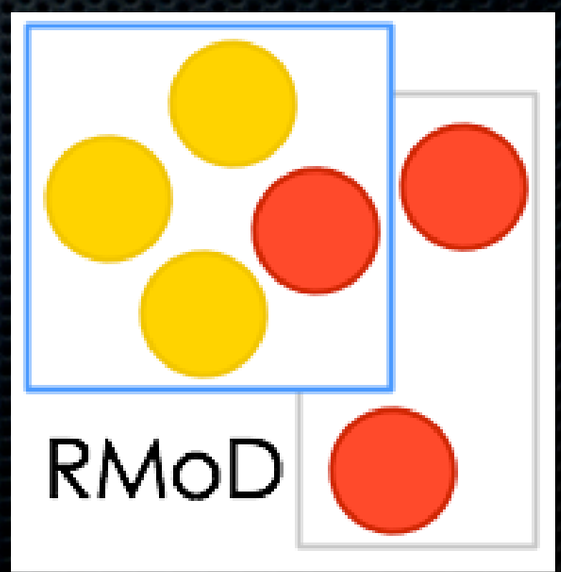
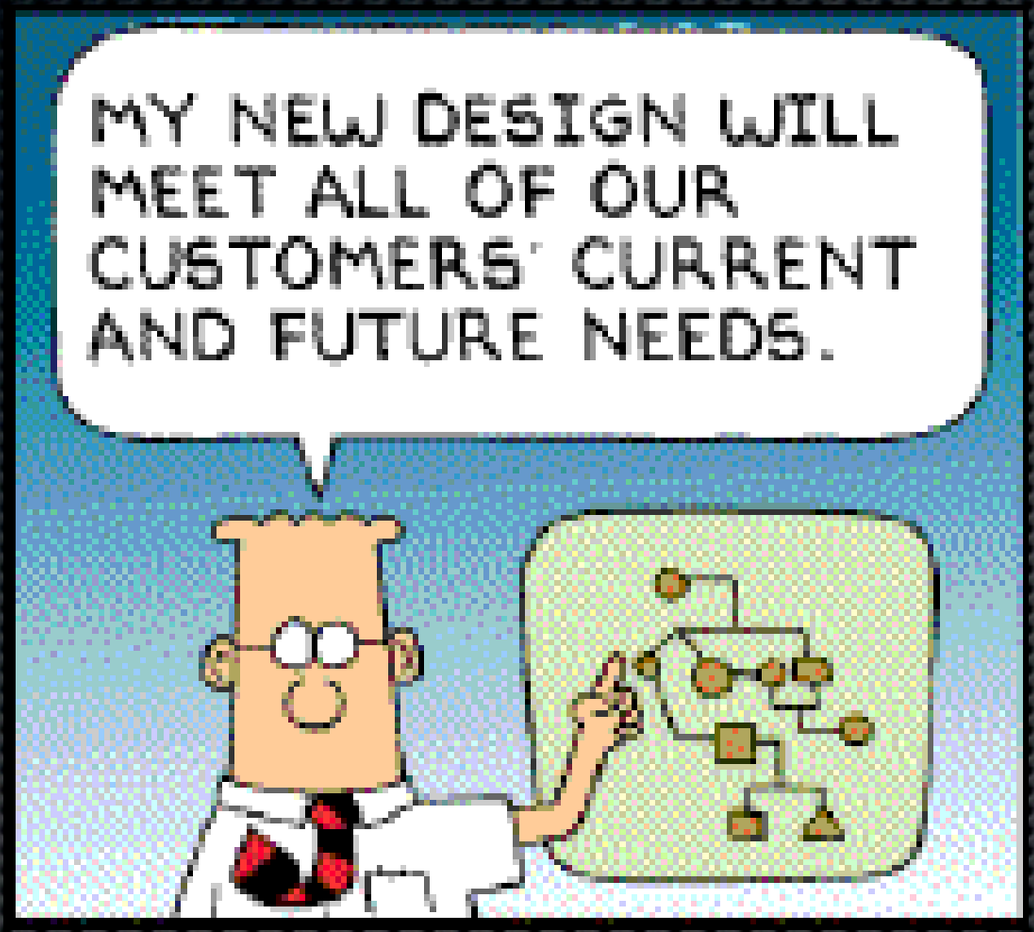
Software Evolution: a Maintenance Perspective

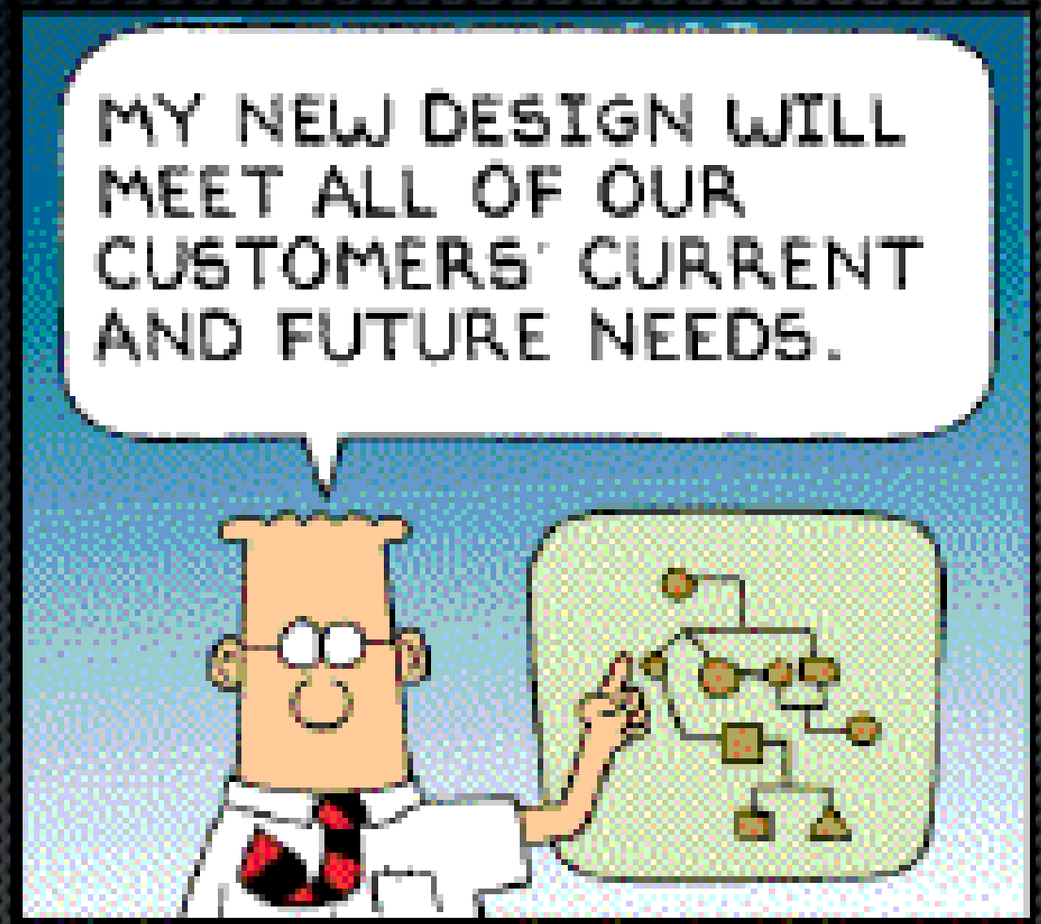
S. Ducasse

http://rmod.lille.inria.fr

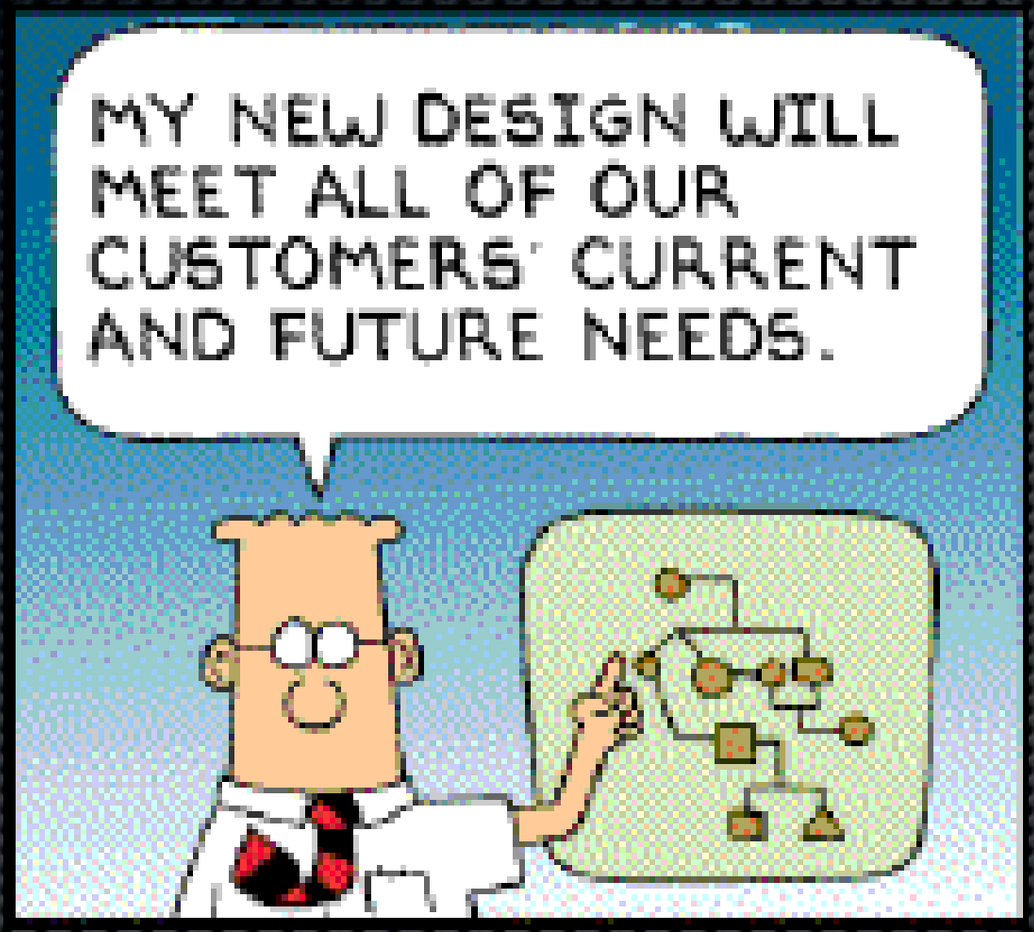


- http://stephane.ducasse.free.fr
- Co-creator of Moose
- Co-founder of http://www.synectique.eu
- Core http://www.pharo-project.org developer
- Coder and designer


MY NEW DESIGN WILL MEET ALL OF OUR CUSTOMERS' CURRENT AND FUTURE NEEDS.



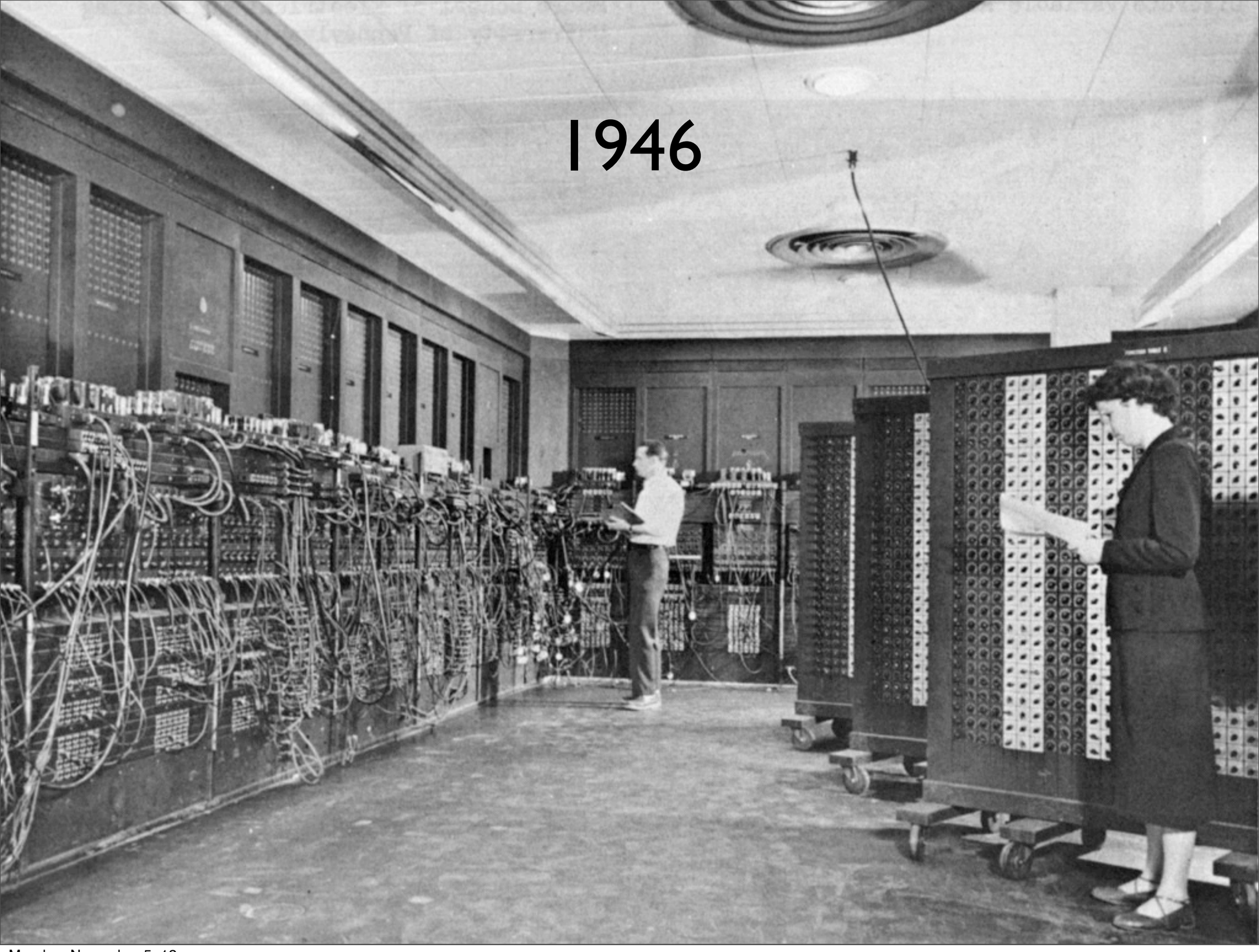
So end of the story?!



No just the graal

Software is

Complex


1946

1’000’000 lines of code

* 2s = 2’000’000 seconds

/ 3600 = 560 hours

/ 8 = 70 days

/ 20 = 3 months

Facts

- Cobol > 60% world software
- 70% of business applications
- Applications cobol handle 85%
- Cobol grows of 5 billions lines of code per year [eWeeks, 2001]

Counting a bit

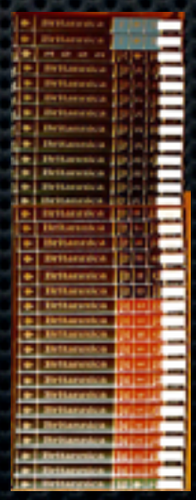
- 1 sheet ~ 60 lines of code
- Two sides ~ 120 loc

Windows NT 3.1 (1993)

- 4 à 5 MLOC

3.75m

3.2m



Encyclopedia Britanica
(15 ed., 32 volumes)

Windows server 2003
50 MLOC

41.m

46 m

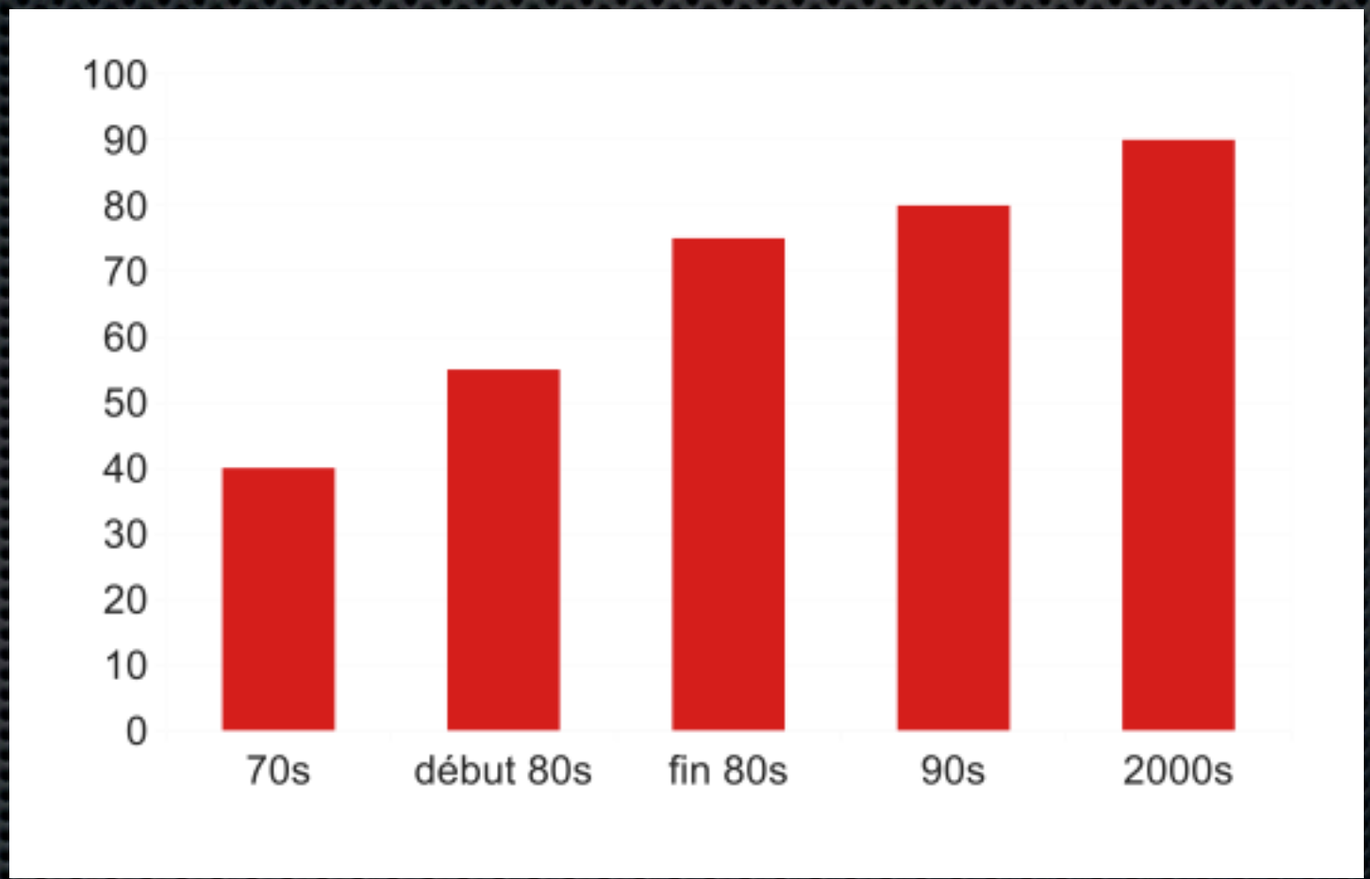
Business Relevance

- 1990 → 120 billions LOC in maintenance (Ulrich, 1990) 100 km height :)
- 2000 → 250 billions LOC in maintenance (Sommerville, 2000)
- Maintained code double every 7 years (Müller et al., 1994)

$$$

- US Yearly cost > $70 billions (Sutherland, 1995; Edelstein, 1993)
- Nokia spent $90 millions on 2000 migration
- US Federal Gov spent $8.38 Billions over 5 yers for the 2000 migration

% maintenantce in total cost



What? It still exist?

- Advanced languages (OO, AOP)
- Modern Processes (RUP, Agiles)
- Quality (CMMi)
- New Development (MDE, SOA)

One upon a time

- Un marchand de moules construit un magasin à Dunkerque ...

Il était une fois ...

- Les affaires marchent bien

Il était une fois ...

- Vraiment bien

Il était une fois ...

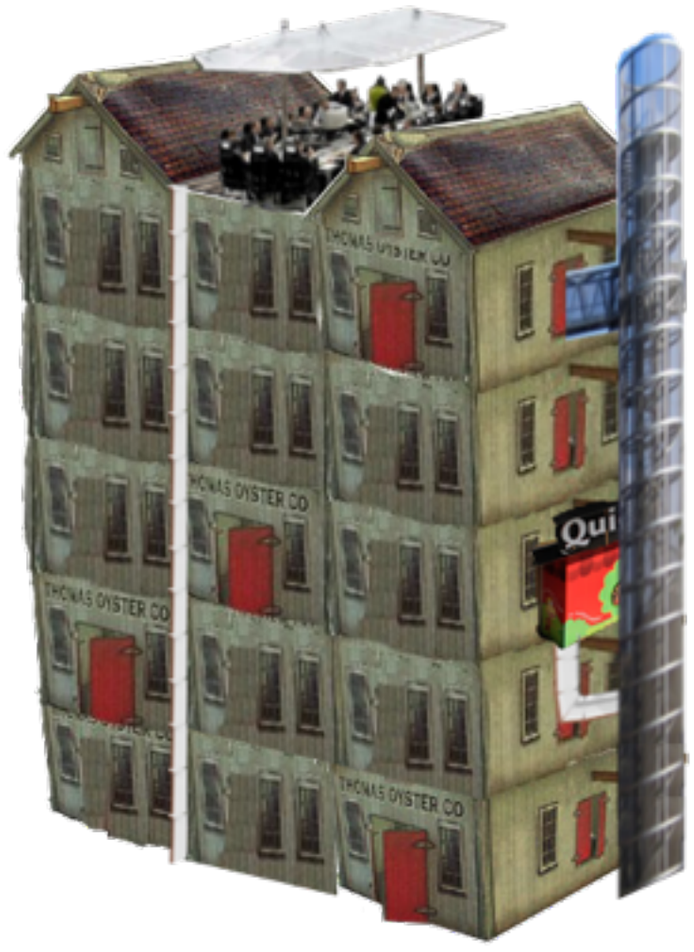
- Les employés veulent un restaurant

Il était une fois ...

