Information Hiding in KWIC

D. L. Parnas. On the Criteria To Be Used in Decomposing Systems into Modules. CACM 15(12):1053-1058, Dec 1972.

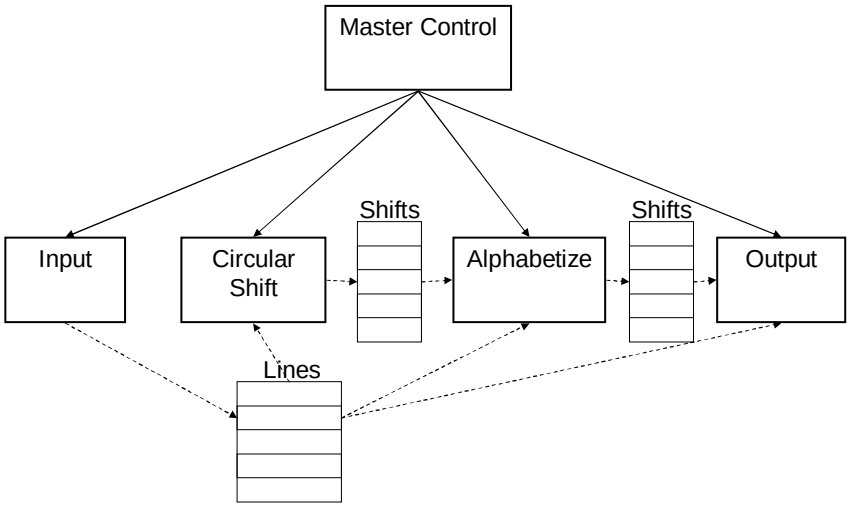
17-654/17-754: Analysis of Software Artifacts

Jonathan Aldrich

Key Word In Context

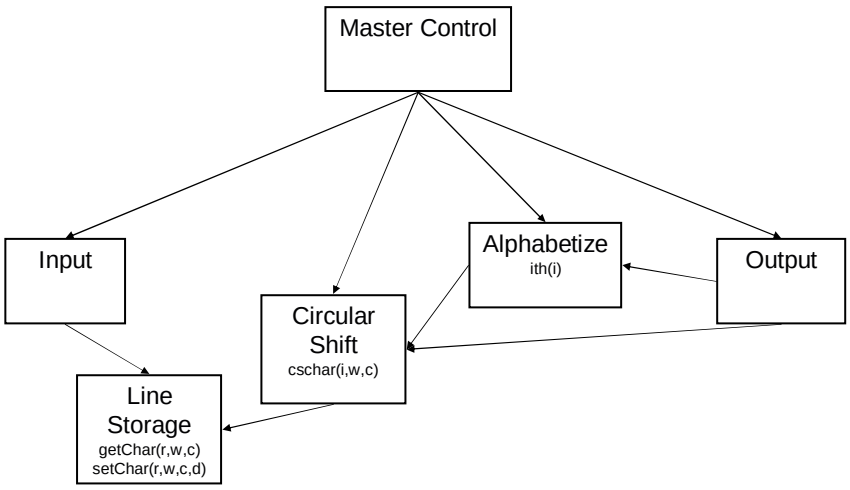
- "The KWIC [Key Word In Context] index system accepts an ordered set of lines, each line is an ordered set of words, and each word is an ordered set of characters. Any line may be "circularly shifted" by repeatedly removing the first word and appending it at the end of the line. The KWIC index system outputs a listing of all circular shifts of all lines in alphabetical order."
  - Parnas, 1972
- How would you design the architecture of this system?

KWIC Modularization #1



3 May 2006

KWIC Modularization #2



3 May 2006

KWIC Observations

- Similar at run time
  - May have identical data representations, algorithms, even compiled code
- Different in code
  - Understanding
  - Documenting
  - Evolving

Software Change

- …accept the fact of change as a way of life, rather than an untoward and annoying exception.
  —Brooks, 1974
- Software that does not change becomes useless over time.
  —Belady and Lehman
- For successful software projects, most of the cost is spent evolving the system, not in initial development
  - Therefore, reducing the cost of change is one of the most important principles of software design

Information Hiding

- Decide what design decisions are likely to change and which are likely to be stable
- Put each design decision likely to change into a module
- Assign each module an interface that hides the decision likely to change, and exposes only stable design decisions
- Ensure that the clients of a module depend only on the stable interface, not the implementation
- Benefit: if you correctly predict what may change, and hide information properly, then each change will only affect one module
  - That's a big if…do you believe it?

