	17	X	X	X	X													*

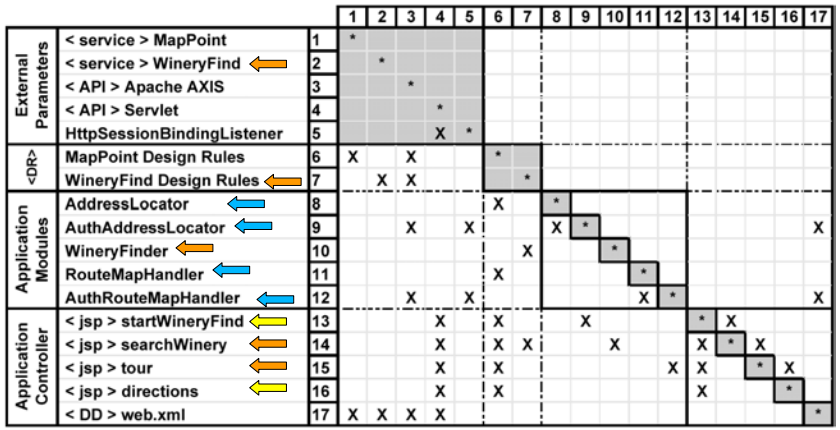
WineryLocator, first version



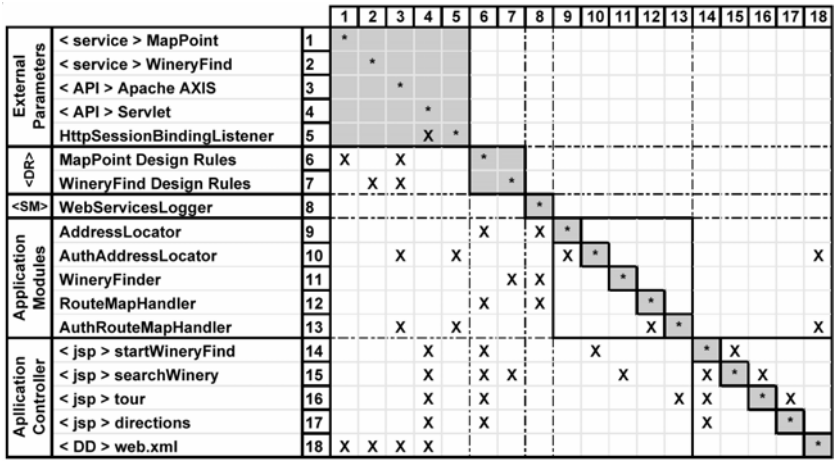
WineryLocator, first version



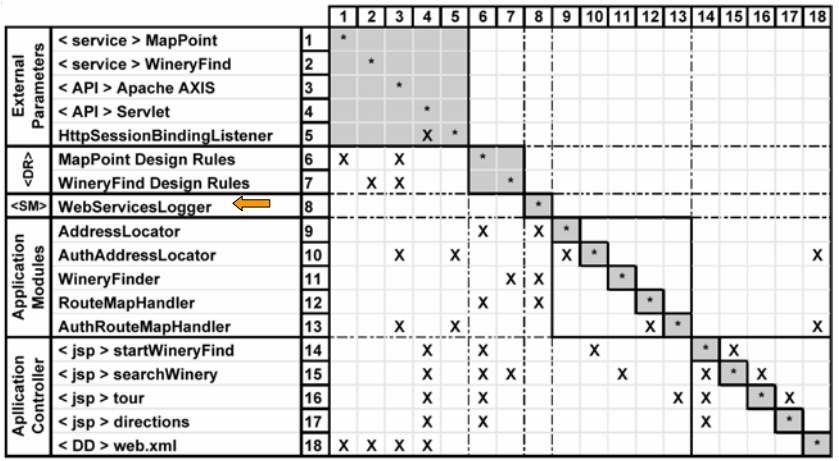
WineryLocator, first version



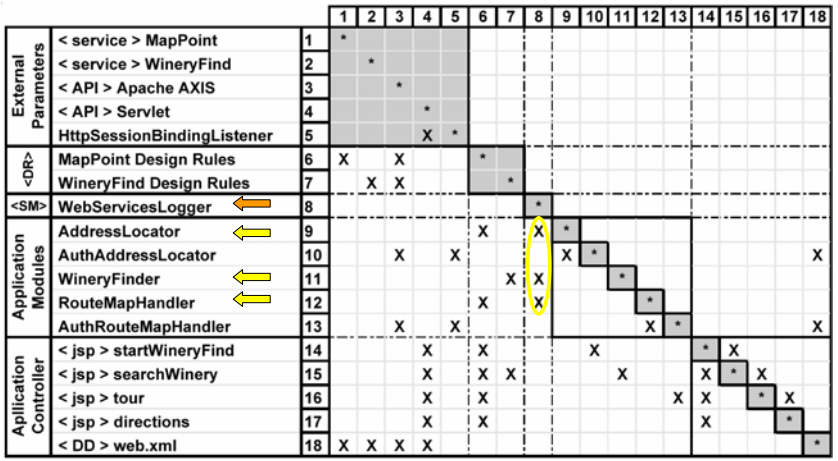
WineryLocator with Logging



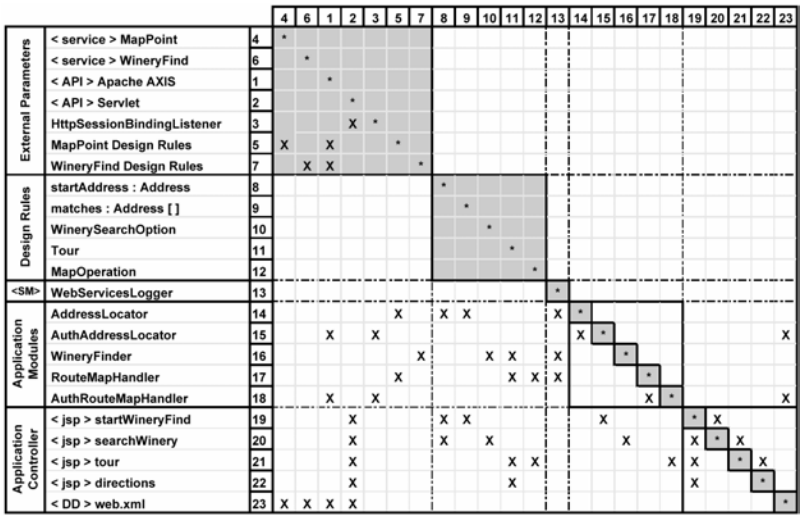
WineryLocator with Logging



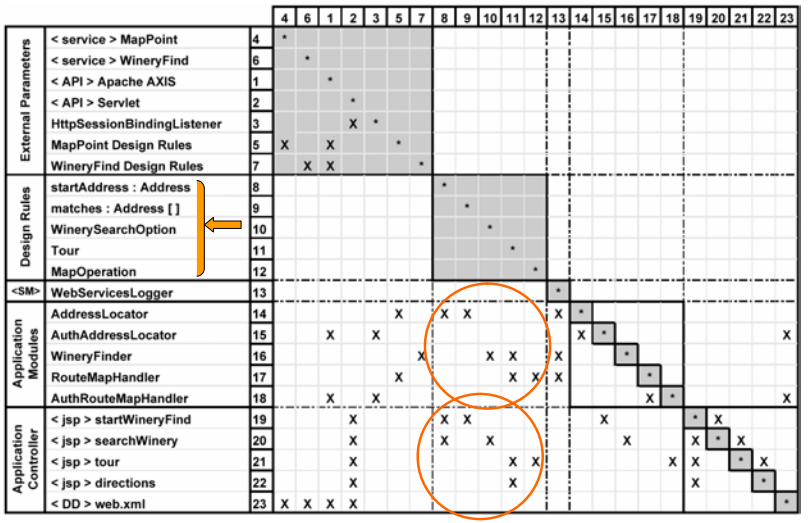
WineryLocator with Logging



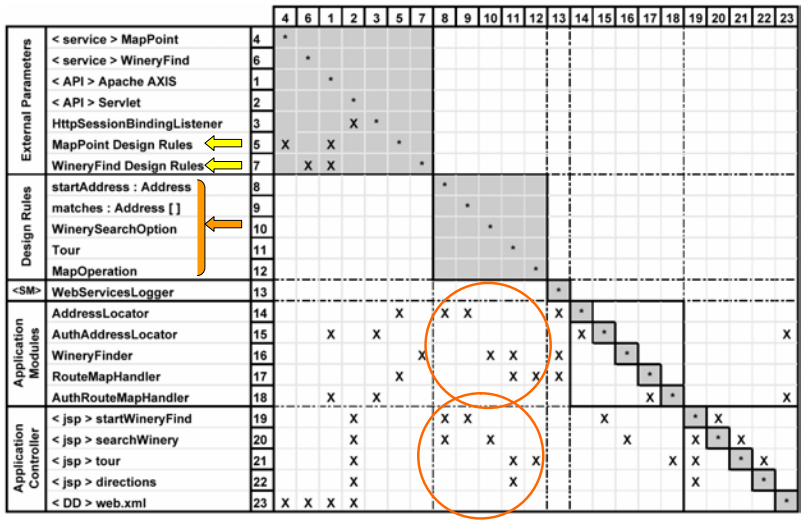
Winery Locator with new Design Rules



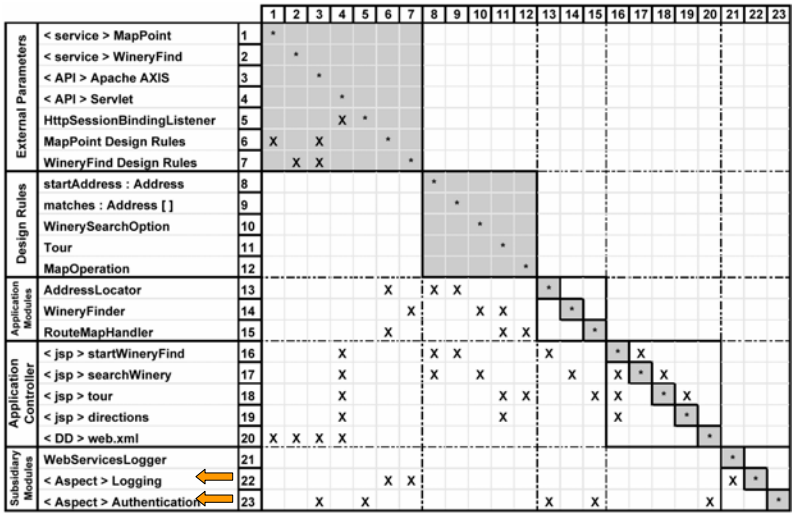
Winery Locator with new Design Rules



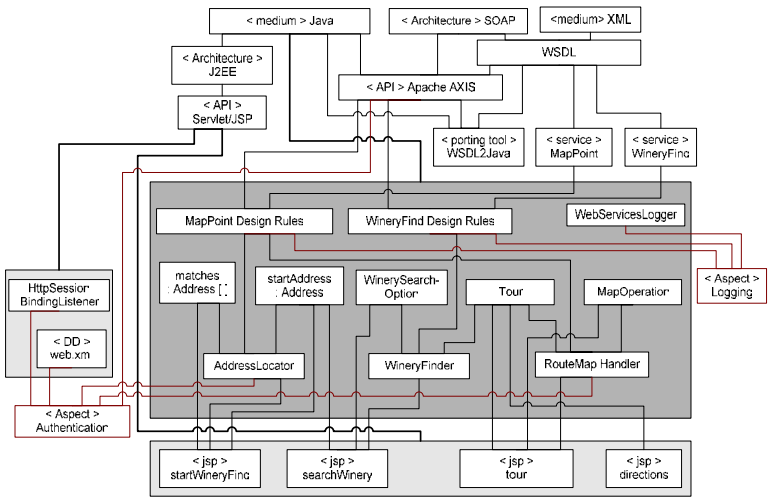
Winery Locator with new Design Rules



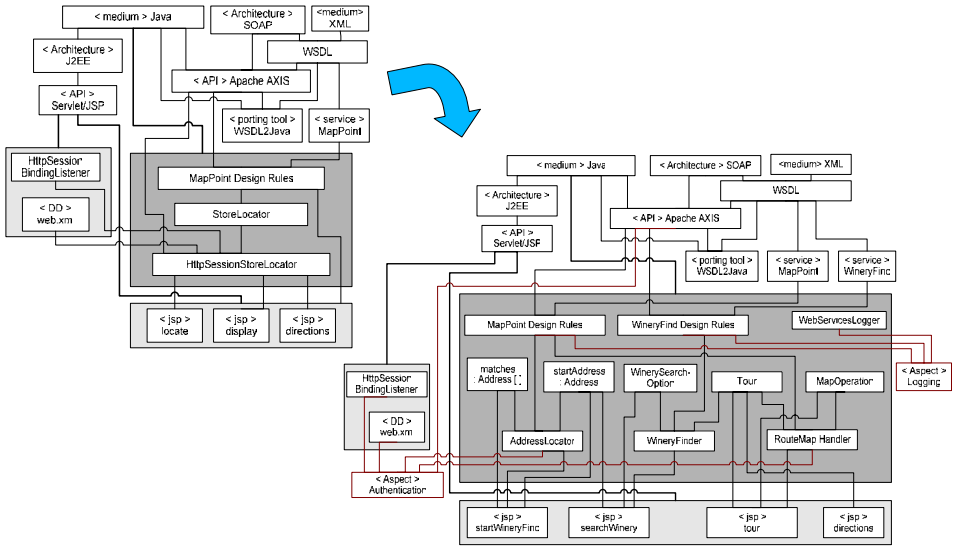
WineryLocator with Aspects