- Les directeurs, une terrasse

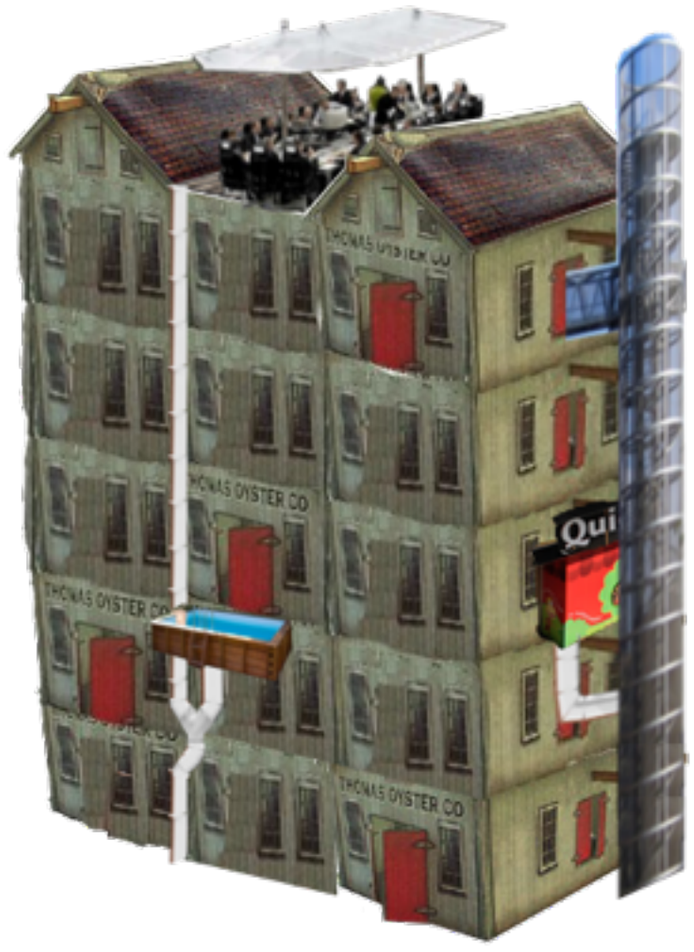
Il était une fois ...

- La loi impose une sortie de secours



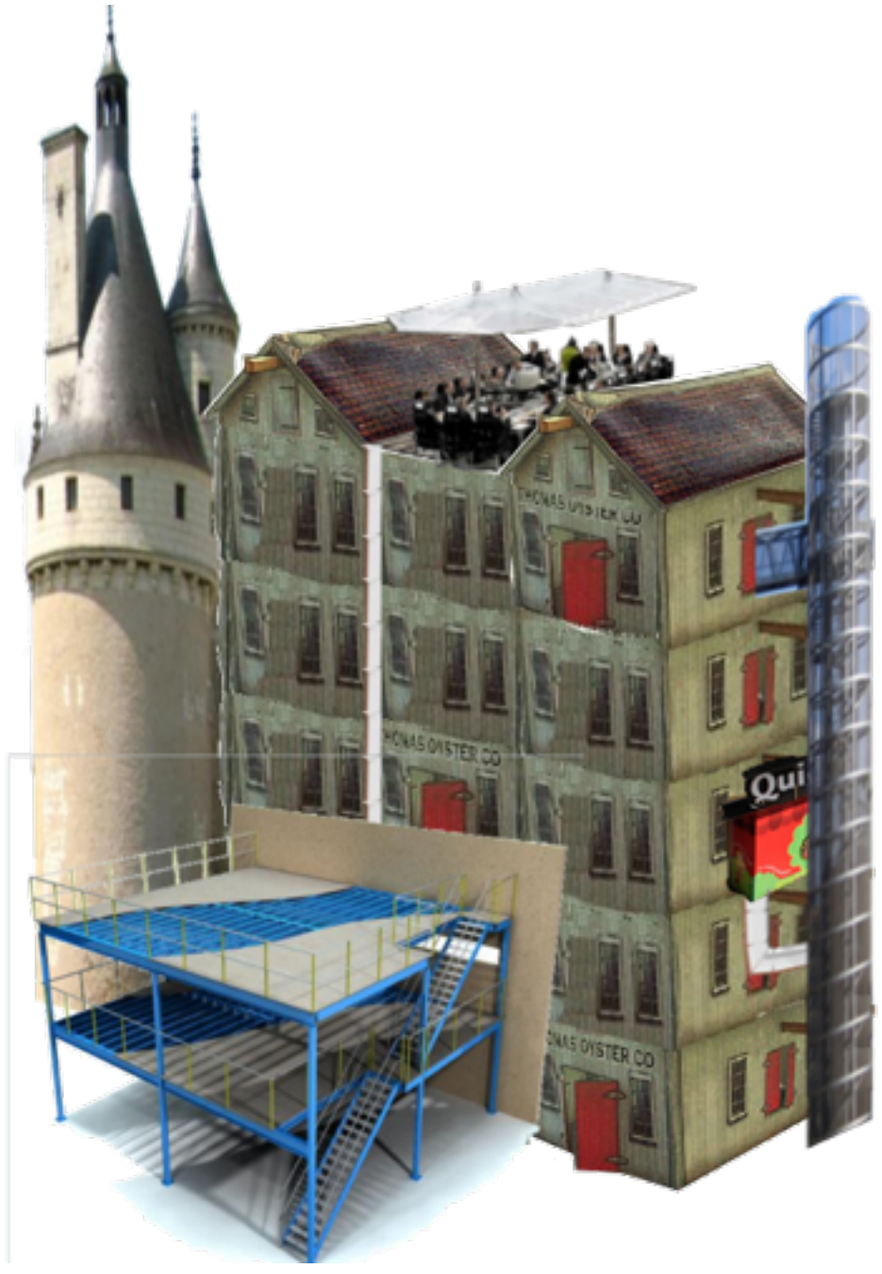
Il était une fois ...

- La concurrence offre des *goodies* aux clients, l'entreprise … une piscine !



Il était une fois ...

- etc …



Laws of software evolution

- Continuing change
  - A program that is used in a real-world environment must change, or become progressively less useful in that environment.
- Increasing complexity
  - As a program evolves, it becomes more complex, and extra resources are needed to preserve and simplify its structure.

Software is a living entity...

- Early decisions were certainly good at that time
- But the context changes
- Customers change
- Technology changes
- People change

Maintenance = Success!!

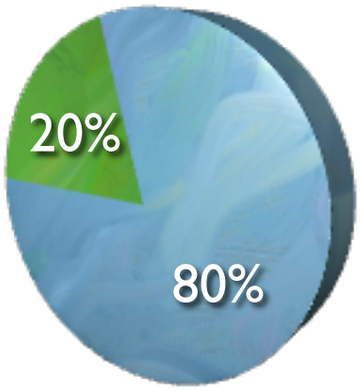
We only maintain useful successful software

Maintenance is controlled by external factors (Success, laws, people...) and not driven by software

- Having better languages does not simplify really maintenance (C++, template, overriding, oop, ...)

- Having better languages just make sure that we will build richer systems

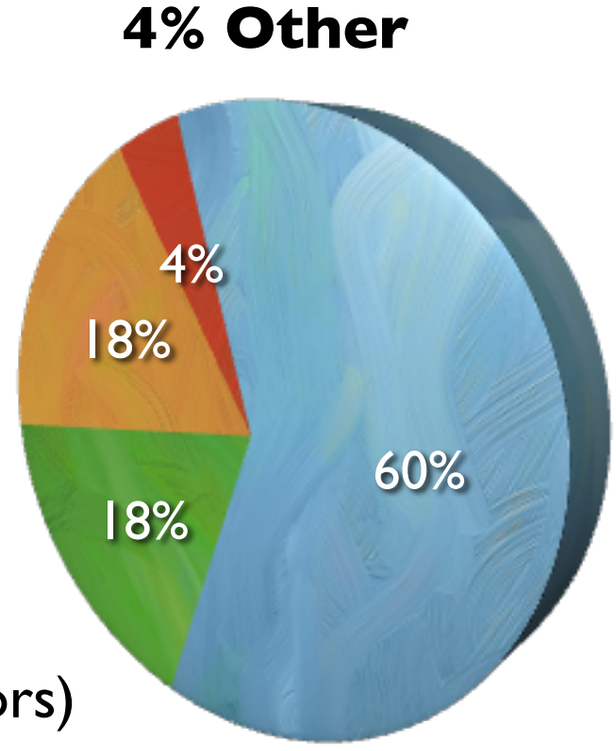
Maintenance is *continuous* Development



Between **50**% and **80**% of ***global*** effort is spent on "maintenance" !

18% Adaptive
(new platforms or OS)

18% Corrective
(fixing reported errors)



60% Perfective
(new functionality)

"Maintenance"

- “**Forward Engineering** is the traditional process of moving from high-level abstractions and logical, implementation-independent designs to the physical implementation of a system.”
- “**Reverse Engineering** is the process of analyzing a subject system to identify the system’s components and their interrelationships and create representations of the system in another form or at a higher level of abstraction.”
- “**Reengineering** ... is the examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form.”

Goals

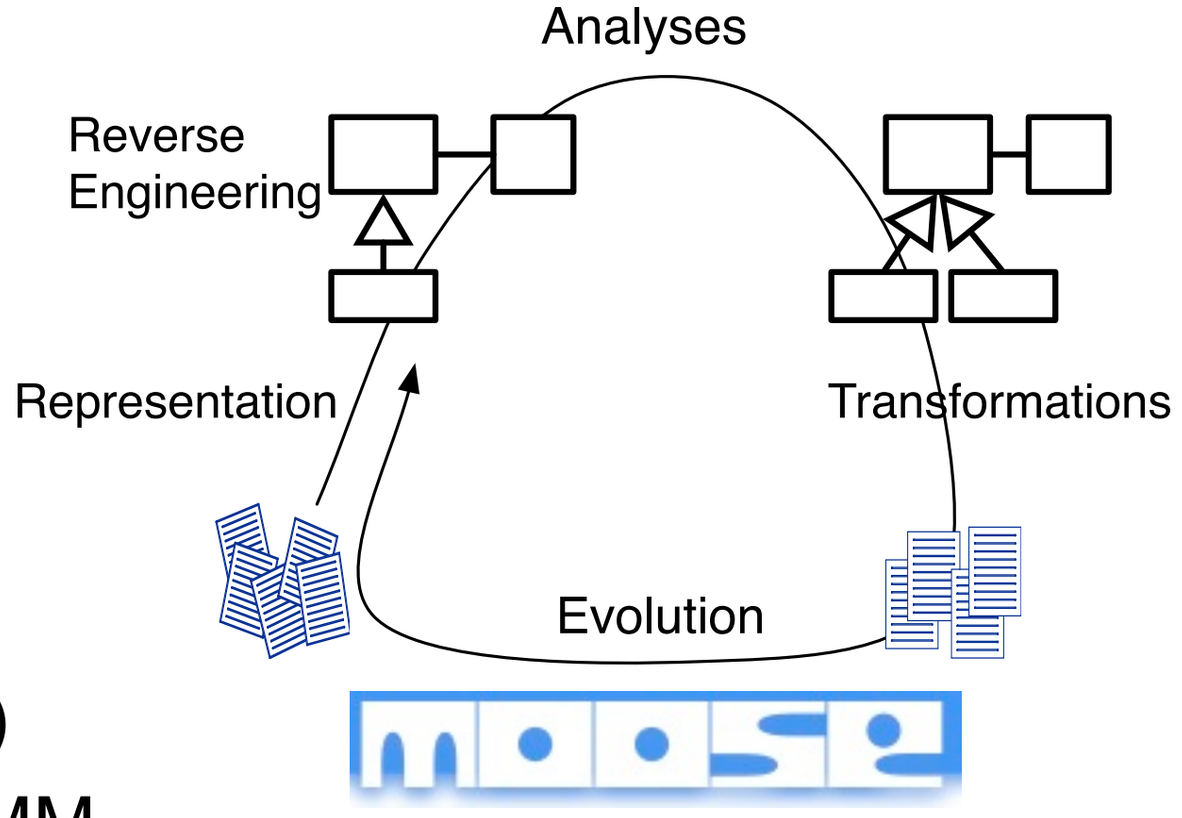
- System daily duties
- System assessment
- System reorganization
- Migration

RMOD

RMoD: code analysis, metamodeling, software metrics, program understanding, program visualization, evolution analysis, refactorings, legacy code, quality, ...

Current focus

- Remodularization analyses
- Quality models (PSA-Airfrance)
- ***Towards semantic merge***
- Rule and bug assessment



Collaborations

Soft-VUB (Belgium), Pleiad (Chile)

UFMG (Brazil), SCG (Swiss), LIRMM

One picture is worth one thousand words

Which one?

How could it be that simple?

Program visualization is difficult

Limited number of colors: 12

Blur and color emergence

Limited screen size

Limited context, edges crossing

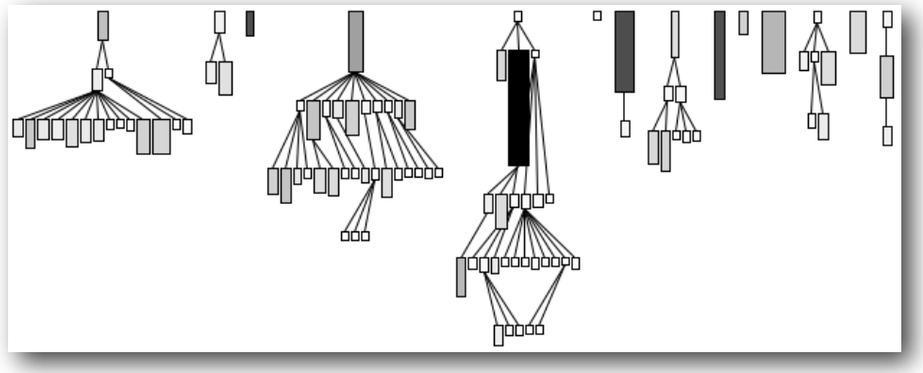
Limited short-term memory (three to nine)

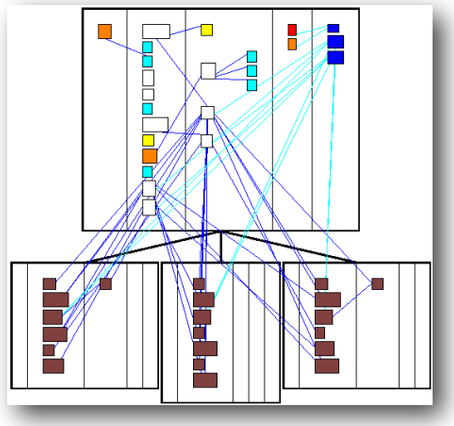
Difficult to remember too many symbols/semantics

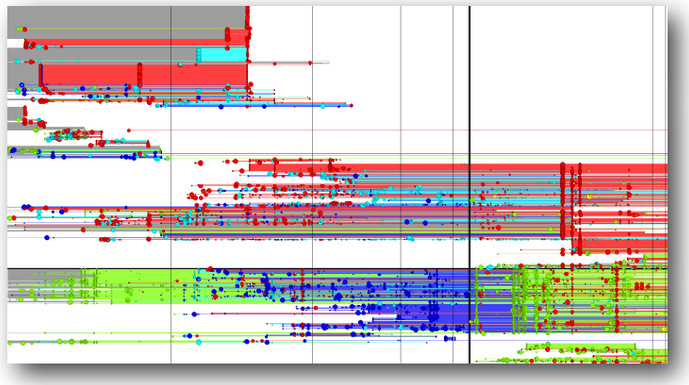
Culture, Colorblind

Visualization principles in 3 min

- Preattentive visualization (unconscious < 200ms)
- Gestalt principles (from 1912)
- 70% of our sensors are dedicated to vision







How many 5?

3332123466509000096766689877835367
78667609109198189717464330398217 68
3446786586088022116768768778976 2

How many 5?

3332123466509000096766689877835367

78667609109198189717464330398217 68

3446786586088022116768768778976 2

Preattentive attributes

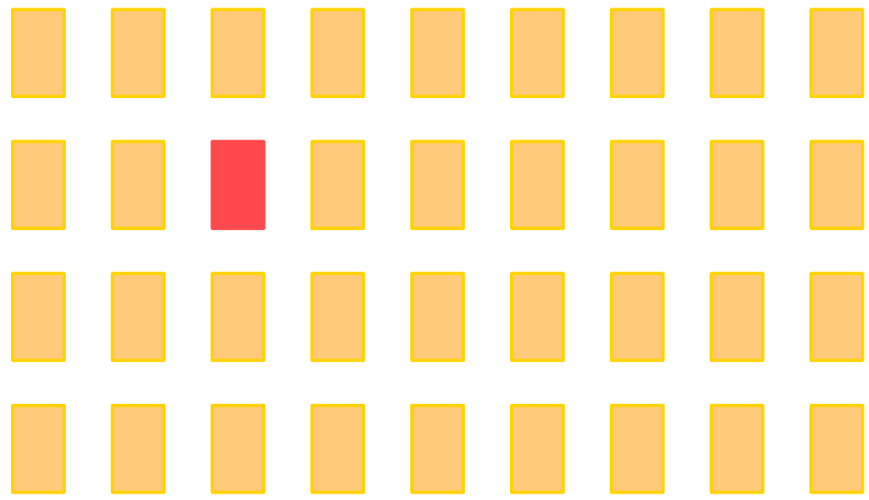
Color intensity

Form: orientation, line length, line width, size, shape, added marks, enclosure

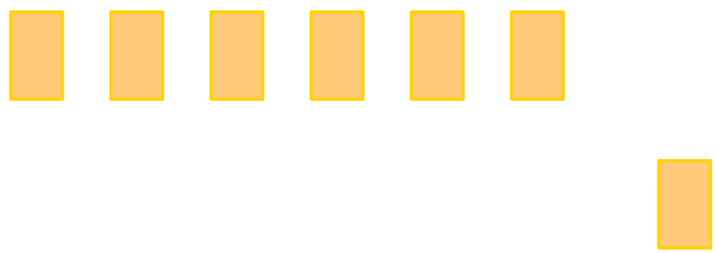
Spatial position (2D location)

Motion (flicker)