Abstraction

- Noun: A representation of some object that focuses on more important information and leaving out less important information – Edward Berard
  - The details (less important information) may be specified separately from the abstraction
- Verb: To come up with such an abstraction
- Distinct from information hiding
  - You're leaving out "less important" information, vs. information likely to change

Encapsulation

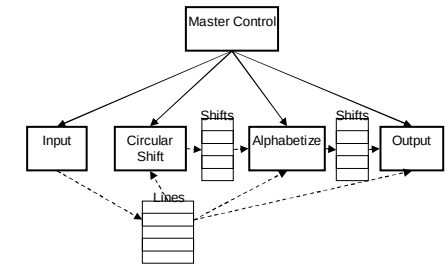
- Noun: a package or enclosure that holds one or more items
- Verb: to enclose one or more items in a container

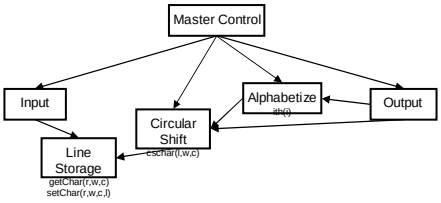
- SE: a *language mechanism* for ensuring that clients of a module do not depend on its implementation
  - e.g. Java's public/private
- A matter of degree
  - C: must define structs used by clients
  - Java: hides all syntactic dependences
    - But semantic dependencies may remain

- Could I hide information in a language without encapsulation mechanisms?

Effect of Change?

- Change input format
- Don't store all lines in memory at once
- Avoid packing 4 characters to a word
- Store the shifts directly instead of indexing
- Amortize alphabetization over searches





Effect of Change?

- Change input format
  - Input module only
- Don't store all lines in memory at once
  - Design #1: all modules
  - Design #2: Line Storage only
- Avoid packing 4 characters to a word
  - Design #1: all modules
  - Design #2: Line Storage only
- Store the shifts directly instead of indexing
  - Design #1: Circular Shift, Alphabetizer, Output
  - Design #2: Circular Shift only
- Amortize alphabetization over searches
  - Design #1: Alphabetizer, Output, and maybe Master Control
  - Design #2: Alphabetizer only

Other Factors

- Independent Development
  - Data formats (#1) more complex than data access interfaces (#2)
  - Easier to agree on interfaces in #2 because they are more abstract
- Comprehensibility
  - Design of data formats depends on details of each module
  - More difficult to understand each module in isolation

Decomposition Criteria

- Functional decomposition
  - Break down by major processing steps
- Information hiding decomposition
  - Each module is characterized by a design decision it hides from others
  - Interfaces chosen to reveal as little as possible about this

A Note on Performance

- Parnas says that if we are not careful, decomposition #2 will run slower
- He points out that a compiler can replace the function calls with inlined, efficient operations
- This is 1972!
  - But we still hear silly arguments about how (otherwise better) designs are slower
  - Smart compilers enable smart designs

Design Structure Matrices

K.J. Sullivan, W.G. Griswold, Y. Cai, and B. Hallen. The Structure and Value of Modularity in Software Design. Foundations of Software Engineering, 2001.

Carliss Baldwin and Kim Clark. Design Rules: The Power of Modularity. MIT Press.

17-654/17-754: Analysis of Software Artifacts

Jonathan Aldrich

Design Structure Matrices

	A	B	C
A	.		
B	X	.	X
C		X	.

Figure 1: DSM for a design of three parameters.

- Goal: to capture dependencies in the structure of a design
- A, B, and C are *design parameters*
  - A choice about some aspect of a design
- X means row depends on column
  - B is *hierarchically dependent* on A
    - If you change A, you might have to change B as well
    - Suggests you should make a decision about A first
  - B and C are *interdependent*
  - C and A are *independent*

Design Structure Matrices

	A	B	C
A	.		
B	X	.	X
C		X	.

Figure 2: DSM for a proto-modular design.

- Lines show clustering into proto-modules
  - Indicates several design decisions will be managed together
- True modules should be independent
  - i.e., no marks outside of its cluster
  - Not true here because B (in the B-C cluster) depends on A

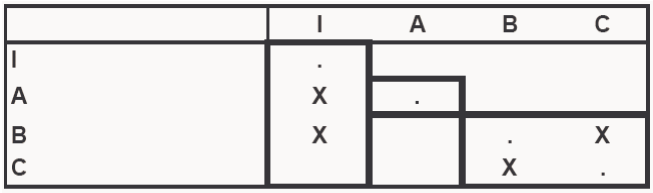
Design Structure Matrices

	I	A	B	C
I	.			
A	X	.		
B	X		.	X
C			X	.