WineryLocator with Aspects, Hierarchy Diagram



StoreLocator to WineryLocator



NOV (Net Options Value)

$$V = S_0 + NOV_1 + NOV_2 + ... + NOV_n \quad (1)$$

$$NOV_i = max_{ki}\{\sigma_i n_i^{1/2} Q(k_i) - C_i(n_i)k_i - Z_i\} \quad (2)$$

$$Z_i = \sum_{j-sees-i} cn_j \quad (3)$$

General expression for NOV of a modular design

NOV (2)

Total value of the system

$$V = S_0 + NOV_1 + NOV_2 + ... + NOV_n \quad (1)$$

NOVs of the individual module

Value of the unmodularized system
(usually normalized to 0)

NOV (3)

NOV of ith module

Benefit

Investment

$$NOV_i = max_{ki}\{\sigma_i n_i^{1/2} Q(k_i) - C_i(n_i)k_i - Z_i\} \quad (2)$$

Maximum value out of k-experiments for the ith module

NOV (3)

NOV of ith module

Benefit

Investment

$$NOV_i = max_{ki}\{\sigma_i n_i^{1/2} Q(k_i) - C_i(n_i)k_i - Z_i\} \quad (2)$$

Visibility cost to replace ith module

Maximum value out of k-experiments for the ith module

Cost incurred in k-experiments: (function of the complexity 'n_i') * k_i

NOV (3)

NOV of i[th] module

Benefit

Investment

$$NOV_i = max_{ki}\{\sigma_i n_i^{1/2} Q(k_i) - C_i(n_i)k_i - Z_i\} \quad (2)$$

Expected value from k-experiments **(Random variable with normal distribution)**

Visibility cost to replace i[th] module

Maximum value out of k-experiments for the i[th] module

Cost incurred in k-experiments: (function of the complexity 'n_i') * k_i

NOV (4)

$$\sigma_i n_i^{1/2} Q(k_i)$$

$$NOV_i = max_{ki}\{\sigma_i n_i^{1/2} Q(k_i) - C_i(n_i)k_i - Z_i\} \quad (2)$$

EV

NOV (4)

Standard Deviation for *EV*

expected value of the best *k* independent trails **from a standard normal distribution** for all positive values in the distribution

$$\sigma_i n_i^{1/2} Q(k_i)$$

$$NOV_i = max_{ki}\{\sigma_i n_i^{1/2} Q(k_i) - C_i(n_i)k_i - Z_i\} \quad (2)$$

EV

NOV (4)

Technical potential of i[th] module

Complexity of i[th] module

Standard Deviation for *EV*

expected value of the best *k* independent trails **from a standard normal distribution** for all positive values in the distribution

$$\sigma_i n_i^{1/2} Q(k_i)$$

$$NOV_i = max_{ki}\{\sigma_i n_i^{1/2} Q(k_i) - C_i(n_i)k_i - Z_i\} \quad (2)$$

EV

NOV (5)

$$Z_i = \sum_{j-sees-i} cn_j$$

Visibility cost to replace i^{th} module

All j-modules that see i^{th} module

Redesign cost (assumed to be same for all modules)