Color / intensity

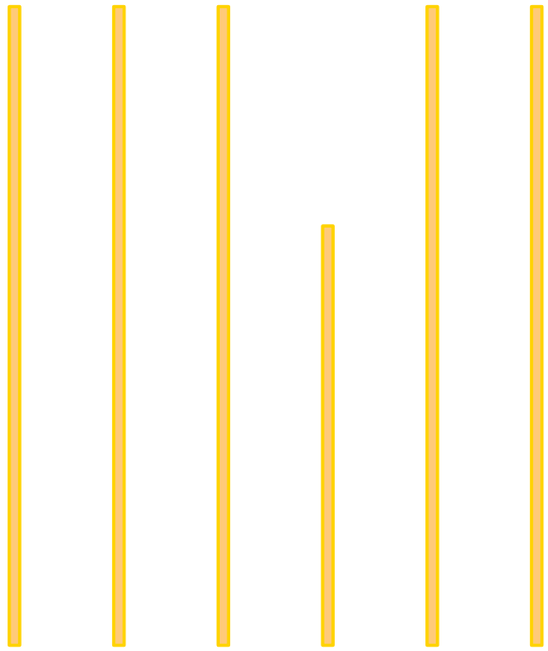


Position

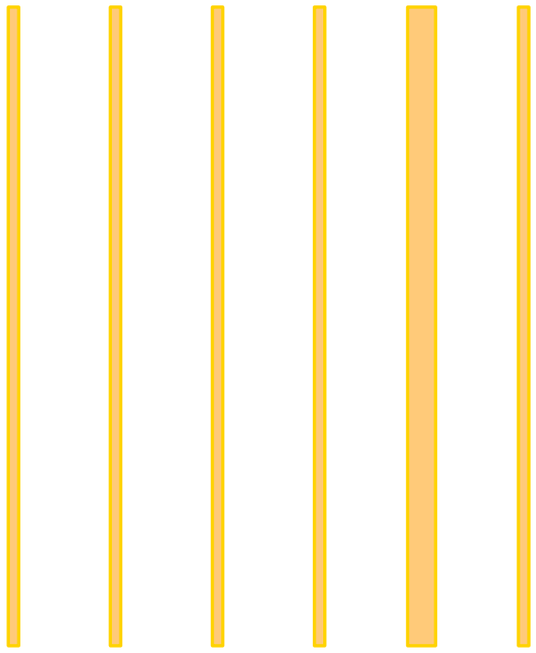


Form / Orientation

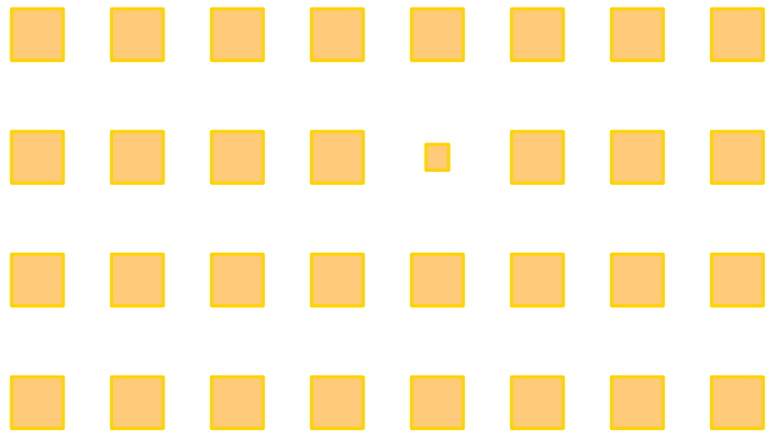
Form / Line length



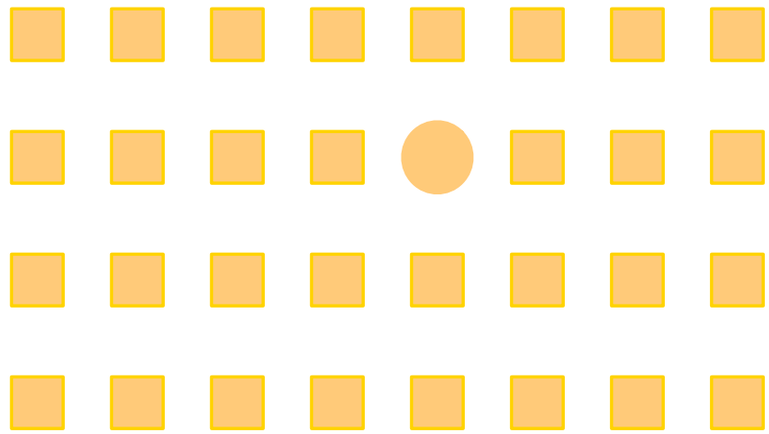
Form / Line width



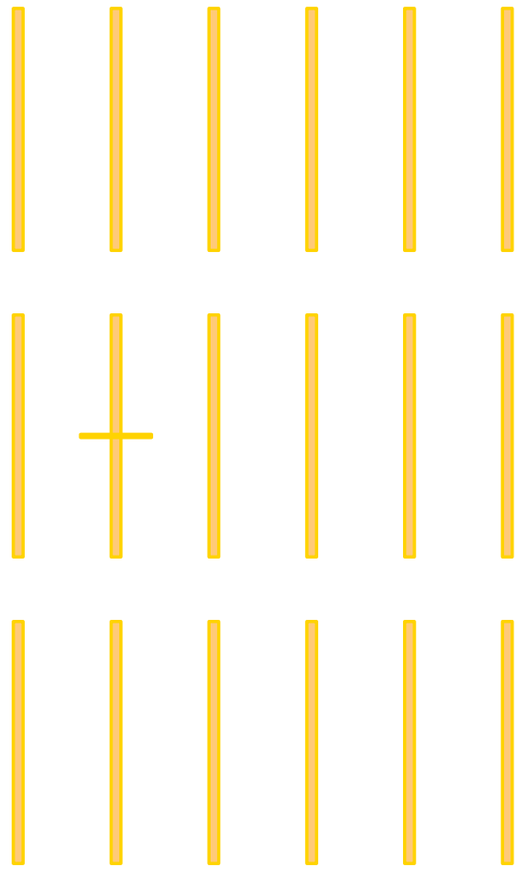
Form / Size



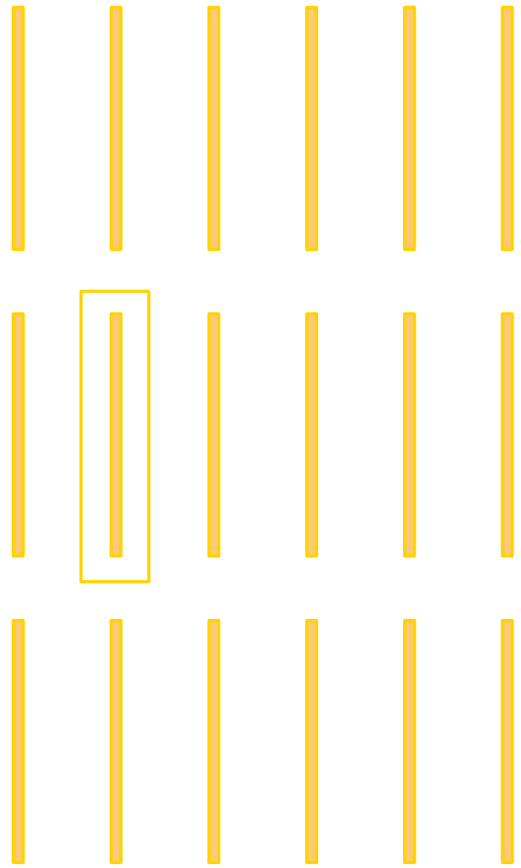
Form / Shapes



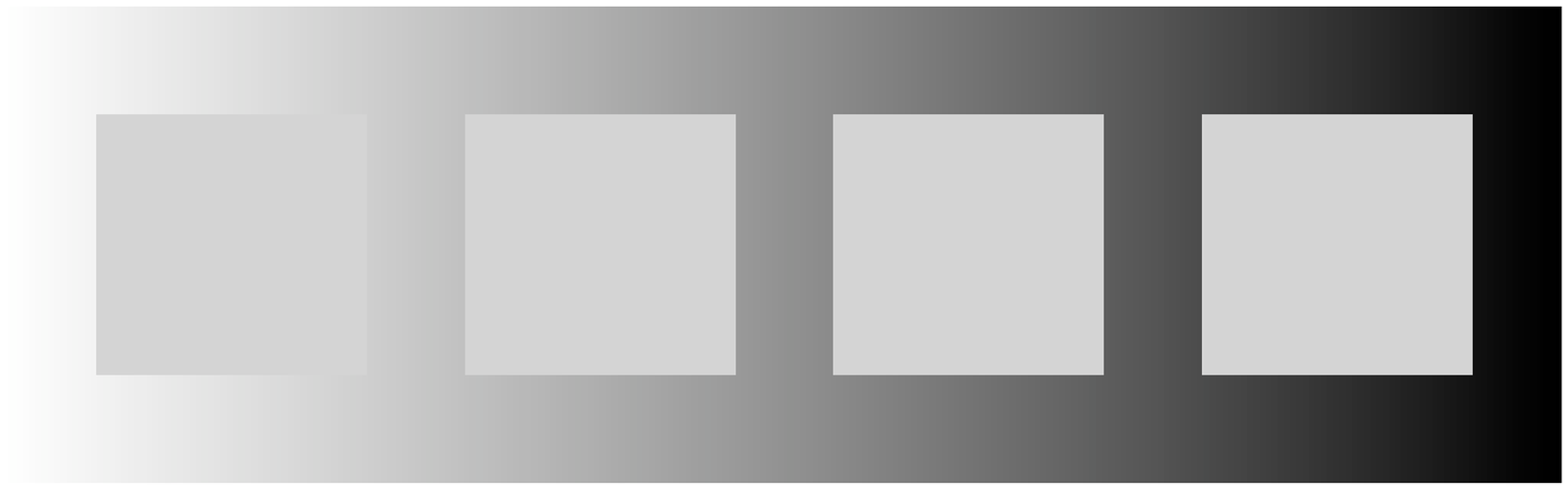
Form / Added marks



Form / Enclosure



Context



Gestalt Principles of Visual Perception

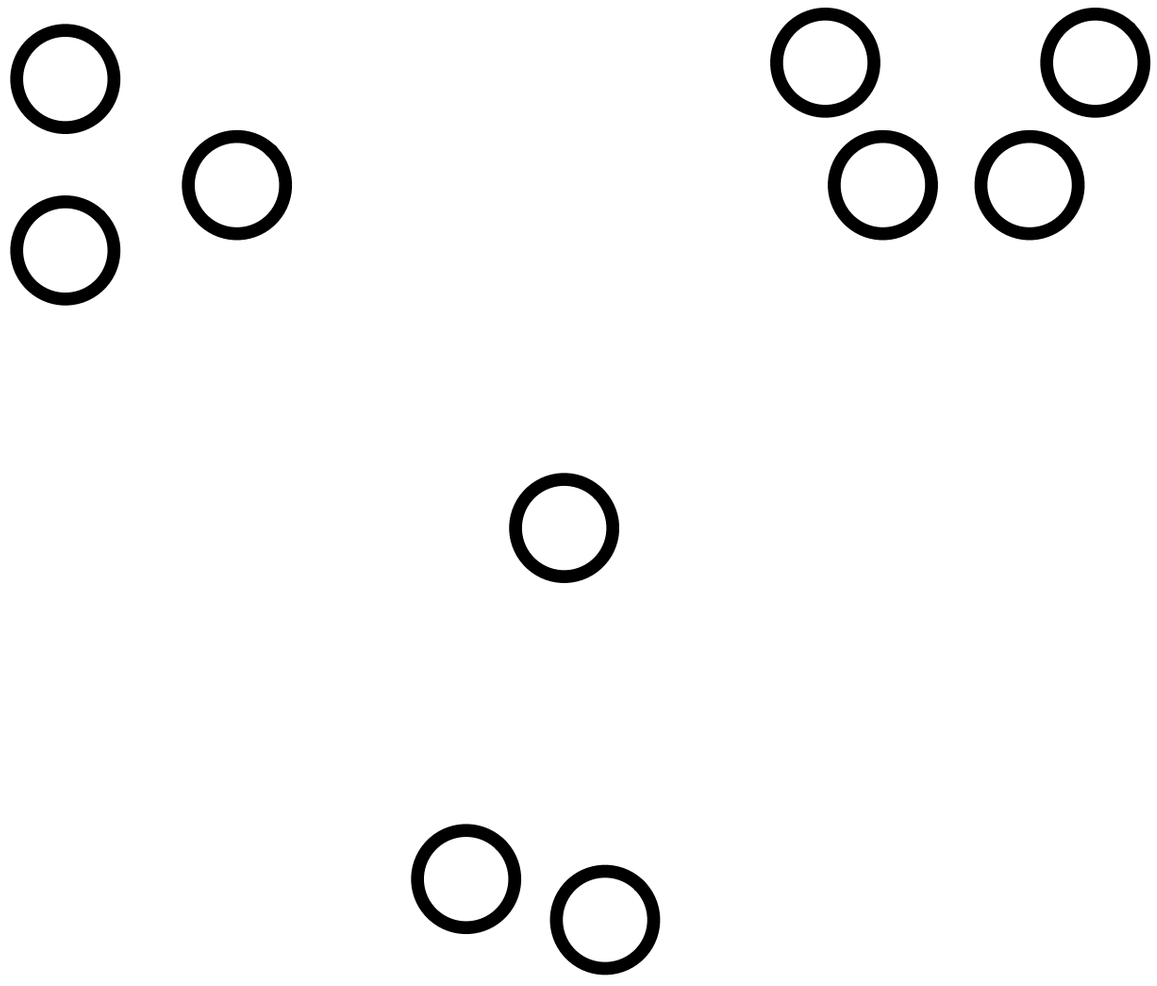
Back in 1912, from the Gestalt School of psychology

Still stand today

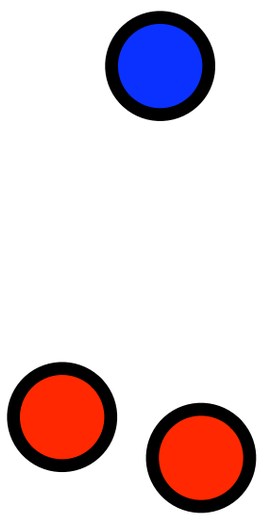
Gestalt means patterns

How do we perceive pattern, form, and organization?