Figure 3: DSM for a modular design obtained by splitting.

- Interface reifies the dependence as a design parameter
  - Instead of B depending on A, now A and B both depend on I
  - Serves to decouple A and B
  - Think of I as the interface of A

Value of Modularity

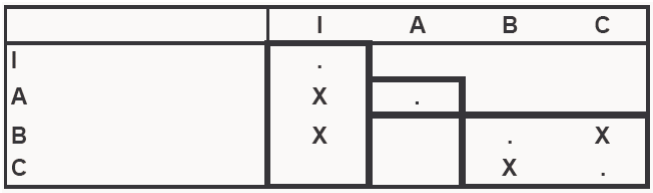


	I	A	B	C
I	.			
A	X	.		
B	X		.	X
C			X	.

Figure 3: DSM for a modular design obtained by splitting.

- Information Hiding
  - If you can anticipate which design decisions are likely to change and hide them in a module, then evolving the system when these changes occur will cost less
  - Reduces maintenance cost and time to market
  - Frees resources to invest in quality, features

Value of Modularity



	I	A	B	C
I	.			
A	X	.		
B	X		.	X
C			X	.

Figure 3: DSM for a modular design obtained by splitting.

- Option value of modules
  - The best design choice for A, B, and C may be uncertain and require experiments
  - Original design: B and C were dependent on A. Therefore if we build new A, B, C implementations, we must use all or reject all
  - New design: A and B,C decoupled through interface. We can build new A, B, and C implementations, and choose independently to use A and B,C
    - If one experiment fails we can still benefit from the others
  - Connection to economic: a portfolio of options is more valuable than an option on a portfolio

KWIC Design #1

	A	D	G	J	B	E	H	K	C	F	I	L	M
A - Input Type	.												
D - Circ Type		.											
G - Alph Type			.										
J - Out Type				.									
B - In Data					.	X	X						
E - Circ Data					X	.	X						
H - Alph Data					X	X	.						
K - Out Data								.					
C - Input Alg	X				X				.				
F - Circ Alg		X			X	X				.			
I - Alph Alg			X		X	X	X				.		
L - Out Alg				X	X		X	X				.	
M - Master	X	X	X	X									.

Interdependence of data formats

Many data dependences

Interface dependences follow calls

True modules

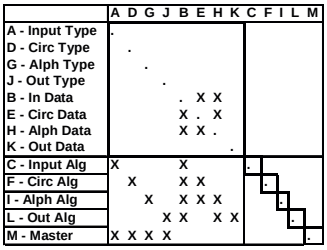
KWIC Design #2

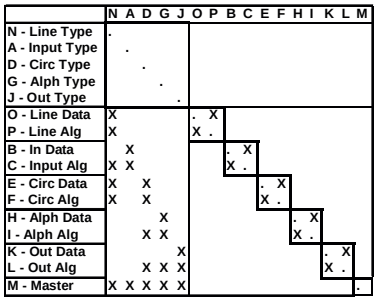
	N	A	D	G	J	O	P	B	C	E	F	H	I	K	L	M
N - Line Type	.															
A - Input Type		.														
D - Circ Type			.													
G - Alph Type				.												
J - Out Type					.											
O - Line Data	X					.	X									
P - Line Alg	X					X	.									
B - In Data		X						.	X							
C - Input Alg	X	X						X	.							
E - Circ Data	X		X							.	X					
F - Circ Alg	X		X							X	.					
H - Alph Data				X								.	X			
I - Alph Alg			X	X								X	.			
K - Out Data					X									.	X	
L - Out Alg			X	X	X									X	.	
M - Master	X	X	X	X	X											.

Dependence on interfaces

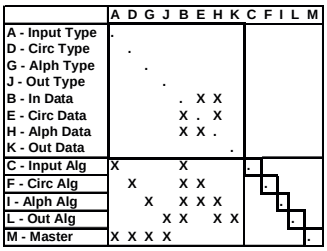
True modules

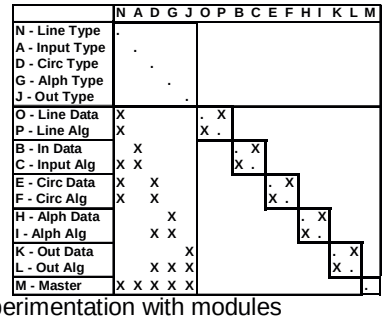
Which Design is More Valuable?