Complexity of j^{th} module

$$NOV_i = max_{ki}\{\sigma_i n_i^{1/2} Q(k_i) - C_i(n_i)k_i - Z_i\} \quad (2)$$

Assumptions for NOV analysis

- *External parameters* are excluded as modules for NOV analysis because they are not subjected to further experimentation.
- We treat all parameters under *design rules* as a single module.
- All other design parameters are treated as individual modules.
- Redesign cost 'c_i' of a single module = 1
  - that leads to maximum value of 's' to be 2.5.
    - ($s N^{1/2} Q(1) - cN = 0$ and $Q(1) = 0.4$)
      - Breakeven assumption [BC00]

Assumptions (2)

- Technical Potential of a module = s_i
  - $s_i = f(e_i, p_i) = (e_i + 1)\ X\ p_i$
- e_i = number of *external parameters* that the i^{th} module depends on
- p_i = i^{th} module's relevance to the *end users*
  - p_i = 2, for Application Controller modules
  - p_i = 1, for functional modules, subsidiary modules, design rules
  - p_i = 0, for web.xml
- Value of '*s*' for all modules scaled relatively, using s_{max} = 2.5
- These assumptions are elaborations of observations made for *s* in [Sul+01]

Assumptions (3)

- Module Complexity = n_i
- N = the complexity of the whole design

  = the total tasks performed by all the modules in the system.
- Complexity of a module

  = (total number of tasks it performs) / N

Observations

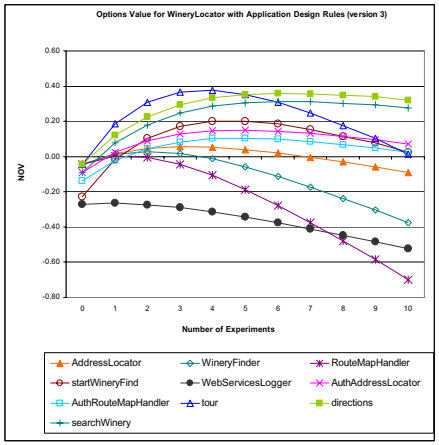
- NOV results for different design variants

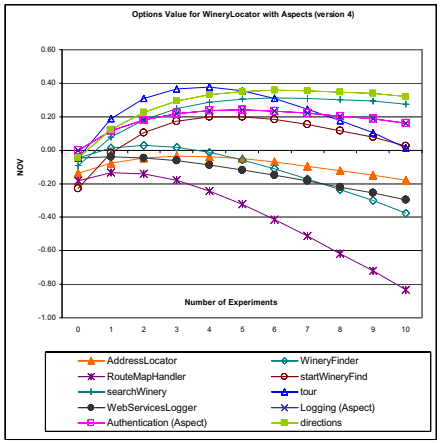
Design	ID	NOV	I_c %	I_s %	I_{w1} %
StoreLocator	s	0.72	NA	NA	NA
WineryLocator	w1	1.38	91.41	91.42	NA
WineryLocator with Logging	w2	1.41	2.18	95.6	2.18
WineryLocator with design rules for application	w3	1.59	12.59	120.2	15.05
WineryLocator with Aspects	w4	1.76	24.55	143.6	27.28

I_c = Cumulative increase in value
I_s = Net increase in value with respect to 's'
I_{w1} = Net increase in value with respect to 'w1'

Observations (2)

- Effect of Aspect-oriented modularization on option values





Conclusions

- Results
  - Aspects can be treated as modules (of design)
  - Aspects add value to an existing design
    - Aspect oriented modularization makes augmentation more profitable even if the added modules have comparatively low technical potential
- Limitations
  - Only two models of aspect oriented mechanism considered
  - General Expression for NOV used instead of individual expressions for operators
  - Distinction between aspect oriented modularization and inversion still is unclear
  - Simple heuristics for determining technical potential of the modules
  - feasibility of modeling and evaluating finer design changes (e.g. various forms of big and small refactorings)
  - Applying standard DSM operations on DSMs for software
  - Limitations of DSMs

Finally

- Questions ?
- Comments
- Thank You!