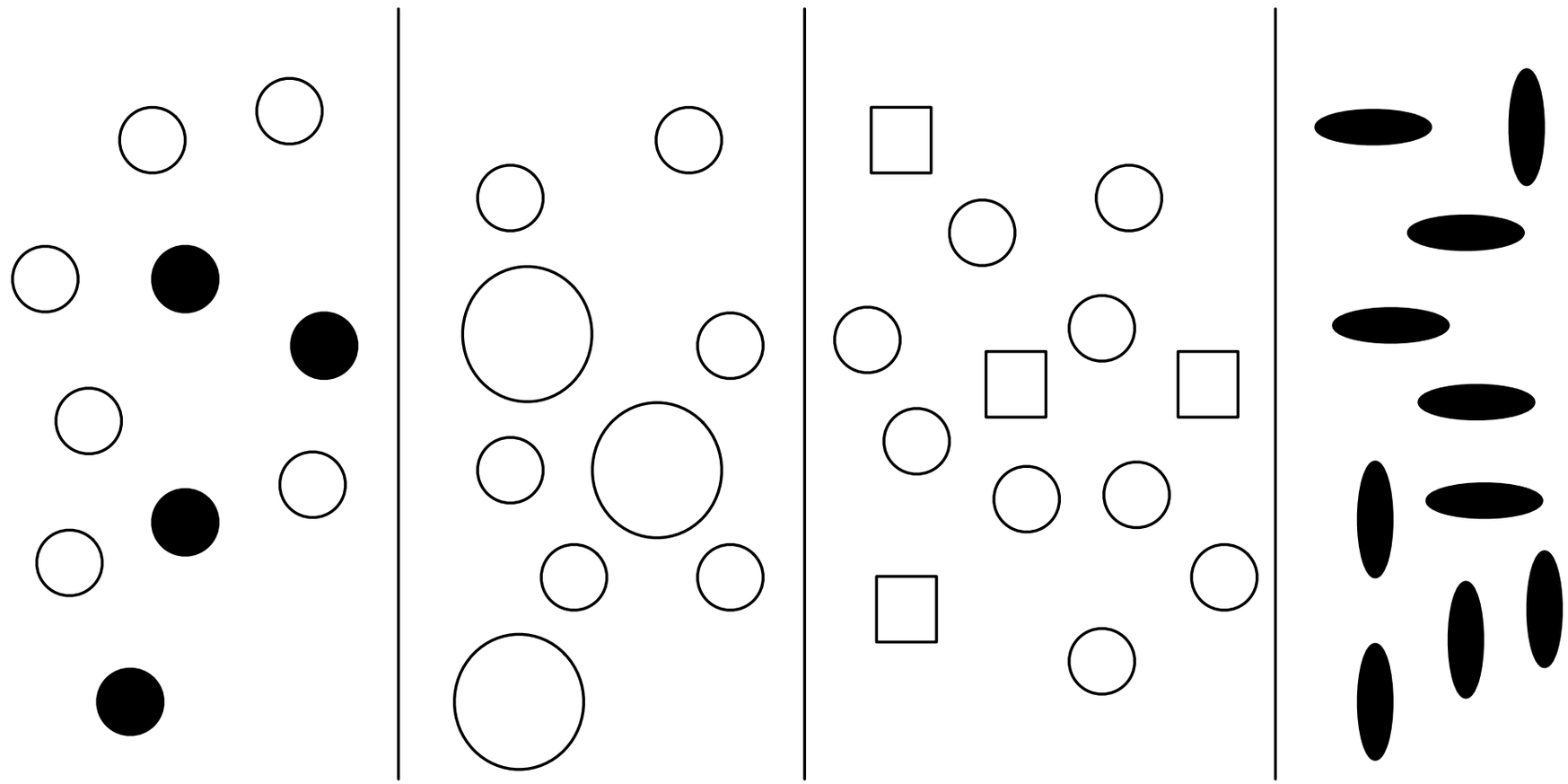
Principle of Proximity



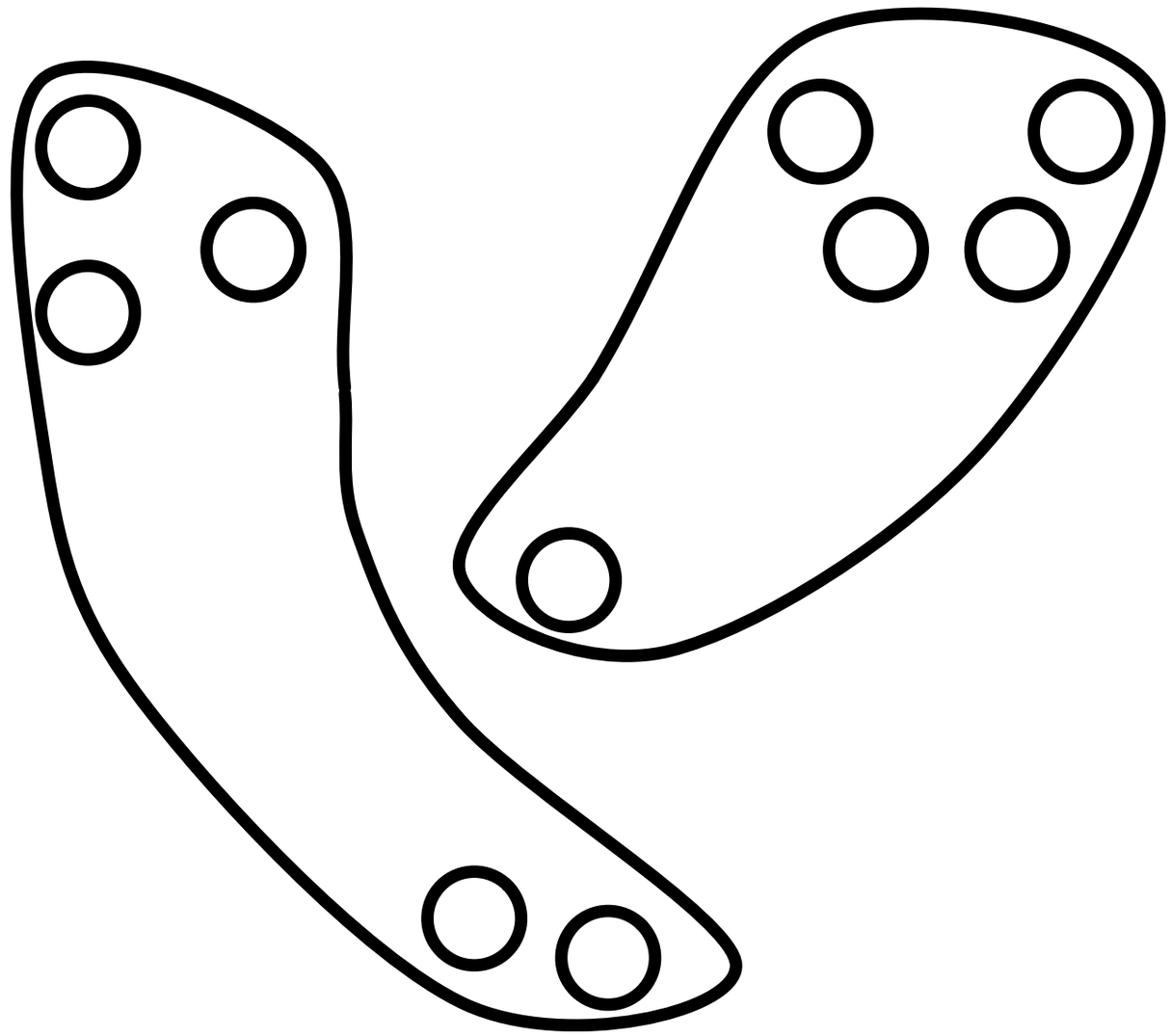
Principle of Similarity



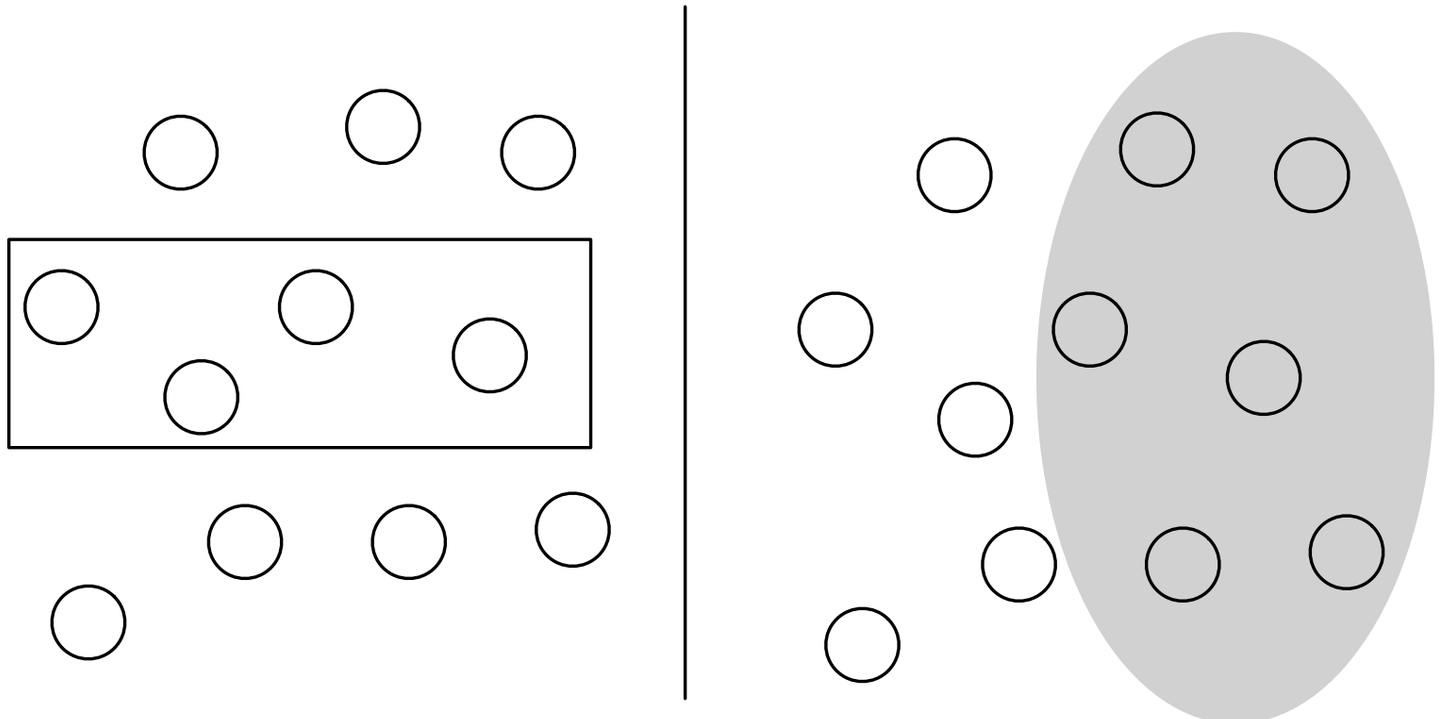
Principle of Similarity



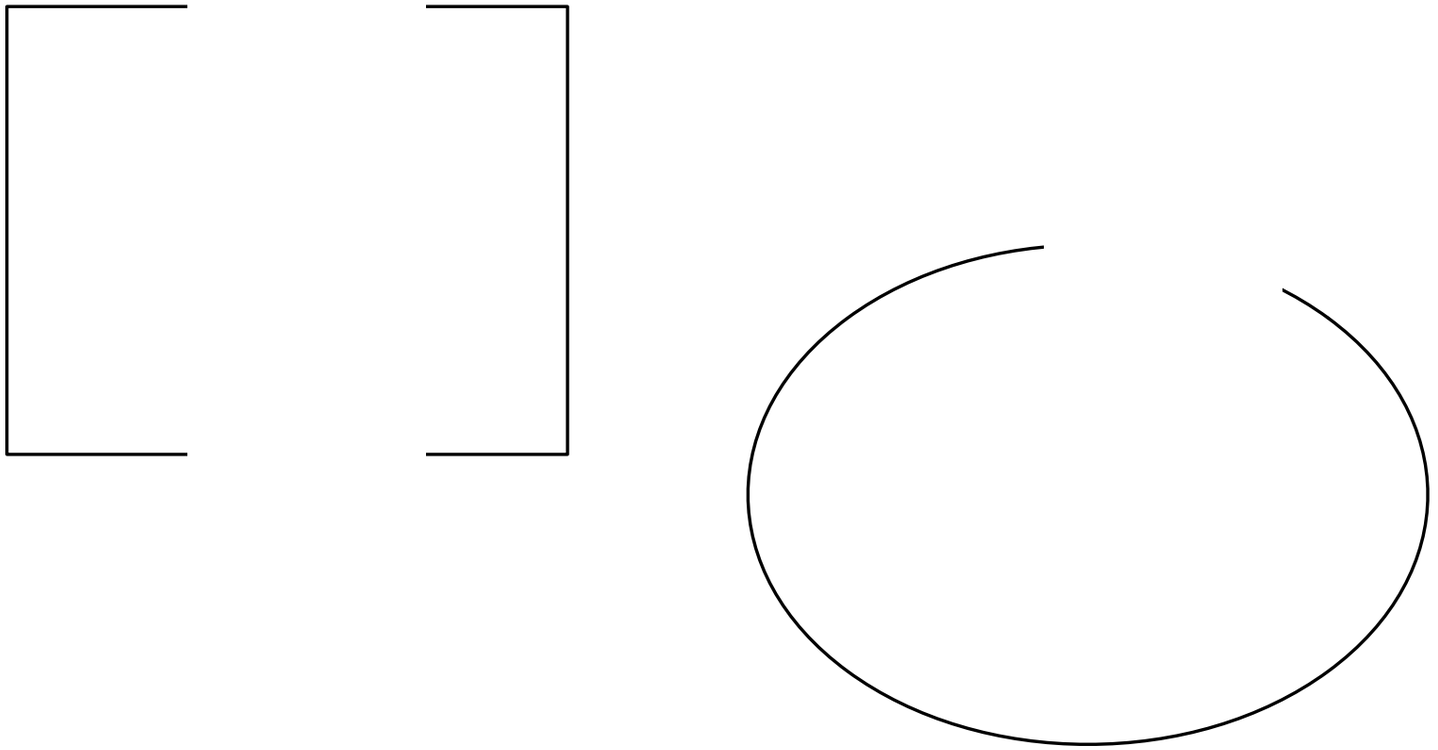
Principle of Enclosure



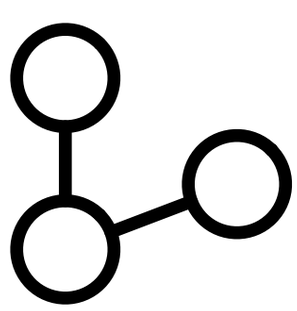
Principle of Enclosure

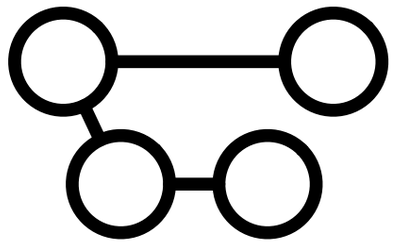


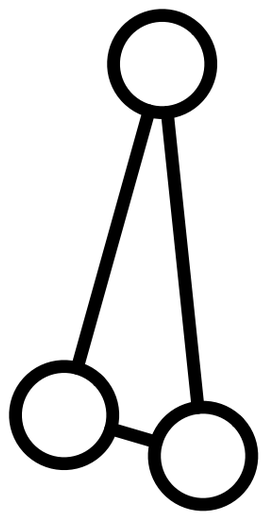
Principle of Closure



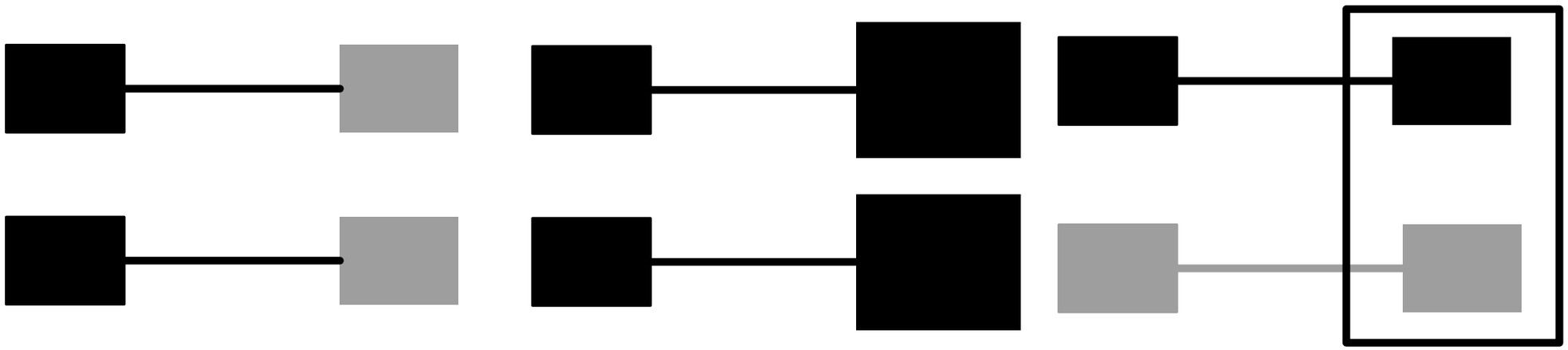
Principle of connectivity







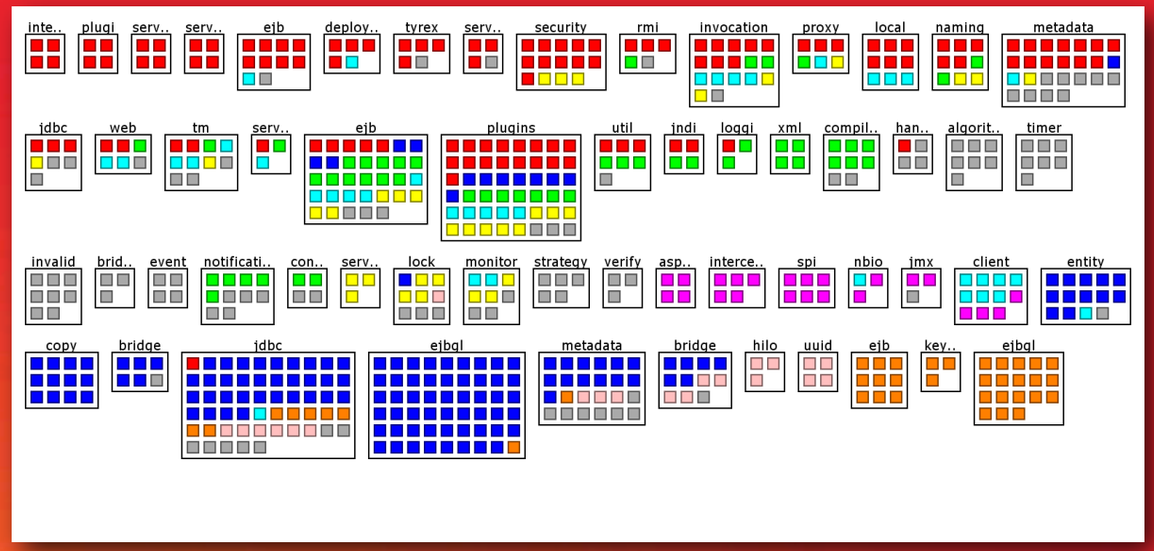
Principle of connectivity



How properties spread on a system?

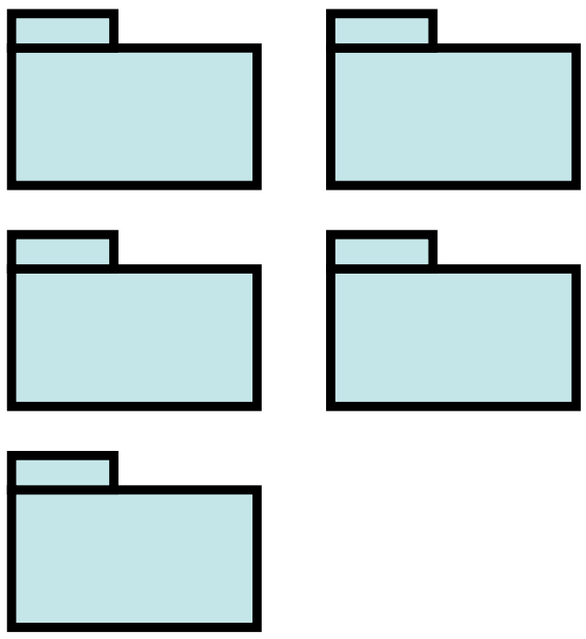
- Where author X worked?
- What are the classes under development the last two weeks?

- Distribution Map [ICSM]

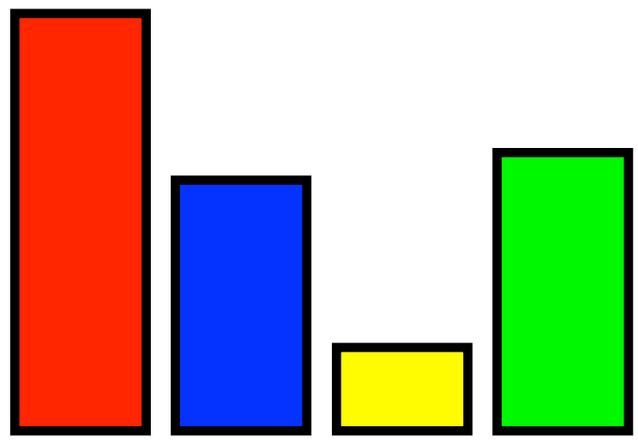


We take any two partitions, and

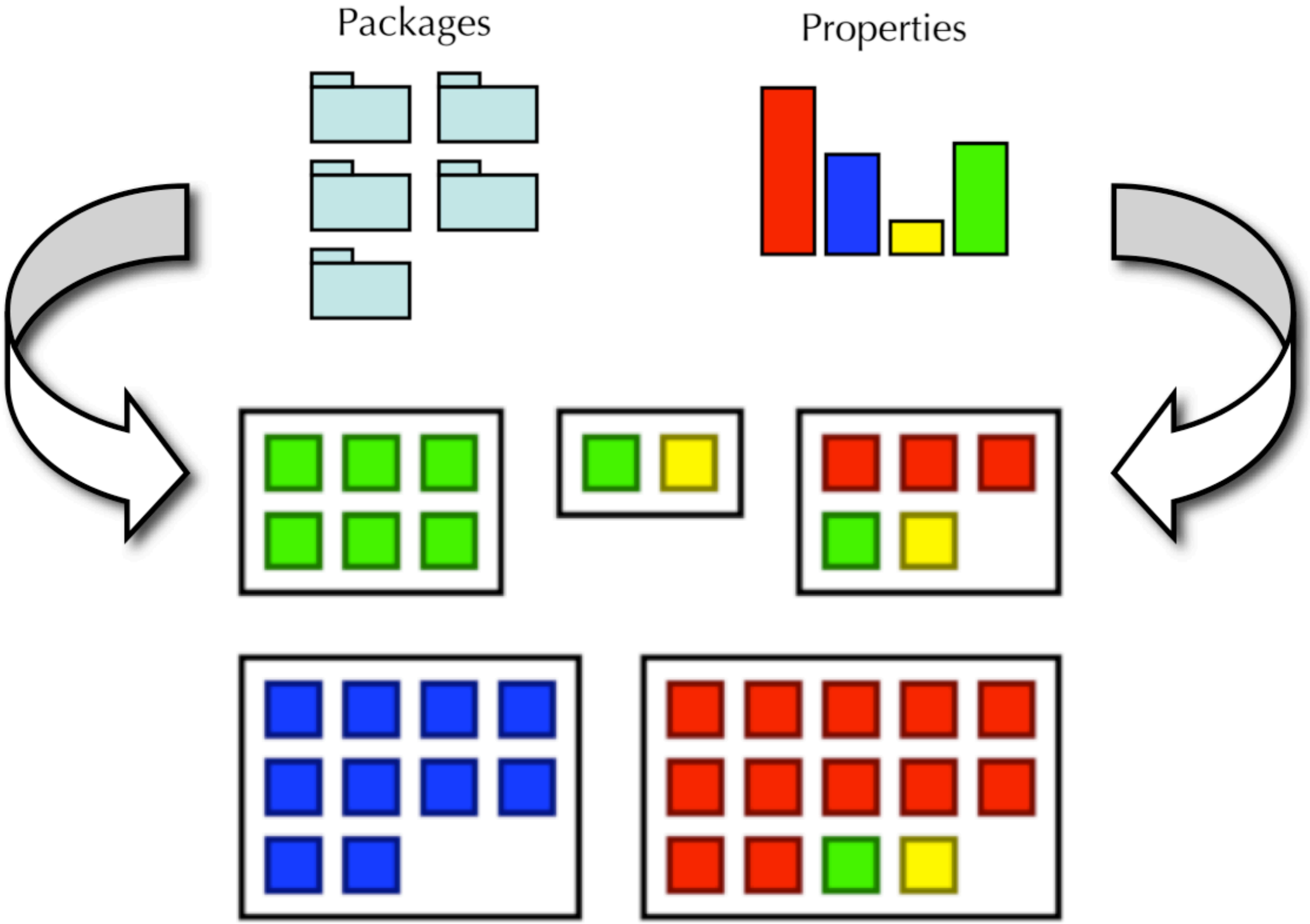
Packages



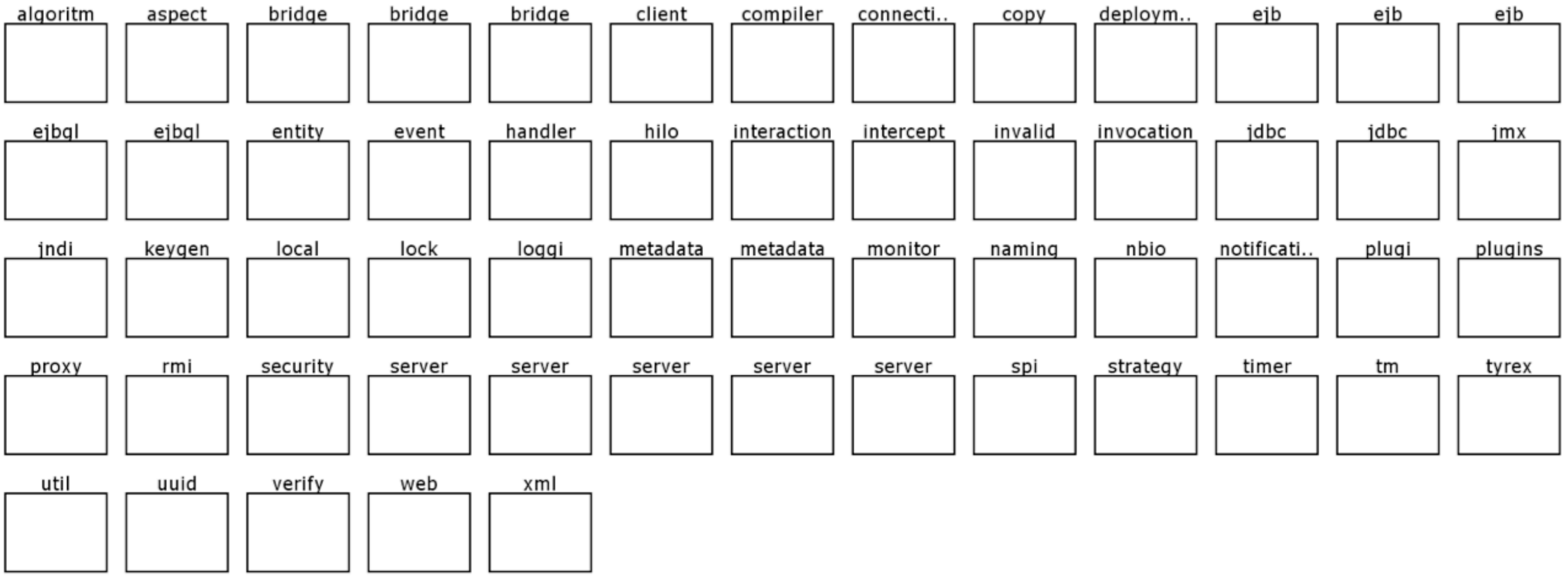
Properties



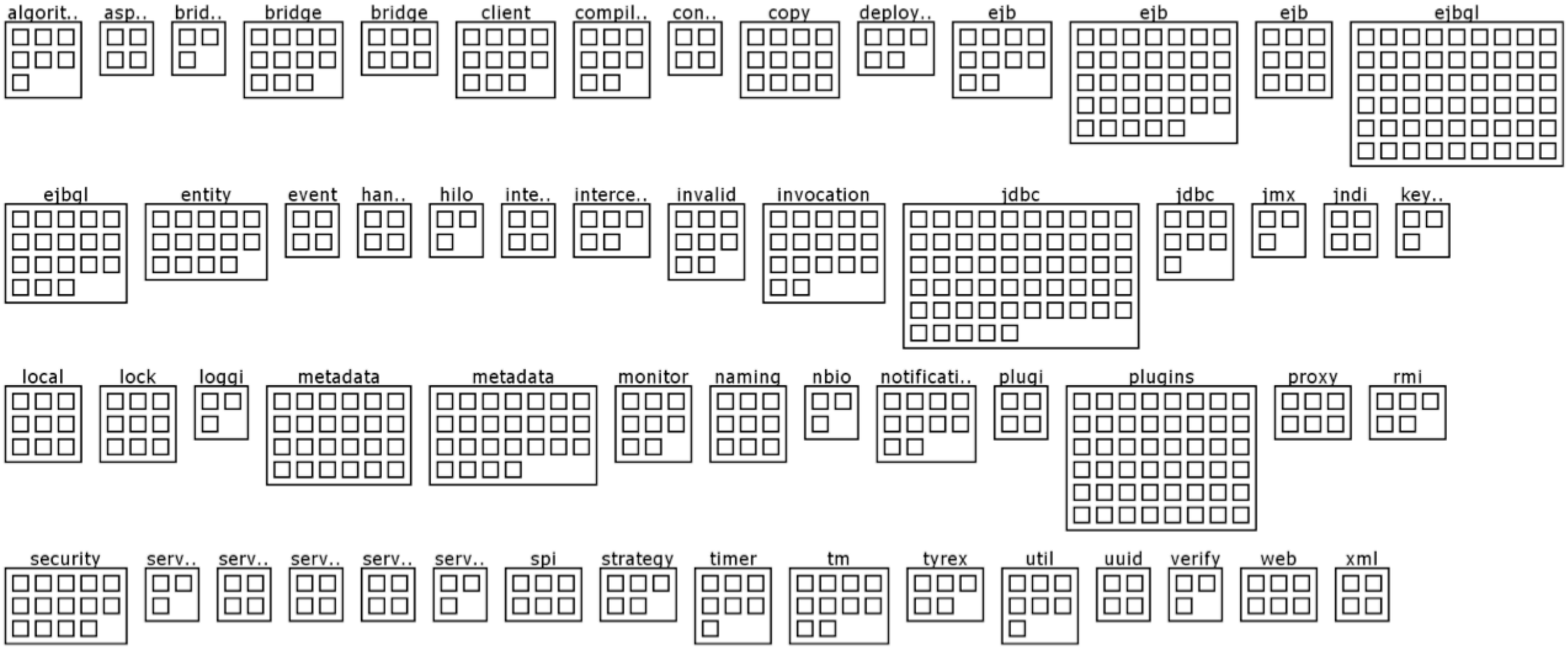
and create a Distribution Map.



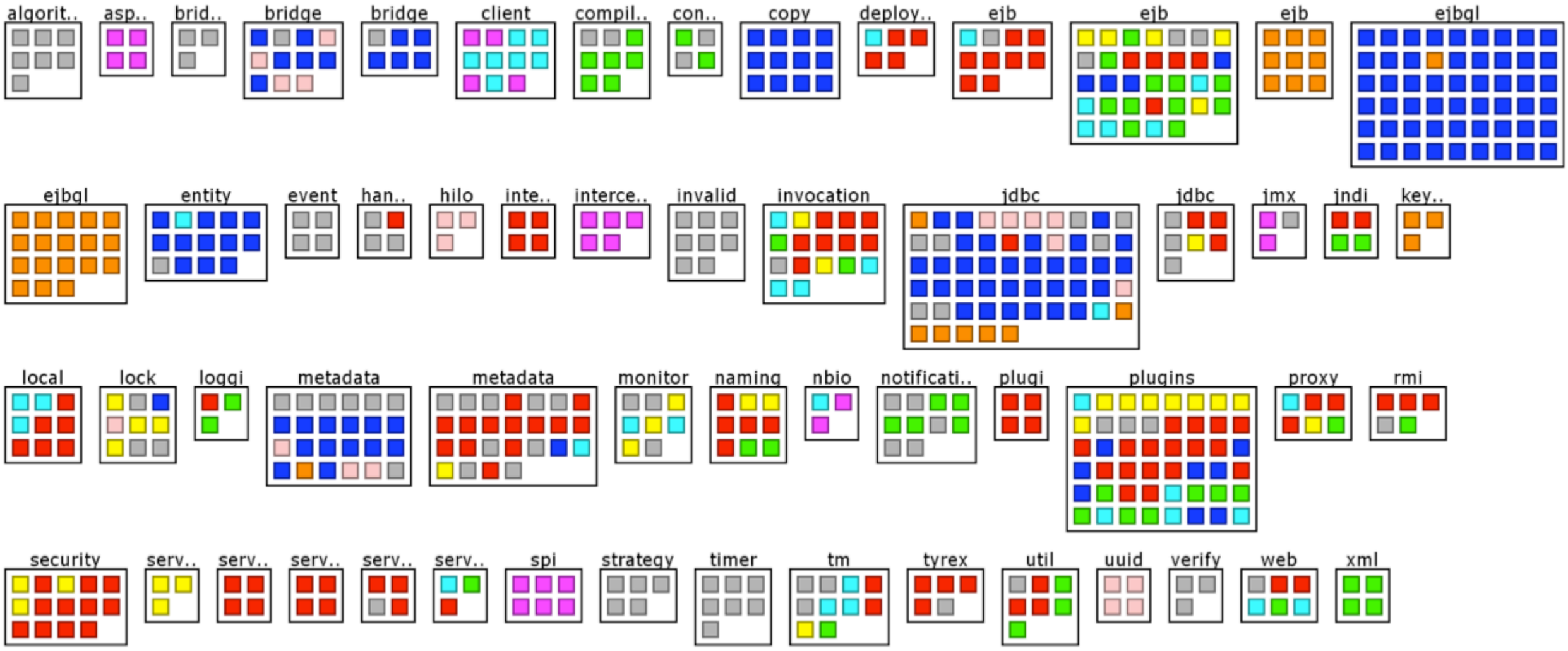
Step 1 — for each package draw a rectangle