Options Value of Designs





- Both designs allow independent experimentation with modules
- However, in the first design the modules are tightly constrained
  - Dependencies on many data structures
  - Very few options for experimentation
- The second design is much less constrained
  - Interfaces are more abstract
  - Fewer dependencies
  - Modules in design #2 have a higher “technical potential”
  - Therefore experimentation in #2 is more likely to yield benefits

EDSMs: Considering Possible Changes

- Environment and Design Structure Matrices
  - Sullivan et al., ESEC/FSE 2001
- Add changes as *environmental parameters*
  - *Note: slightly more concrete than what Sullivan et al. propose*
  - Only partially controlled by designer
  - May affect each other
  - May affect design decisions in code
- What interfaces are affected?
  - Information hiding: interfaces should be stable
- What implementations are affected?
  - Information hiding hypothesis: should be local to a module

Effect of Change – Design #1

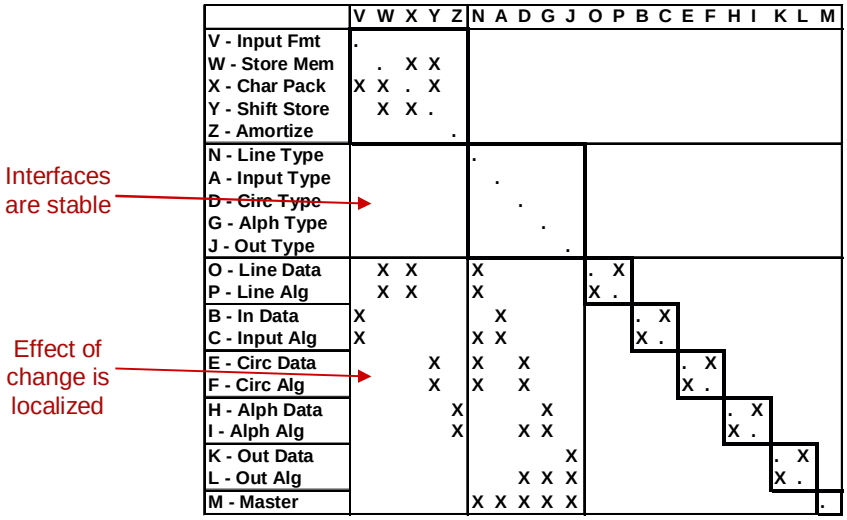
Interdependence of changes

Unstable data interfaces depend on changes

Algorithms depend on data

	V	W	X	Y	Z	A	D	G	J	B	E	H	K	C	F	I	L	M
V - Input Fmt	.																	
W - Store Mem		.		X	X													
X - Char Pack	X	X	.		X													
Y - Shift Store		X		X	.													
Z - Amortize					.													
A - Input Type						.												
D - Circ Type							.											
G - Alph Type								.										
J - Out Type									.									
B - In Data		X		X						.		X	X					
E - Circ Data					X					X		.	X					
H - Alph Data					X	X				X		X	.					
K - Out Data													.					
C - Input Alg	X	X		X		X				X				.				
F - Circ Alg		X		X	X		X			X		X			.			
I - Alph Alg		X		X	X	X		X		X		X	X			.		
L - Out Alg		X		X	X	X			X	X			X	X			.	
M - Master					X	X	X	X										.

Effect of Change – Design #2



	V	W	X	Y	Z	N	A	D	G	J	O	P	B	C	E	F	H	I	K	L	M
V - Input Fmt	.																				
W - Store Mem		.	X	X																	
X - Char Pack	X	X	.	X																	
Y - Shift Store		X	X	.																	
Z - Amortize					.																
N - Line Type						.															
A - Input Type							.														
D - Circ Type								.													
G - Alph Type									.												
J - Out Type										.											
O - Line Data		X	X			X					.	X									
P - Line Alg		X	X			X					X	.									
B - In Data	X						X						.	X							
C - Input Alg	X					X	X						X	.							
E - Circ Data				X		X		X							.	X					
F - Circ Alg				X		X		X							X	.					
H - Alph Data					X				X								.	X			
I - Alph Alg					X			X	X								X	.			
K - Out Data										X									.	X	
L - Out Alg								X	X	X									X	.	
M - Master						X	X	X	X	X											.

Summary

- DSMs are a structured way of thinking about the value of design
  - Are design decisions isolated to a module, or do they affect several modules?
  - How do modules depend on interfaces?
  - On which parts of the system can I experiment independently?
  - How much value is there in the experiments?
    - Technical potential of the module
- EDSMs incorporate change scenarios
  - How are interfaces and code affected by change?
- More to explore
  - Baldwin and Clark – discuss value of modularity
  - Sullivan and Griswold – apply B&C to S/W, introduce EDSMs
  - Lattix LDM tool – derives DSMs from code