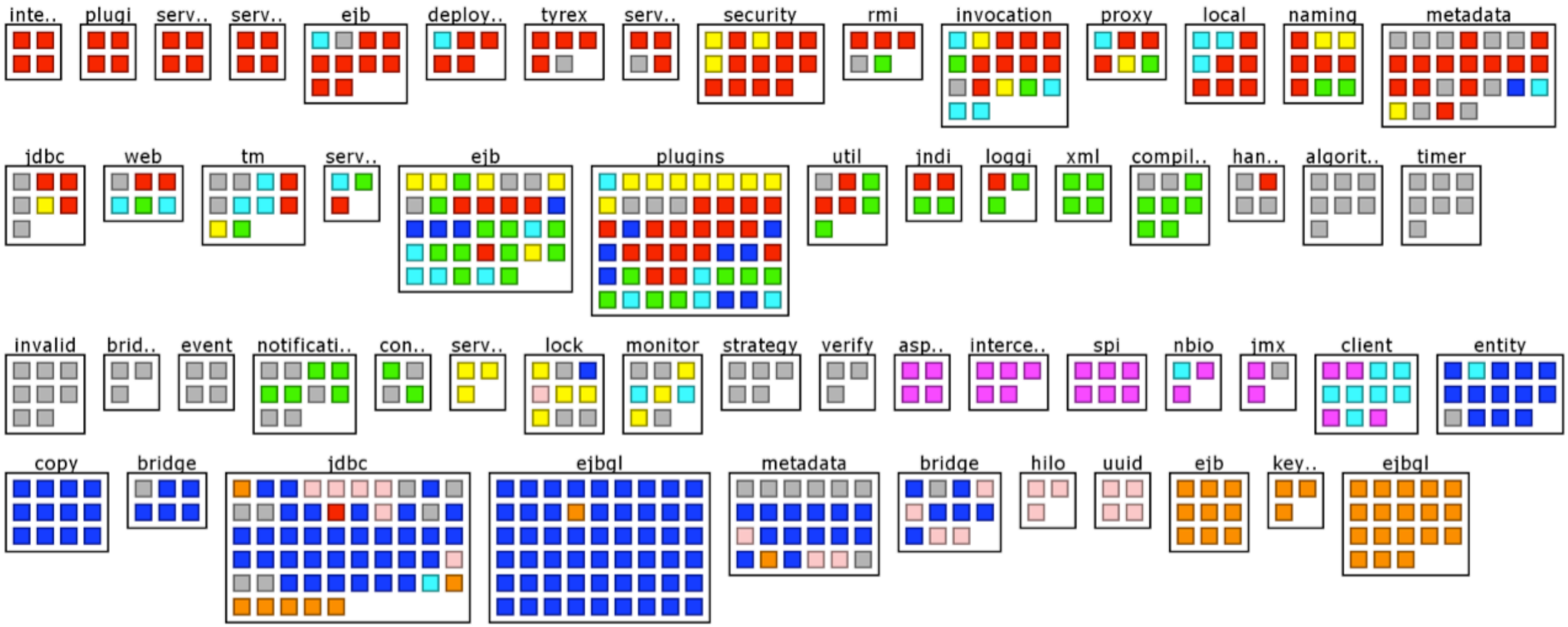
Step 2 — populate packages with classes



Step 3 — color the classes by property

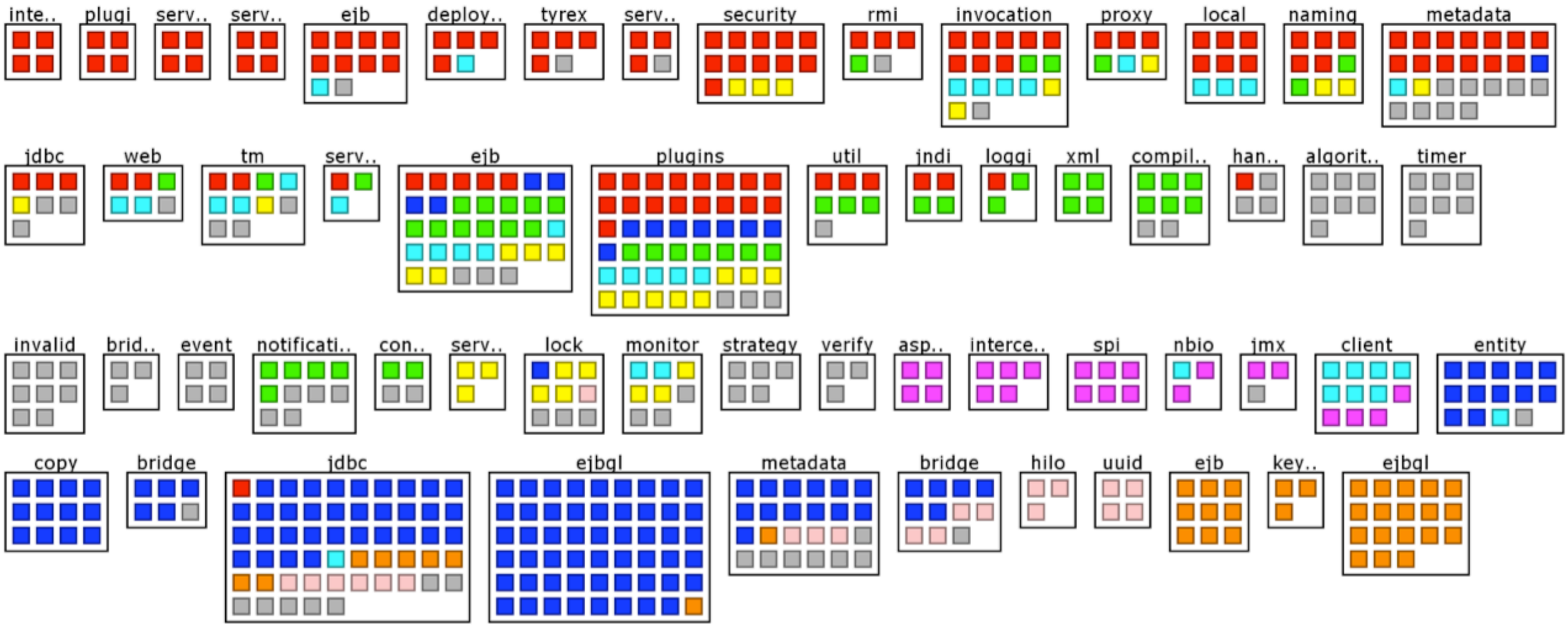


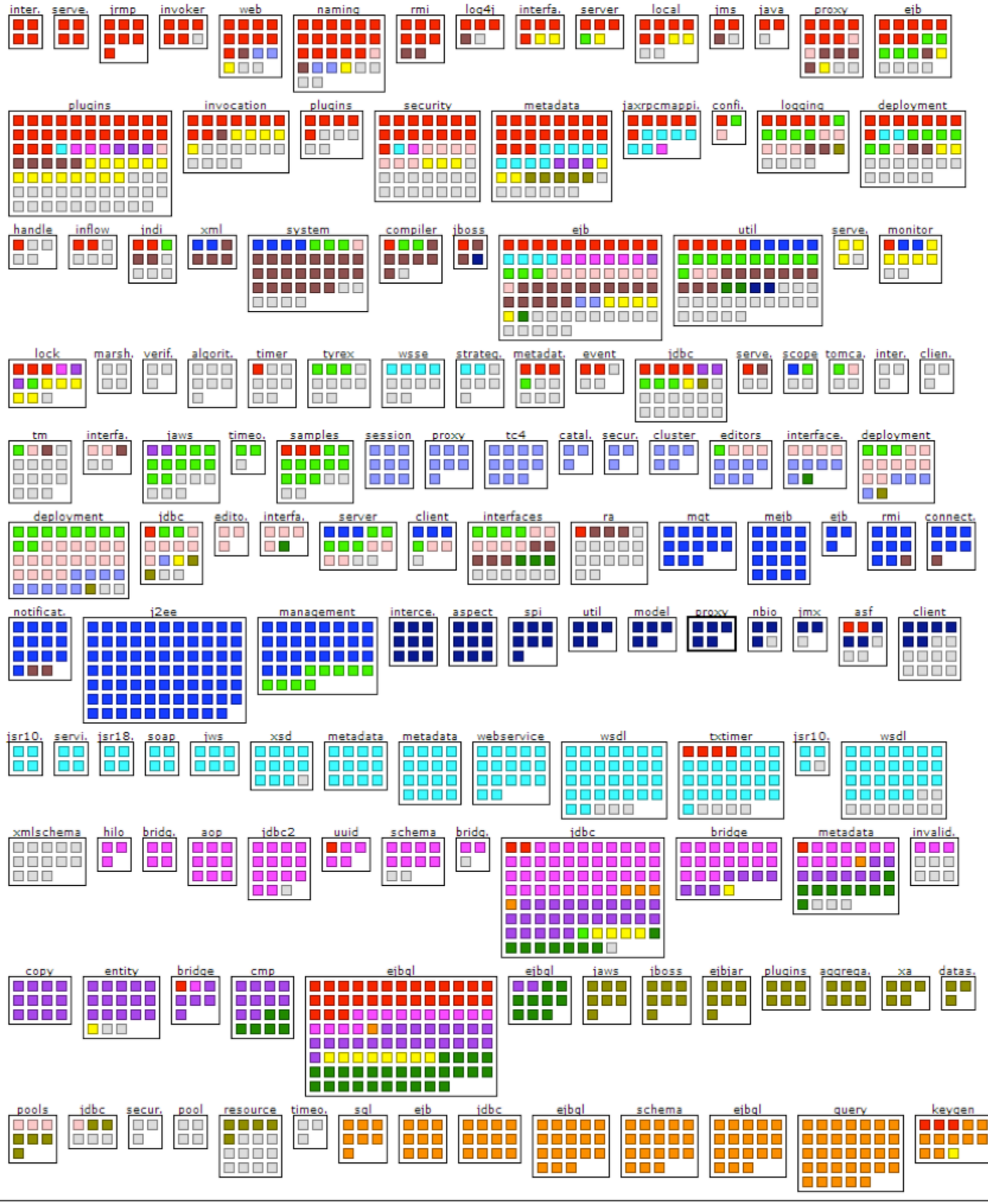
Step 4 — sort packages by content



Sorting with dendrogram seriation.

Step 5 — sort classes by properties

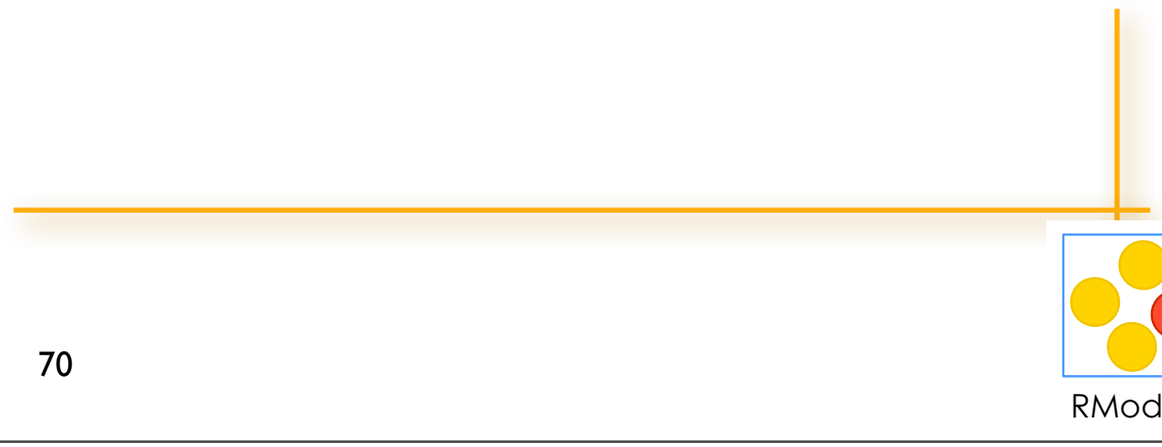



inter.
serve.
jrmp
invoker
web
naming
rmi
log4j
interfa.
server
local
jms
java
proxy
ejb
plugins
invocation
plugins
security
metadata
jaxrpcmappi.
confi.
logging
deployment
handle
inflow
jndi
xml
system
compiler
jboss
ejb
util
serve.
monitor
lock
marsh.
verif.
algorit.
timer
tyrex
wsse
strateg.
metadat.
event
jdbc
serve.
scope
tomca.
inter.
clien.
tm
interfa.
jaws
timeo.
samples
session
proxy
tc4
catal.
secur.
cluster
editors
interface.
deployment
deployment
jdbc
edito.
interfa.
server
client
interfaces
ra
mgt
mejb
ejb
rmi
connect.
notificat.
j2ee
management
interce.
aspect
spi
util
model
proxy
nbio
jmx
asf
client
jsr10.
servi.
jsr18.
soap
jws
xsd
metadata
metadata
webservice
wsdl
txtimer
jsr10.
wsdl
xmlschema
hilo
bridg.
aop
jdbc2
uuid
schema
bridg.
jdbc
bridge
metadata
invalid.
copy
entity
bridge
cmp
ejbql
ejbql
jaws
jboss
ejbjar
plugins
aggrega.
xa
datas.
pools
jdbc
secur.
pool
resource
timeo.
sql
ejb
jdbc
ejbql
schema
ejbql
query
keygen

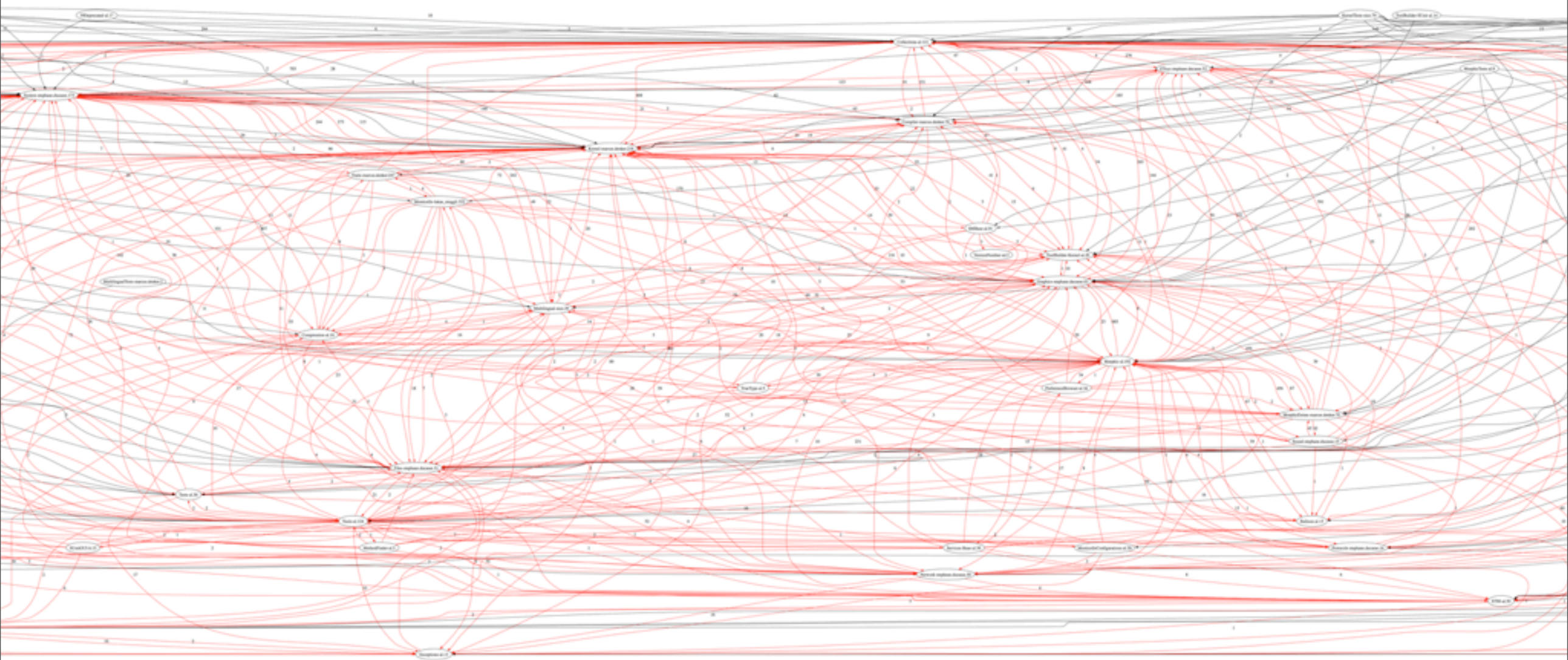
Challenges

- How to modularize a system?
- Where are the cycles?
- What produce cycles?
- Where are the layers

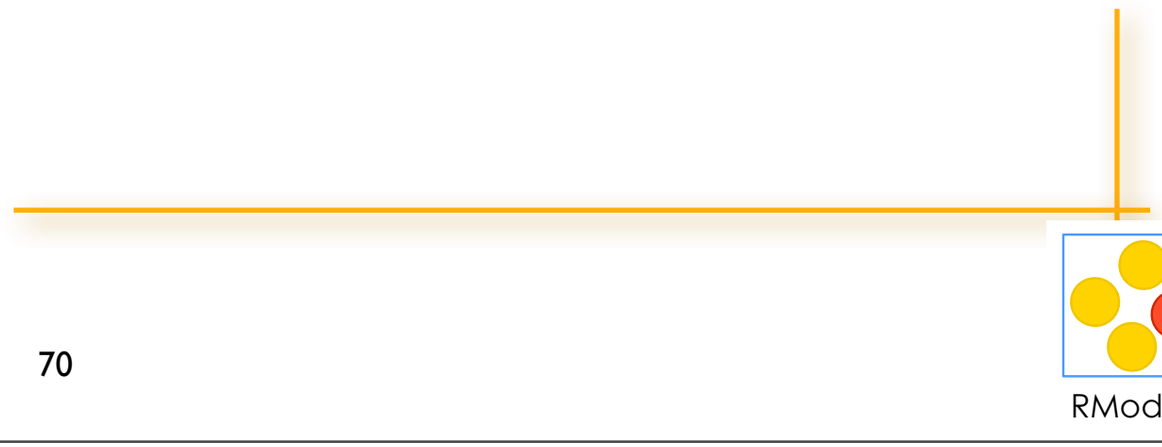
Graph you said?



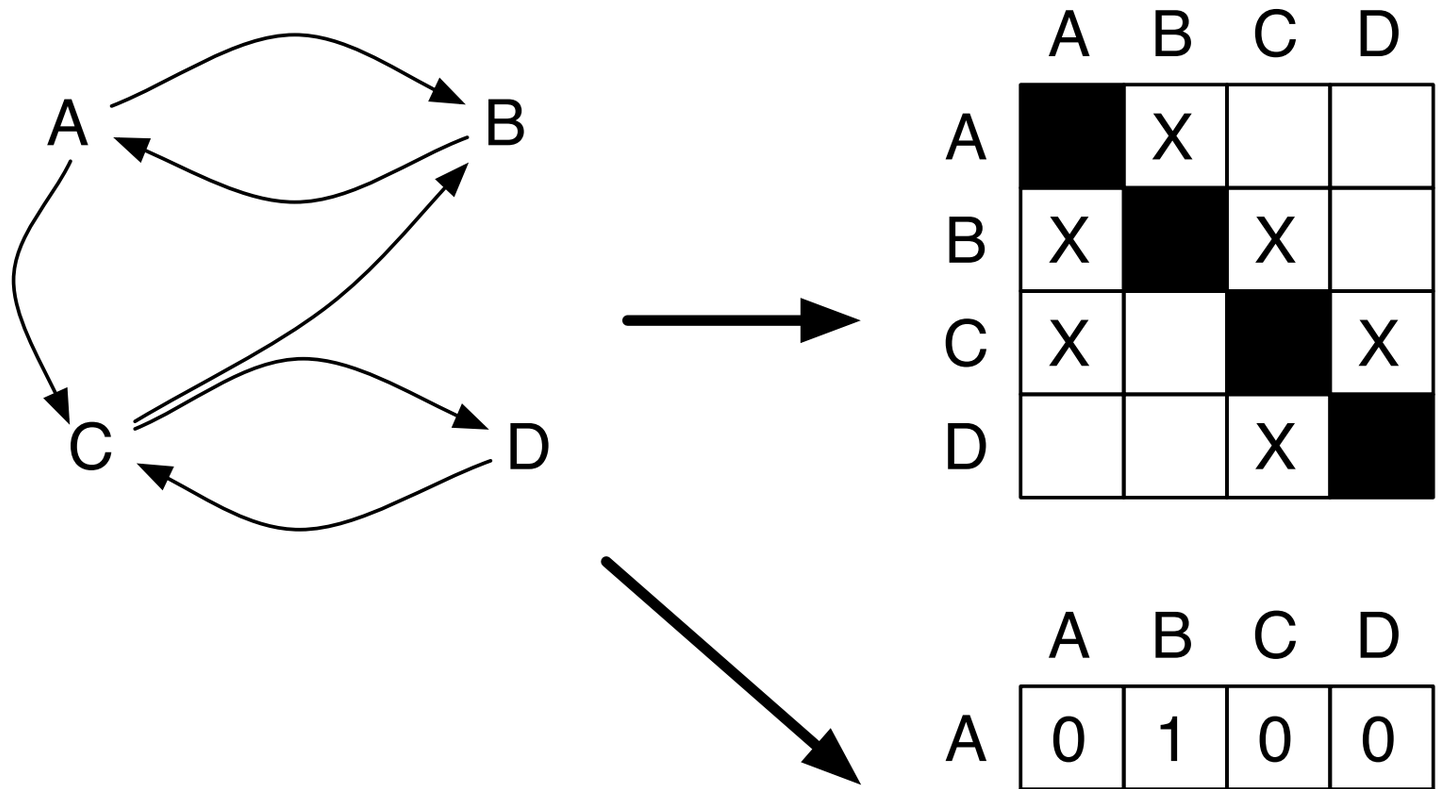
Graph you said?



Graph you said?



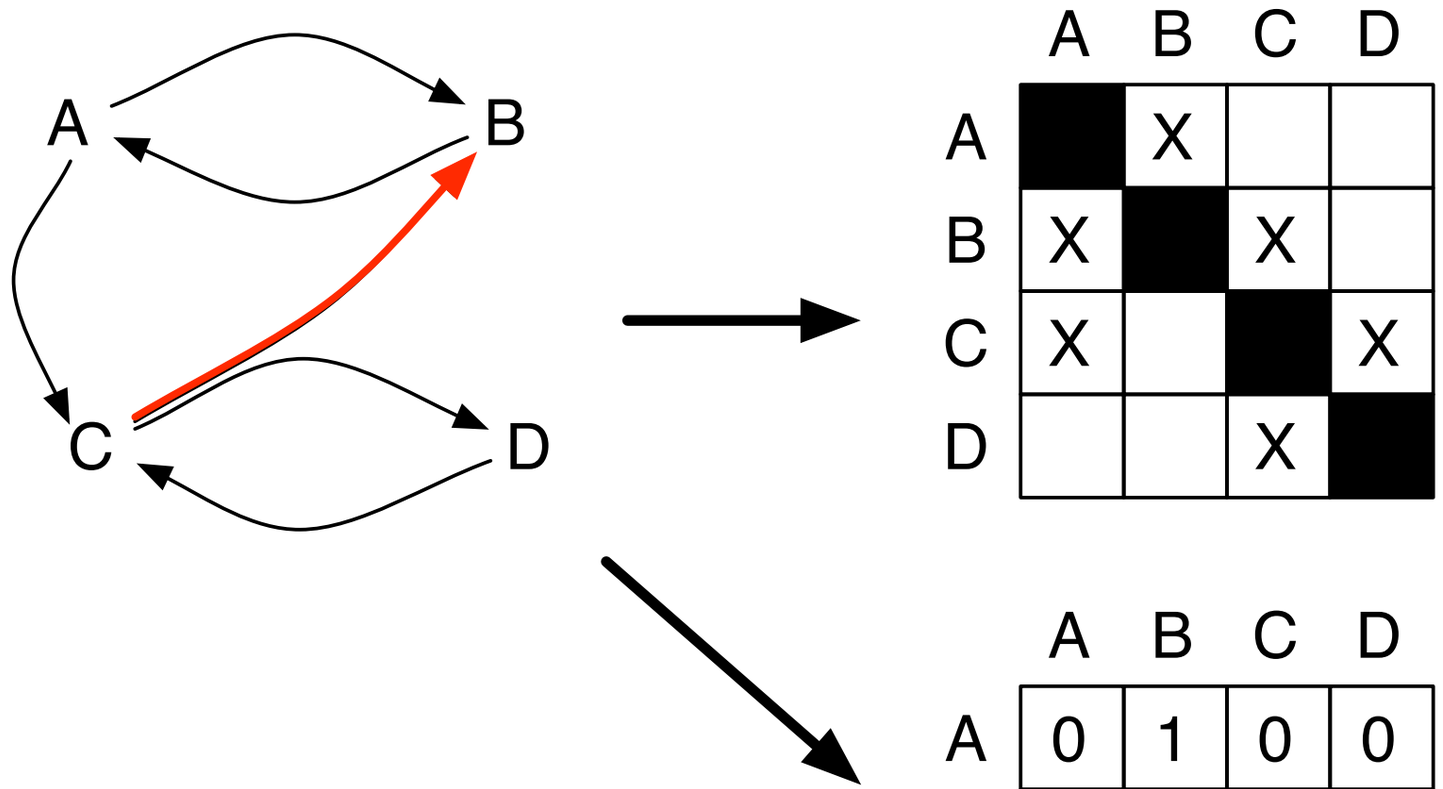
Building a DSM



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

	A	B	C	D
A	0	1	0	0
B	1	0	1	0
C	1	0	0	1
D	0	0	1	0

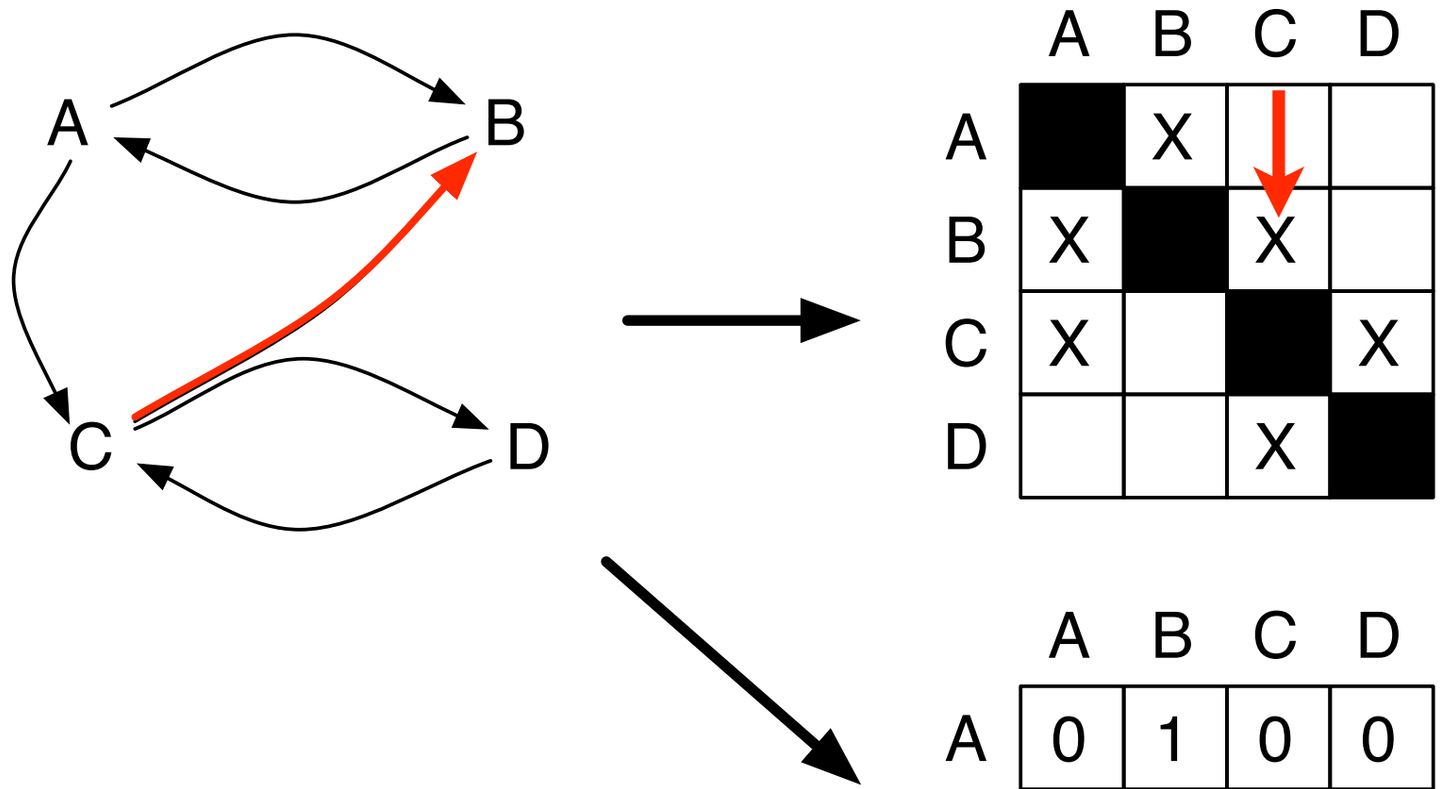
Building a DSM



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

	A	B	C	D
A	0	1	0	0
B	1	0	1	0
C	1	0	0	1
D	0	0	1	0

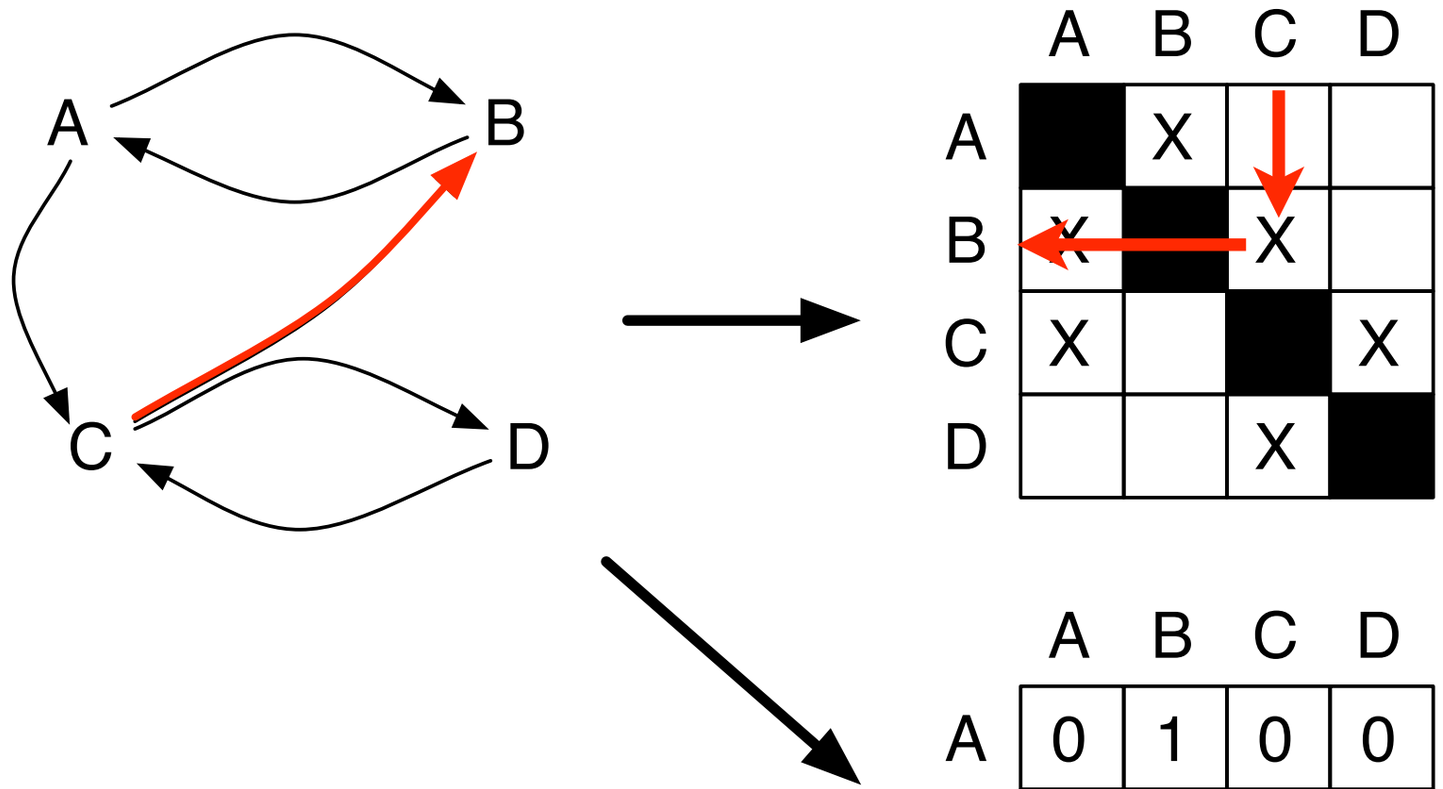
Building a DSM



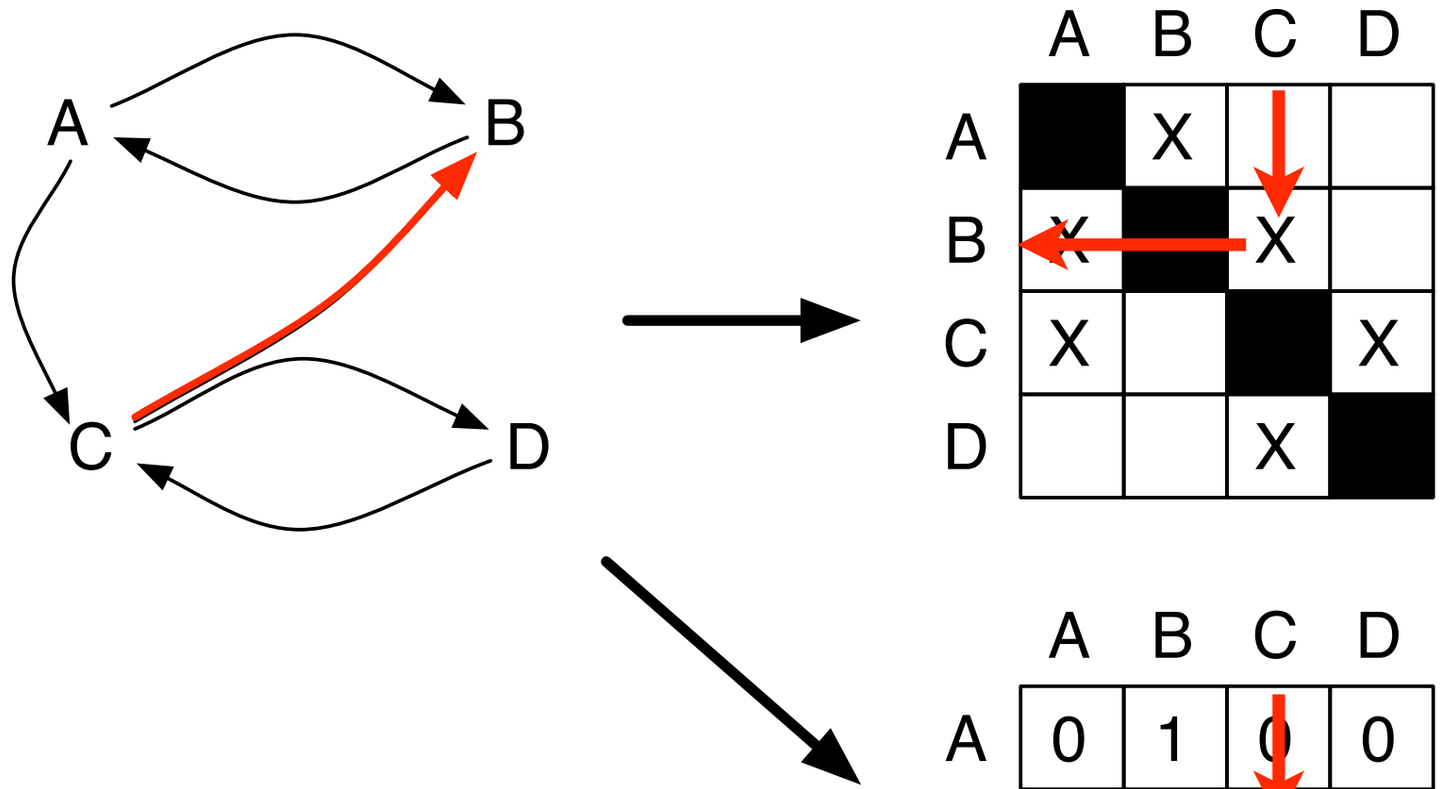
	A	B	C	D
A		X		
B	X		X	
C	X			X
D			X	

	A	B	C	D
A	0	1	0	0
B	1	0	1	0
C	1	0	0	1
D	0	0	1	0

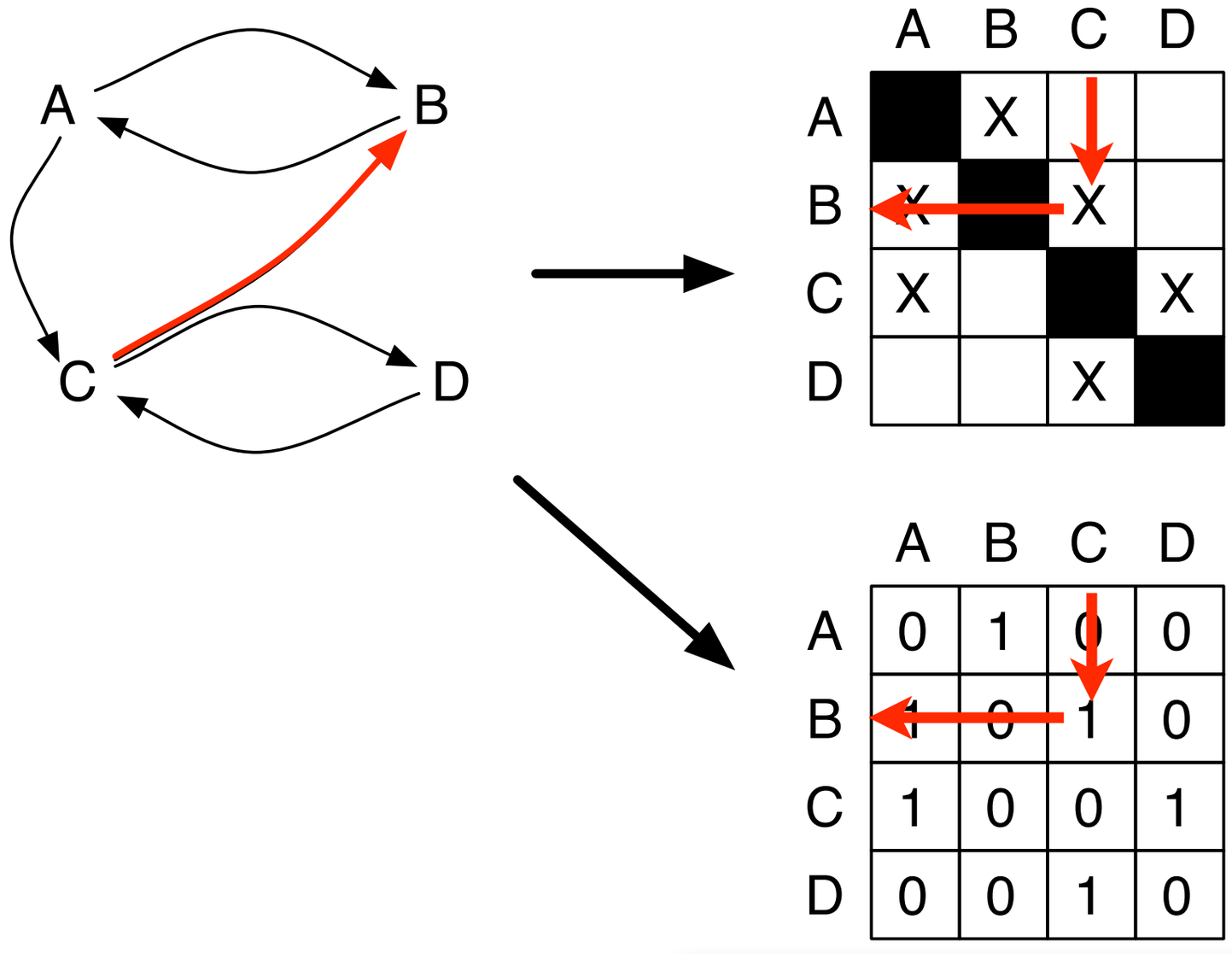
Building a DSM



Building a DSM



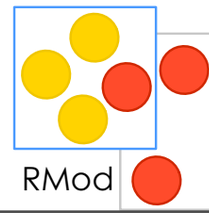
Building a DSM



7 Packages visualization

- 1 cell = 1 dependency
- 1 column = used packages
- 1 line = using packages

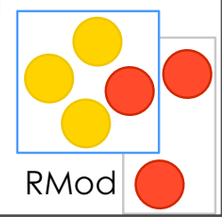
	x	x	x	x	x	x	x	x	x	x
x										
x										
x				71	3					
x	2	1	8		7	6				
x						3				
x	4	51		2	2		2			
x	4			10	4	34		3		
x		15					1			
x		30								
x		2		2		6				



7 Packages visualization

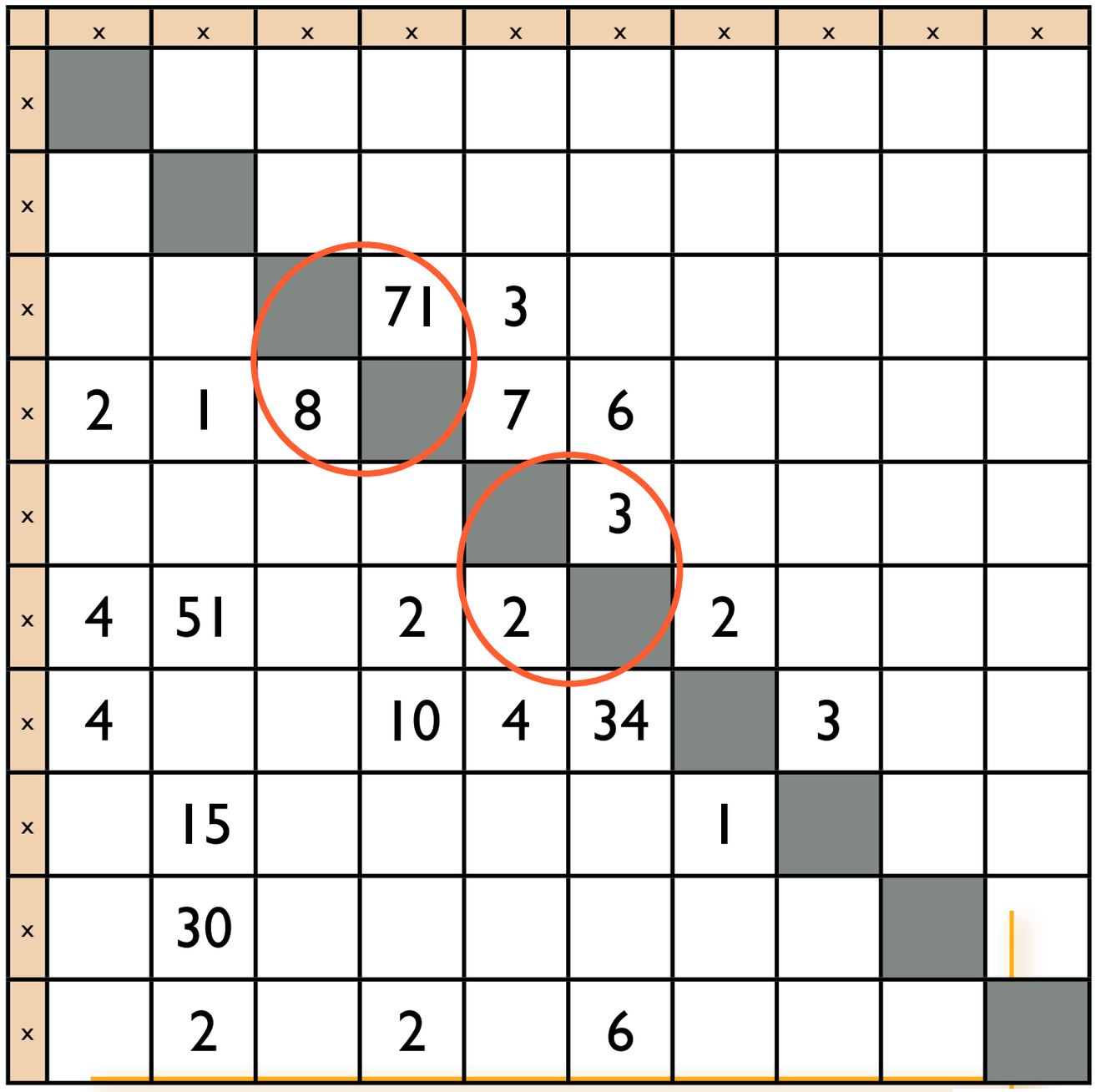
- 1 cell = 1 dependency
- 1 column = used packages
- 1 line = using packages

	x	x	x	x	x	x	x	x	x	x
x										
x										
x				71	3					
x	2	1	8		7	6				
x						3				
x	4	51		2	2		2			
x	4			10	4	34		3		
x		15					1			
x		30								
x		2		2		6				



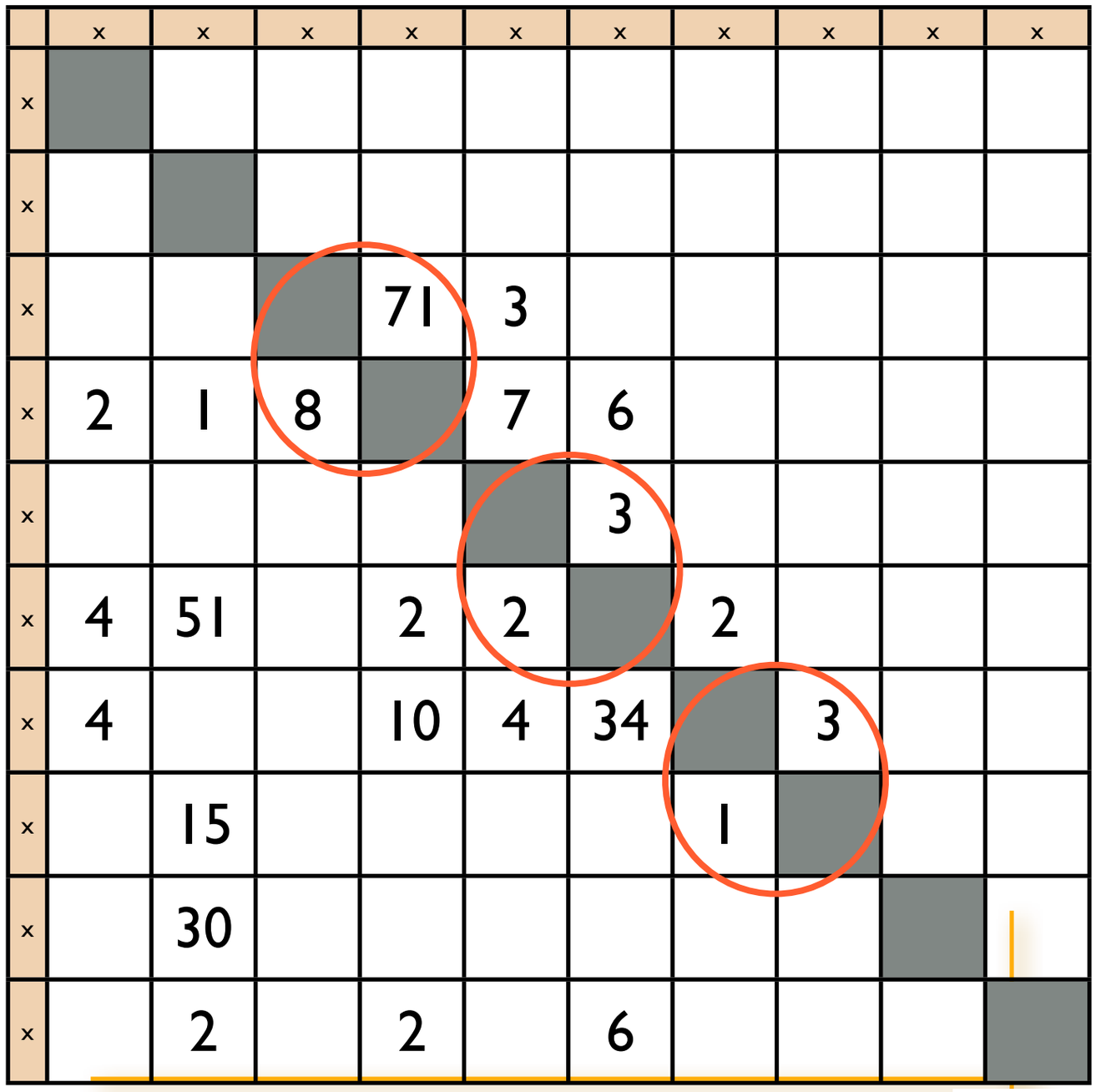
7 Packages visualization

- 1 cell = 1 dependency
- 1 column = used packages
- 1 line = using packages



7 Packages visualization

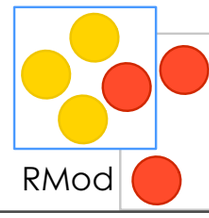
- 1 cell = 1 dependency
- 1 column = used packages
- 1 line = using packages



7 Packages visualization

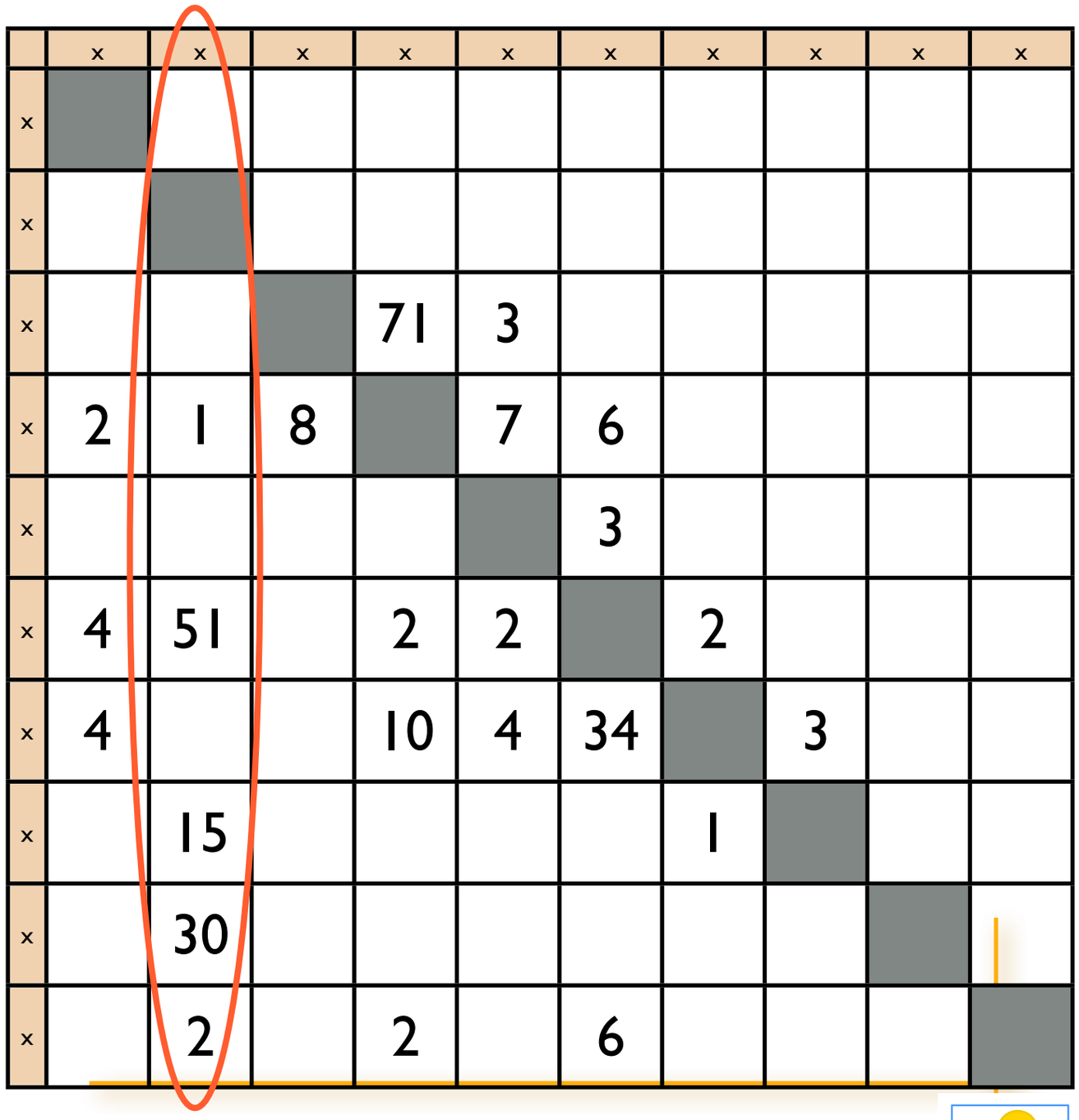
- 1 cell = 1 dependency
- 1 column = used packages
- 1 line = using packages

	x	x	x	x	x	x	x	x	x	x
x										
x										
x				71	3					
x	2	1	8		7	6				
x						3				
x	4	51		2	2		2			
x	4			10	4	34		3		
x		15					1			
x		30								
x		2		2		6				



7 Packages visualization

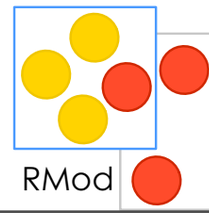
- 1 cell = 1 dependency
- 1 column = used packages
- 1 line = using packages



7 Packages visualization

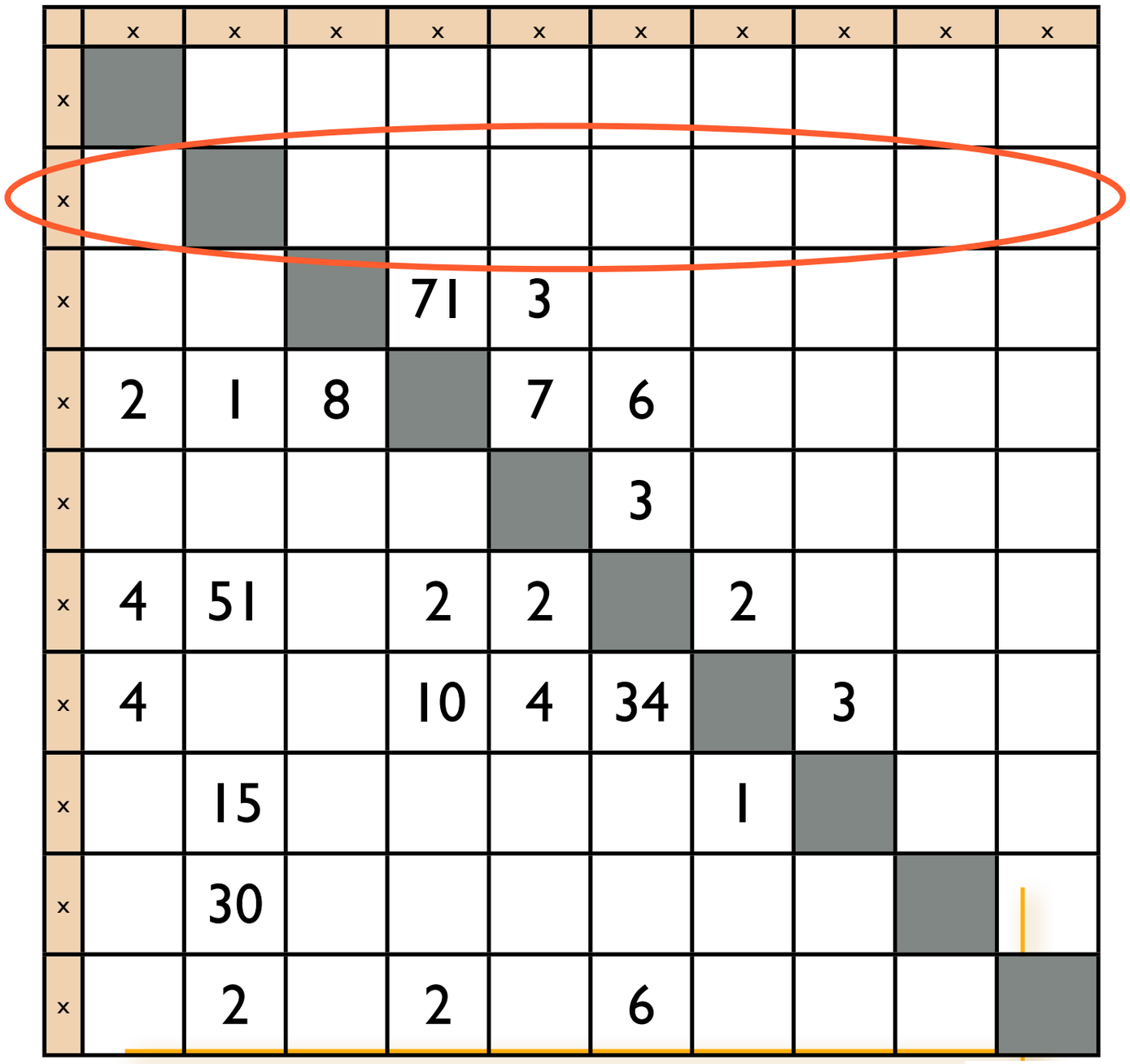
- 1 cell = 1 dependency
- 1 column = used packages
- 1 line = using packages

	x	x	x	x	x	x	x	x	x	x
x										
x										
x				71	3					
x	2	1	8		7	6				
x						3				
x	4	51		2	2		2			
x	4			10	4	34		3		
x		15					1			
x		30								
x		2		2		6				

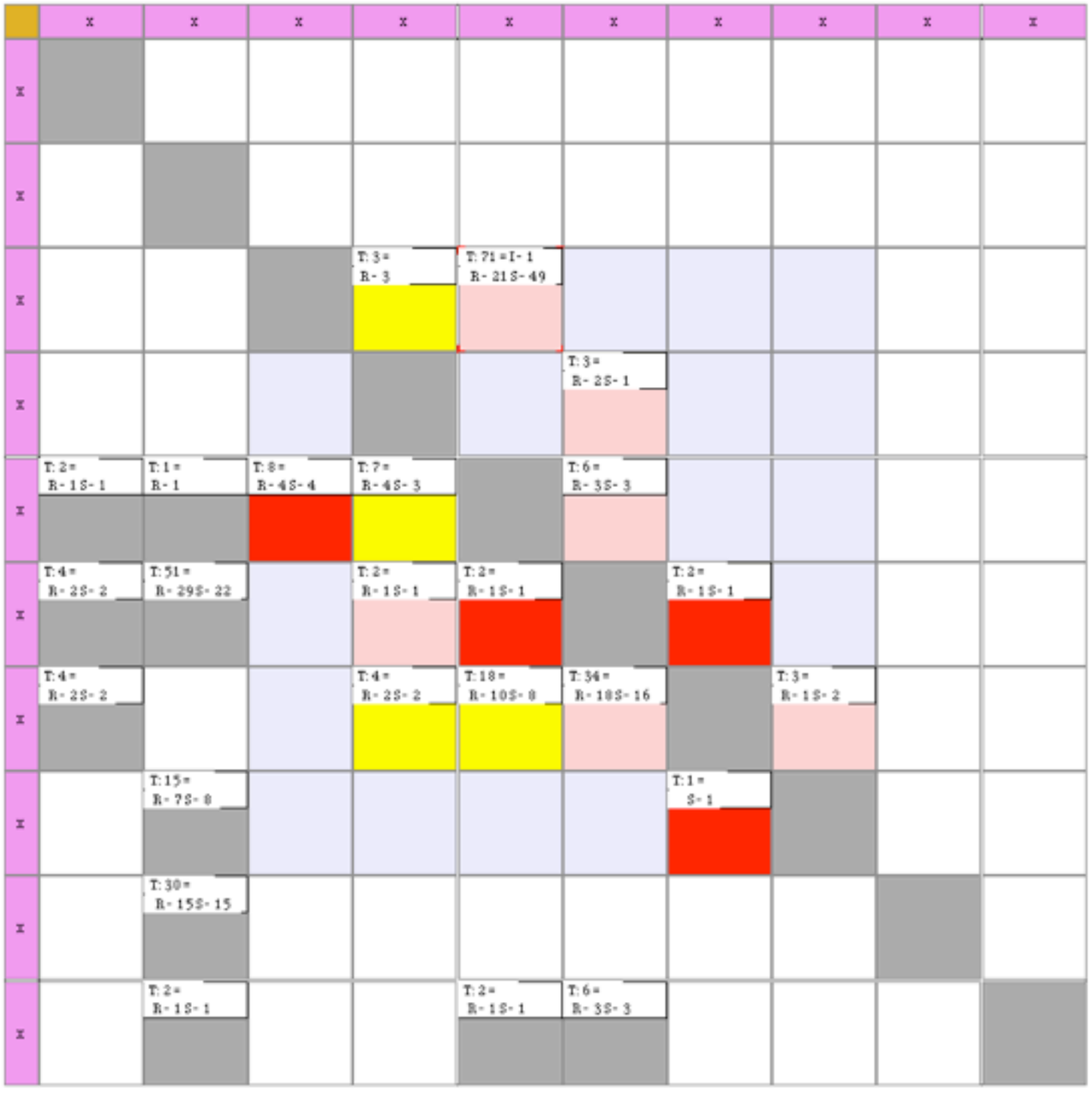


7 Packages visualization

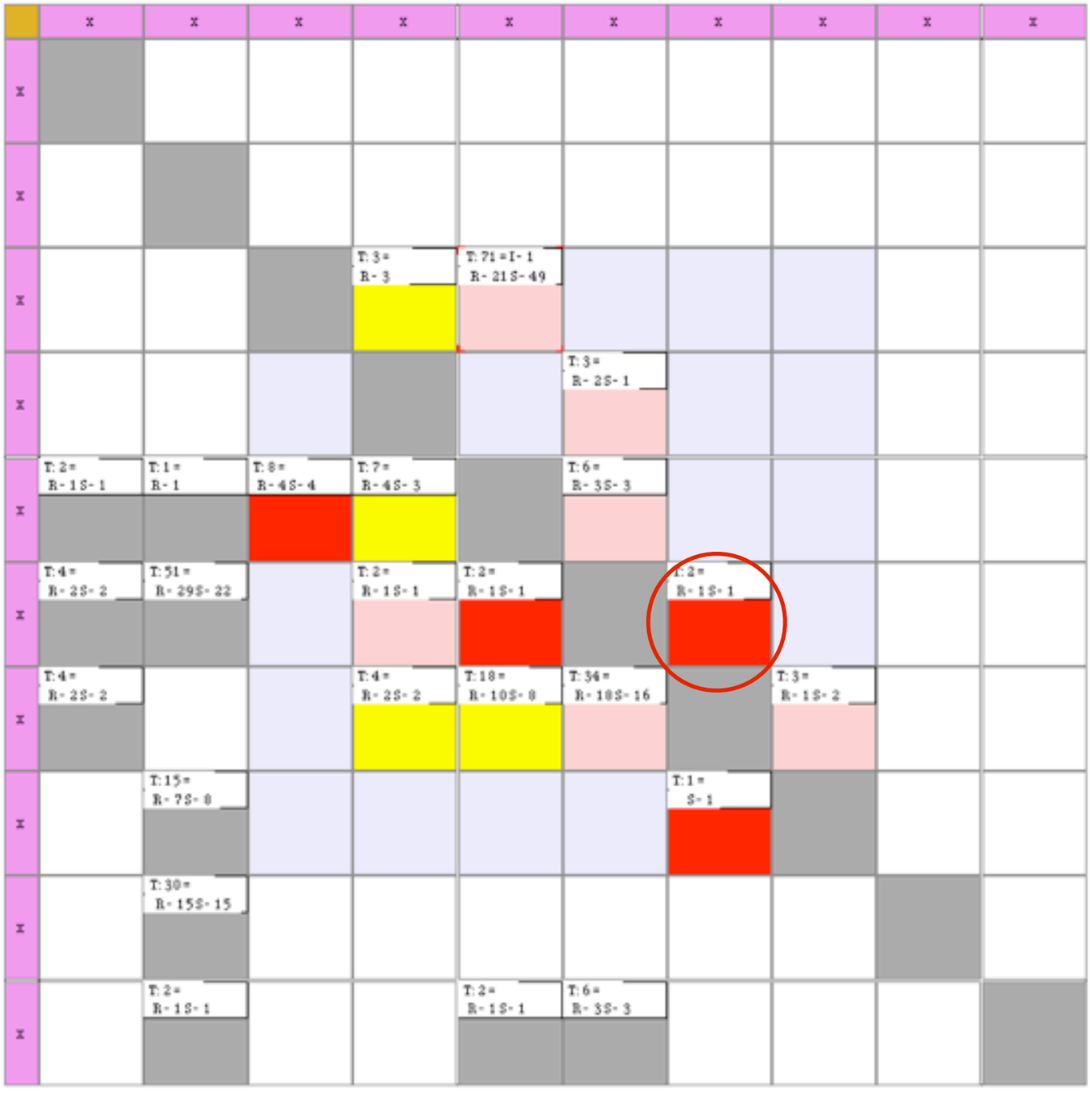
- 1 cell = 1 dependency
- 1 column = used packages
- 1 line = using packages



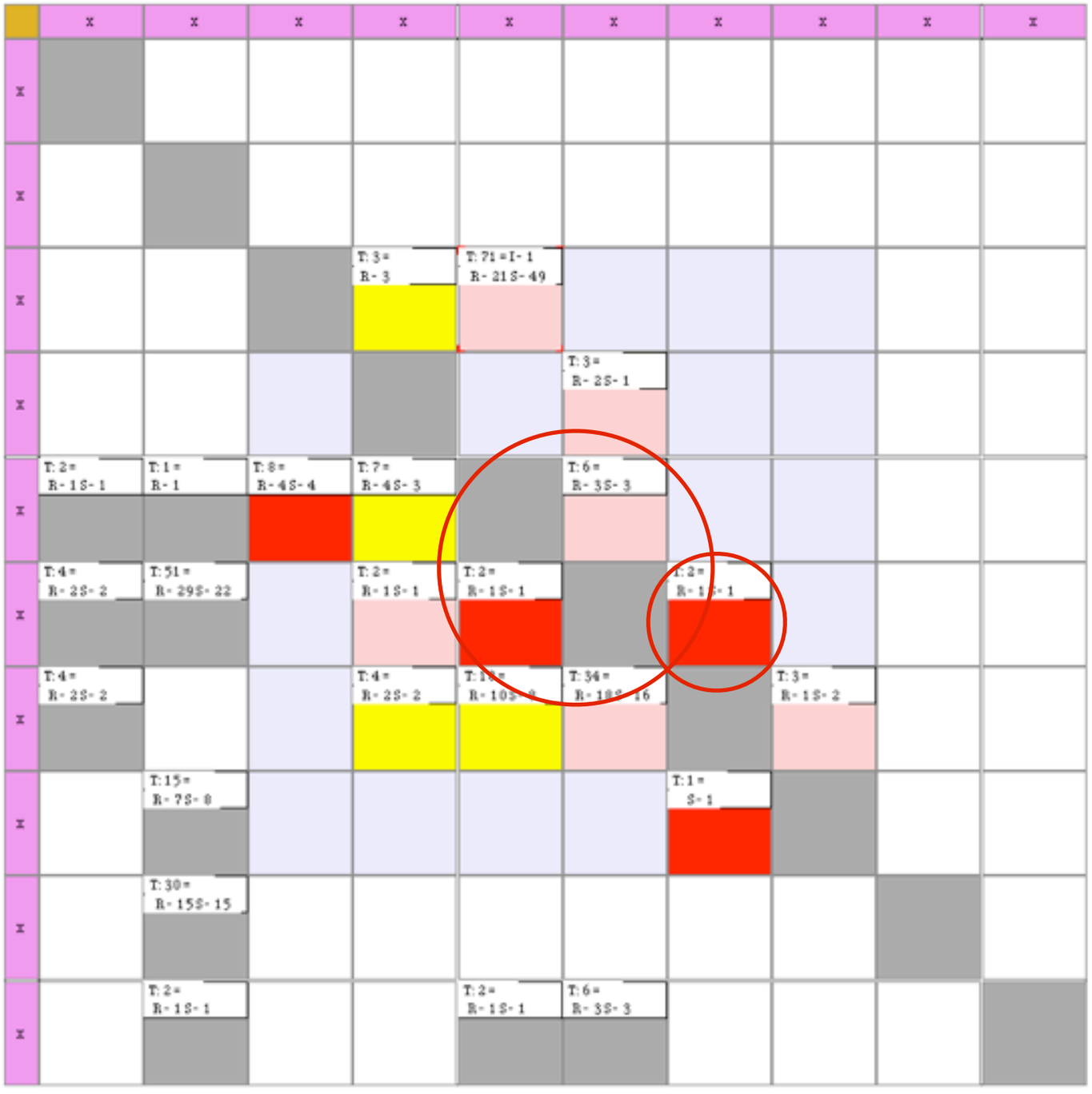
Identify cycles



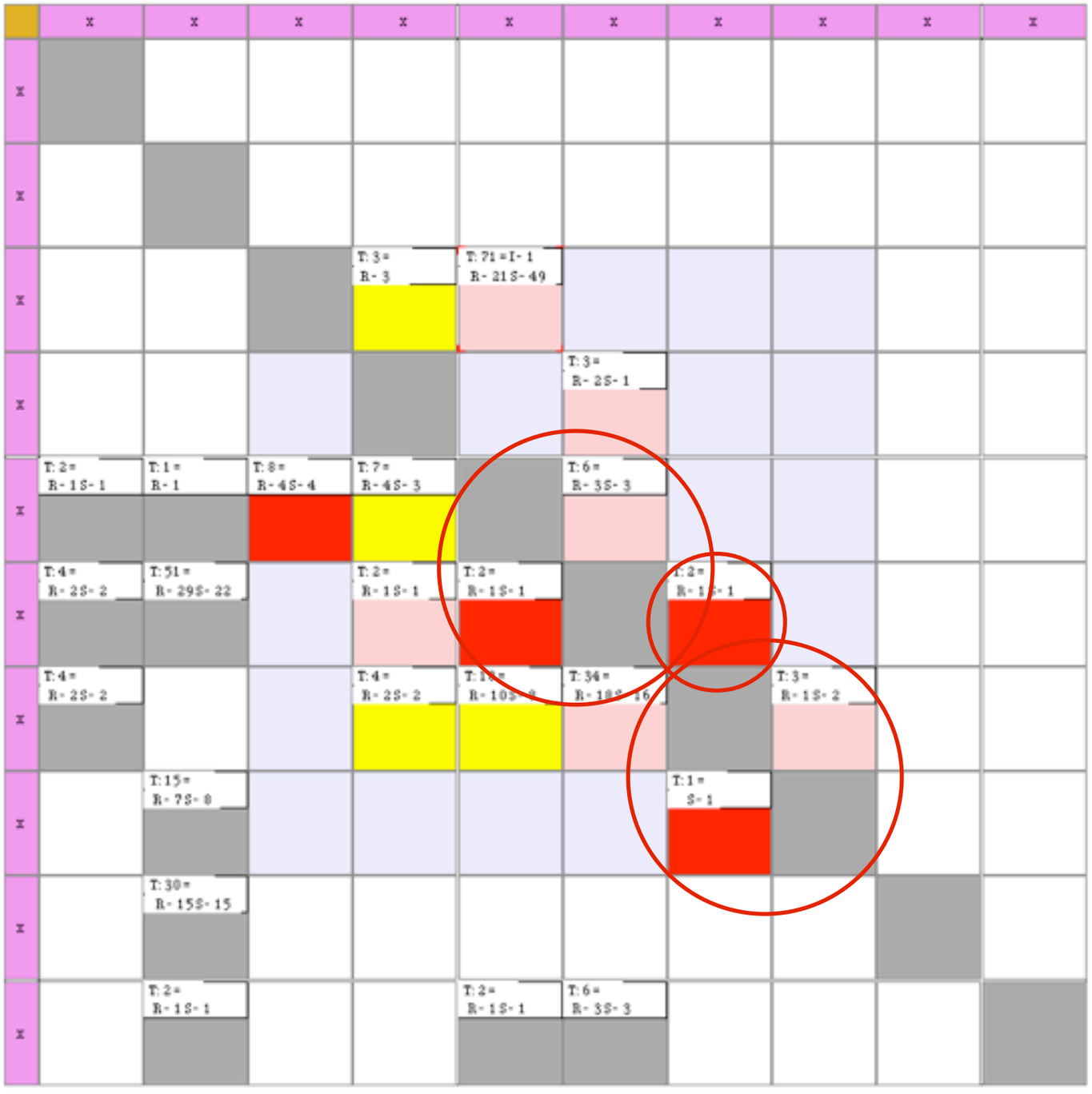
Identify cycles



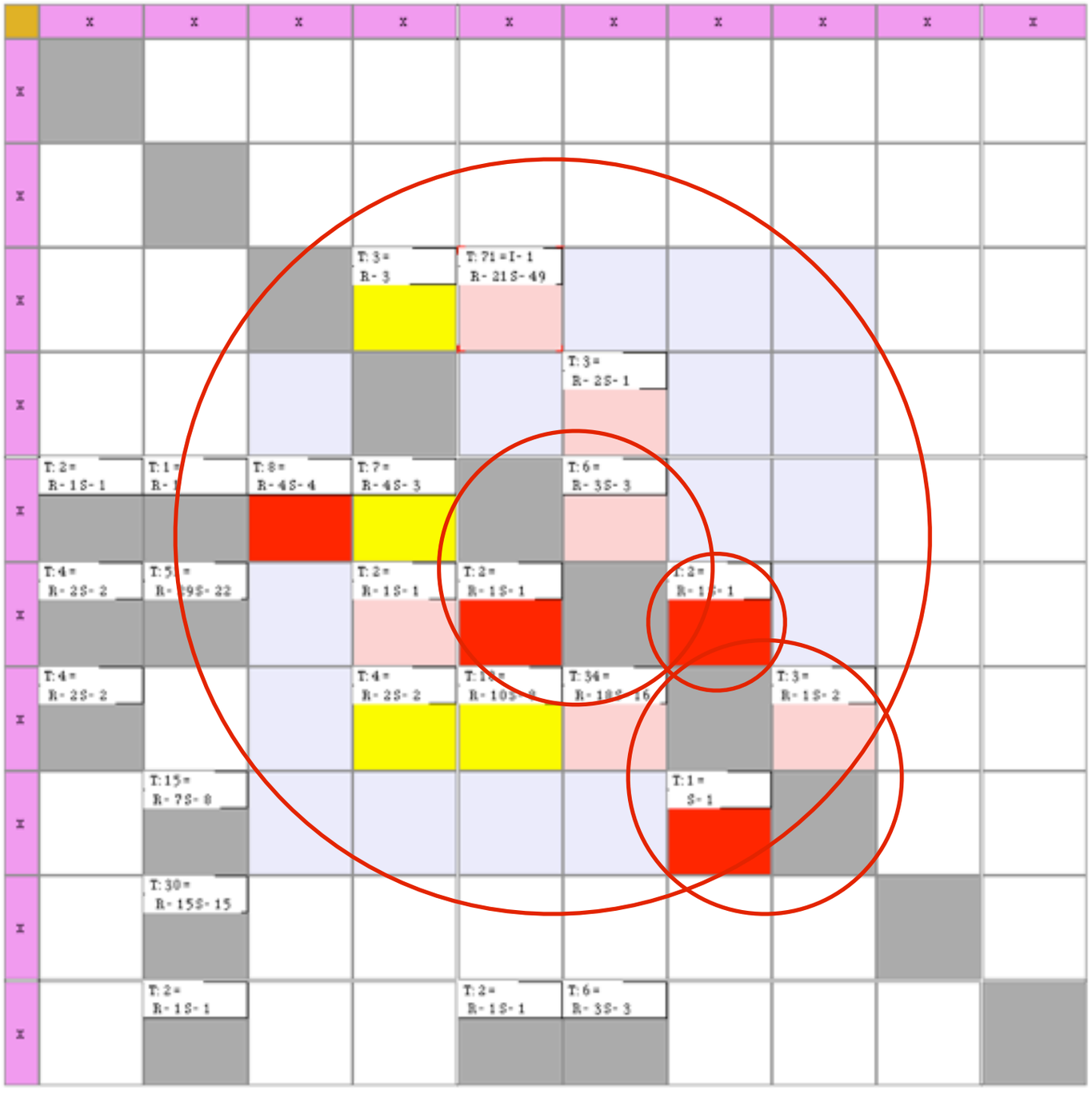
Identify cycles



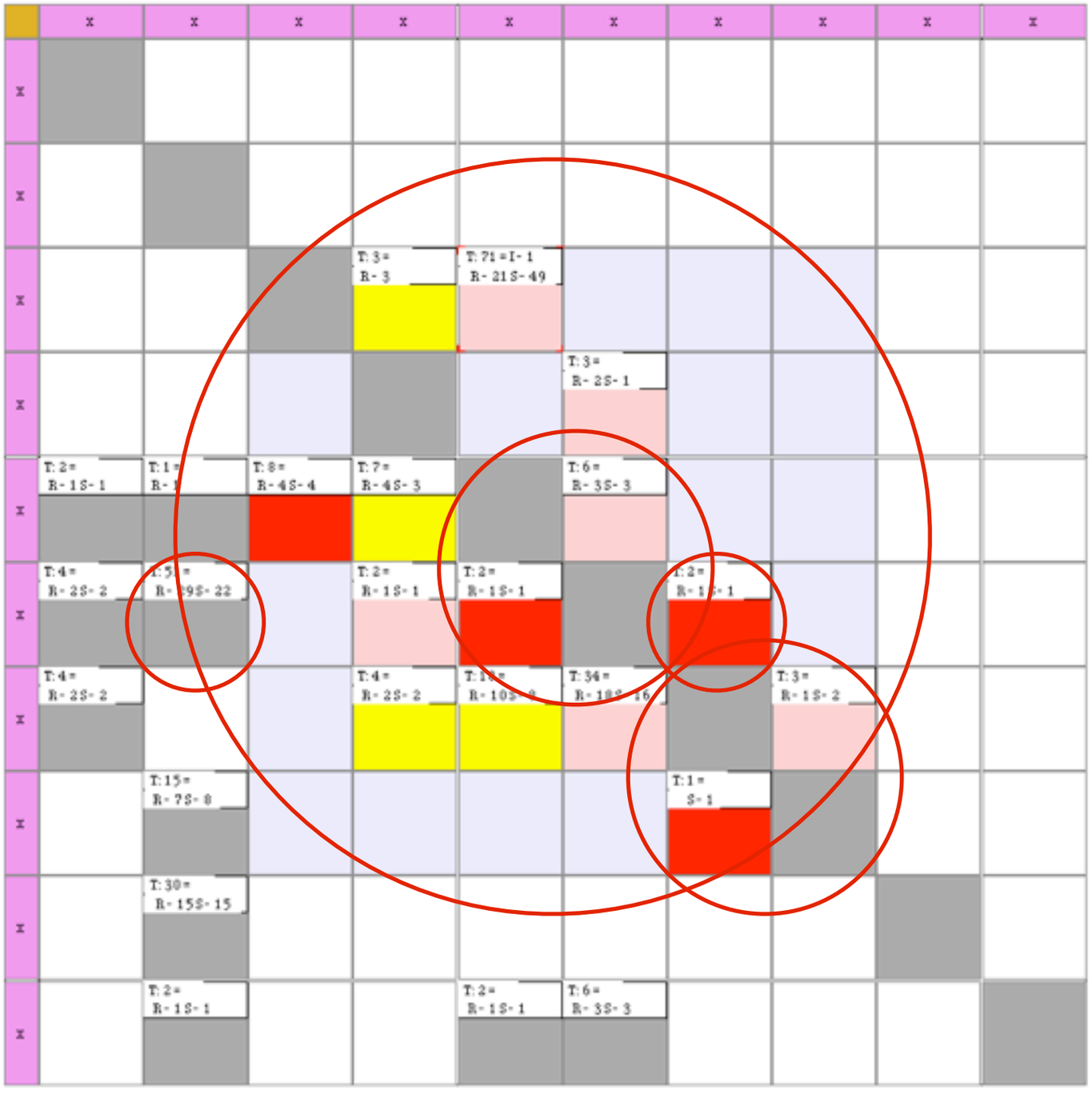
Identify cycles



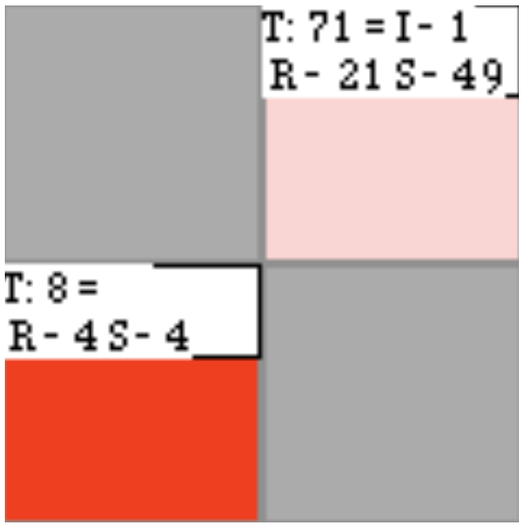
Identify cycles



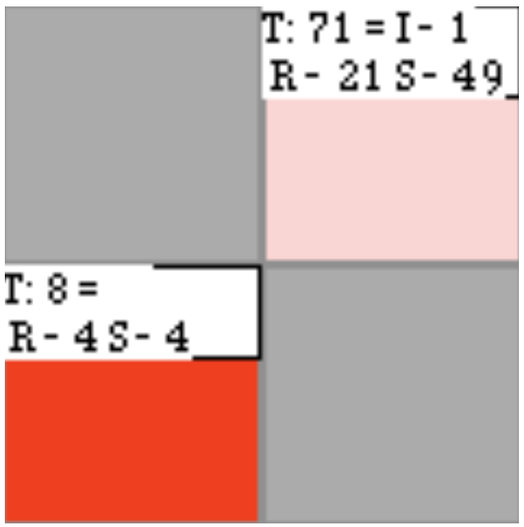
Identify cycles

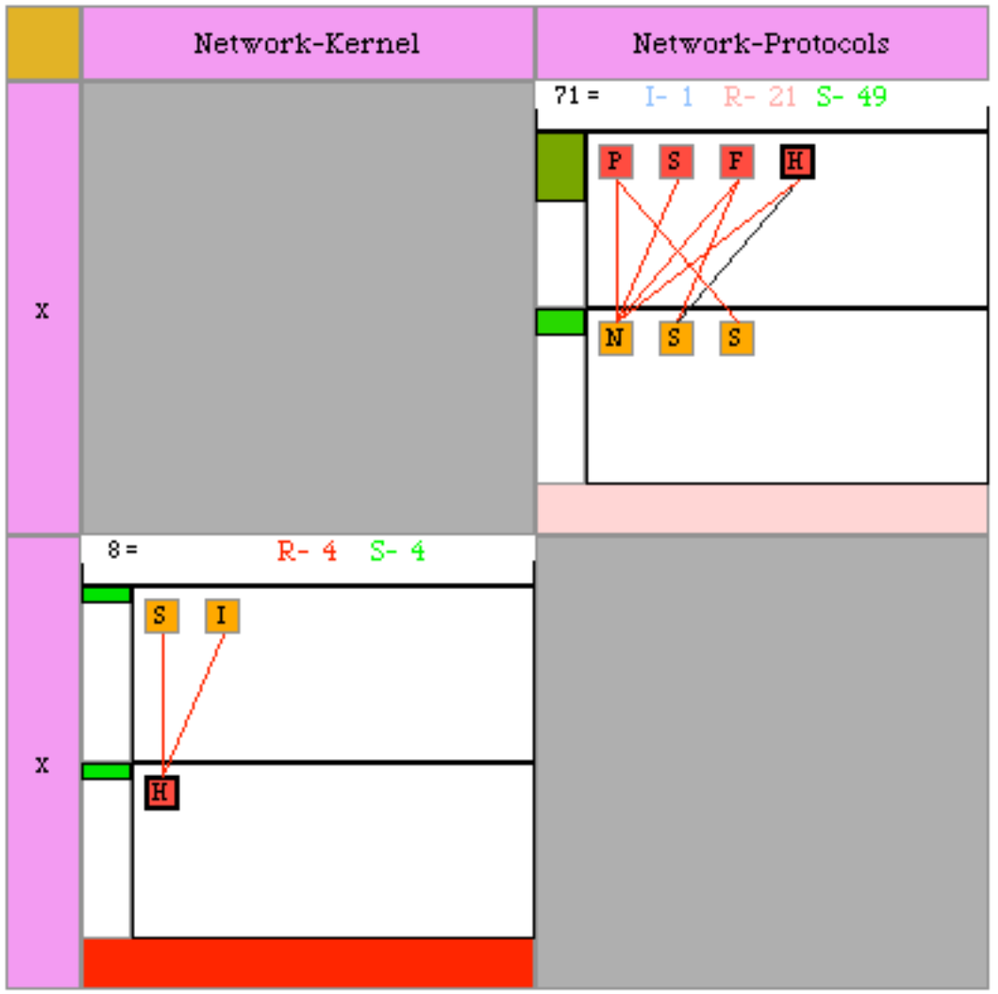


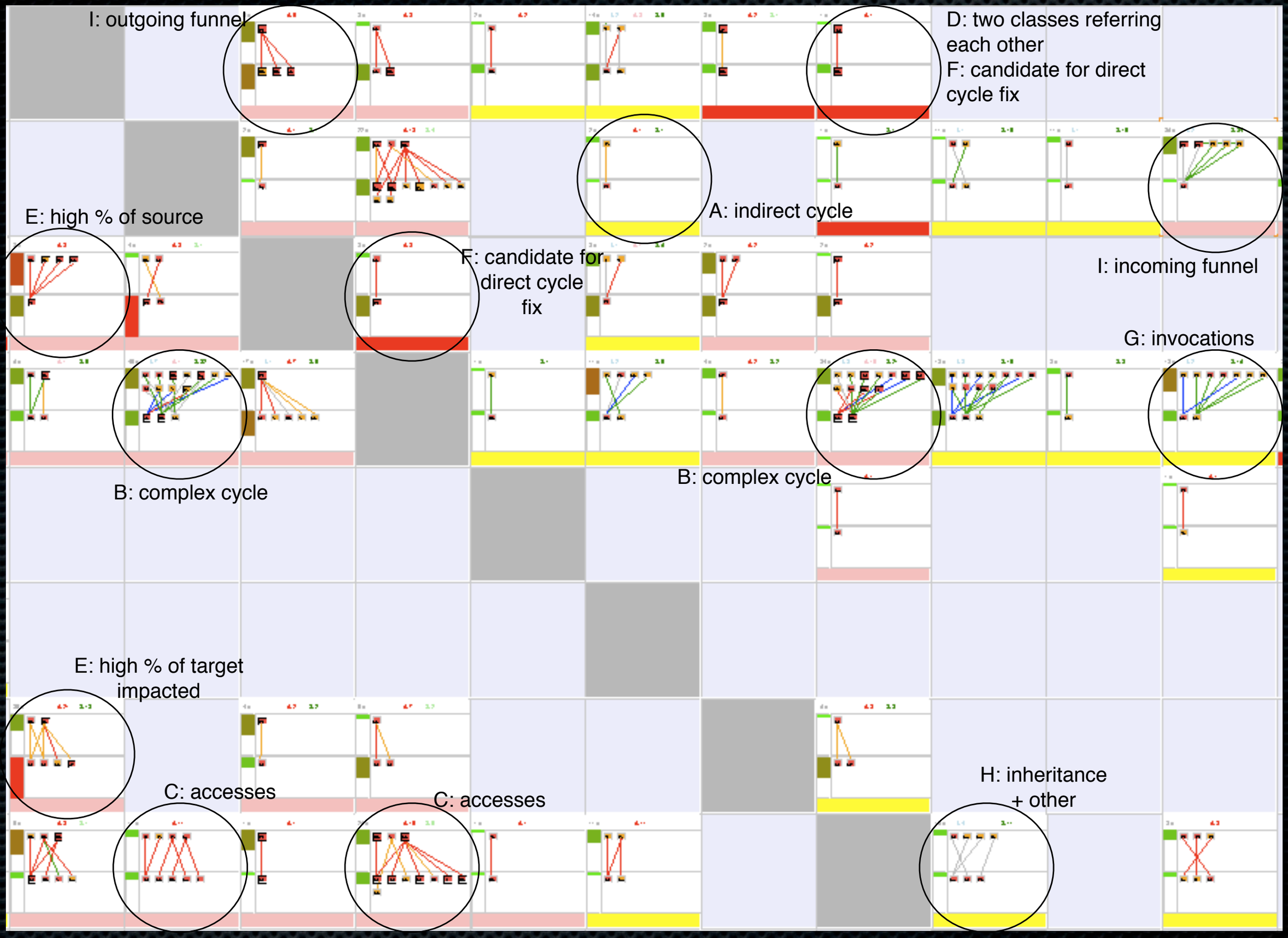
Causes and distribution



Causes and distribution






I: outgoing funnel
D: two classes referring each other
F: candidate for direct cycle fix
E: high % of source
A: indirect cycle
F: candidate for direct cycle fix
I: incoming funnel
G: invocations
B: complex cycle
B: complex cycle
E: high % of target impacted
C: accesses
C: accesses
H: inheritance + other

Challenges

- How to help taking the right decision?
- What are possible futures impact of a change?

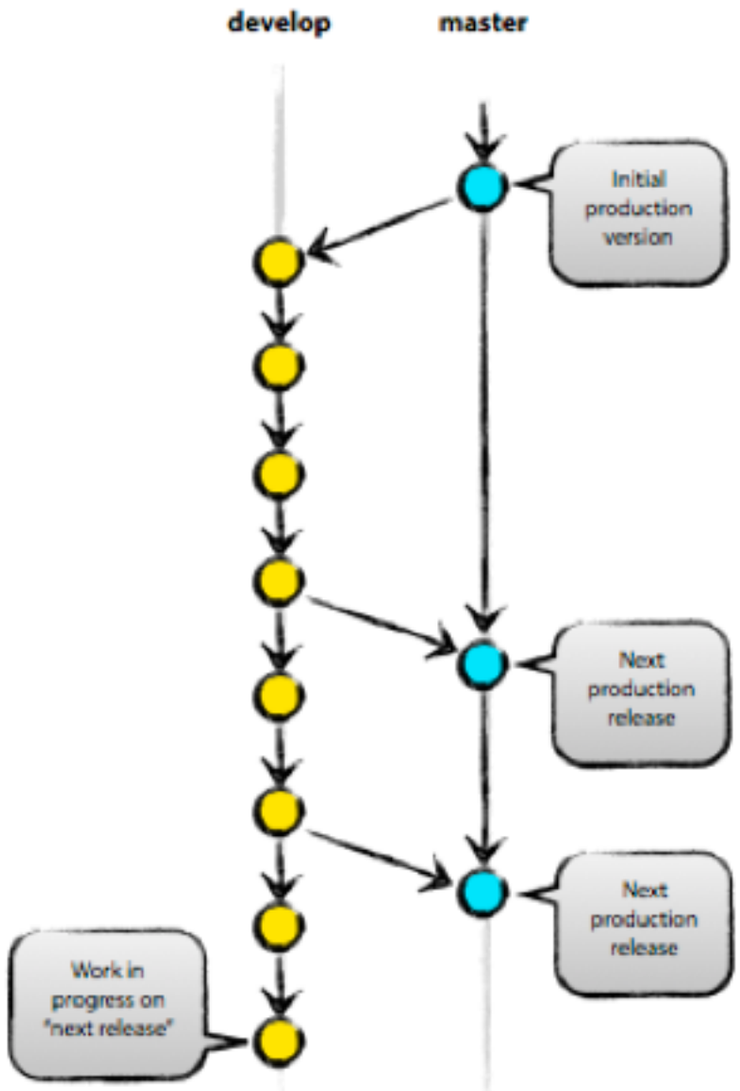
Orion

- Supporting multiple versions of analyzed projects
- Applying analyses on different modifications

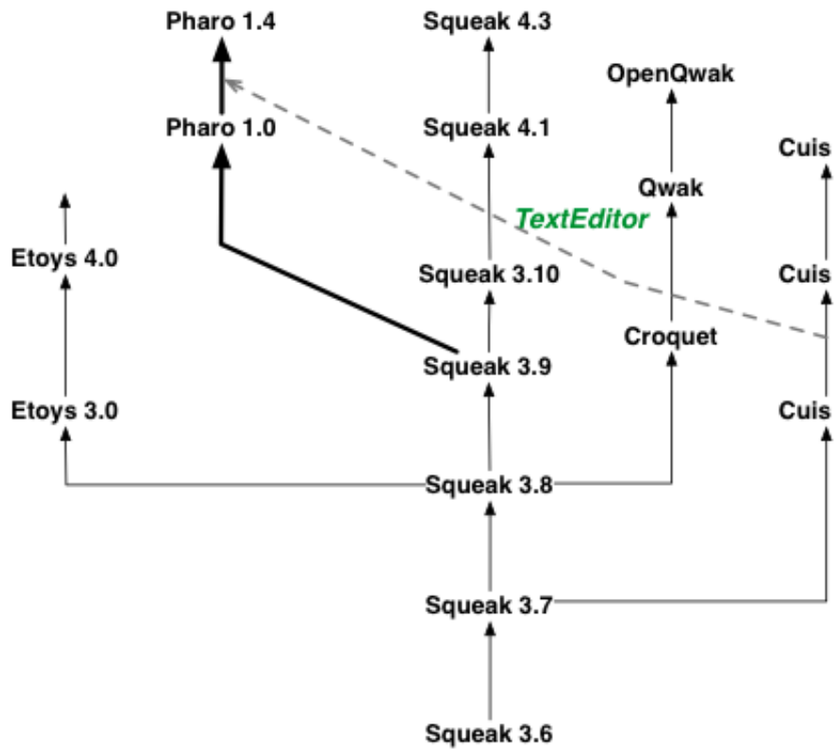
Challenges

- How can we help merging?
- What is the impact of a change?

How to support merging branches?



Git



Forks

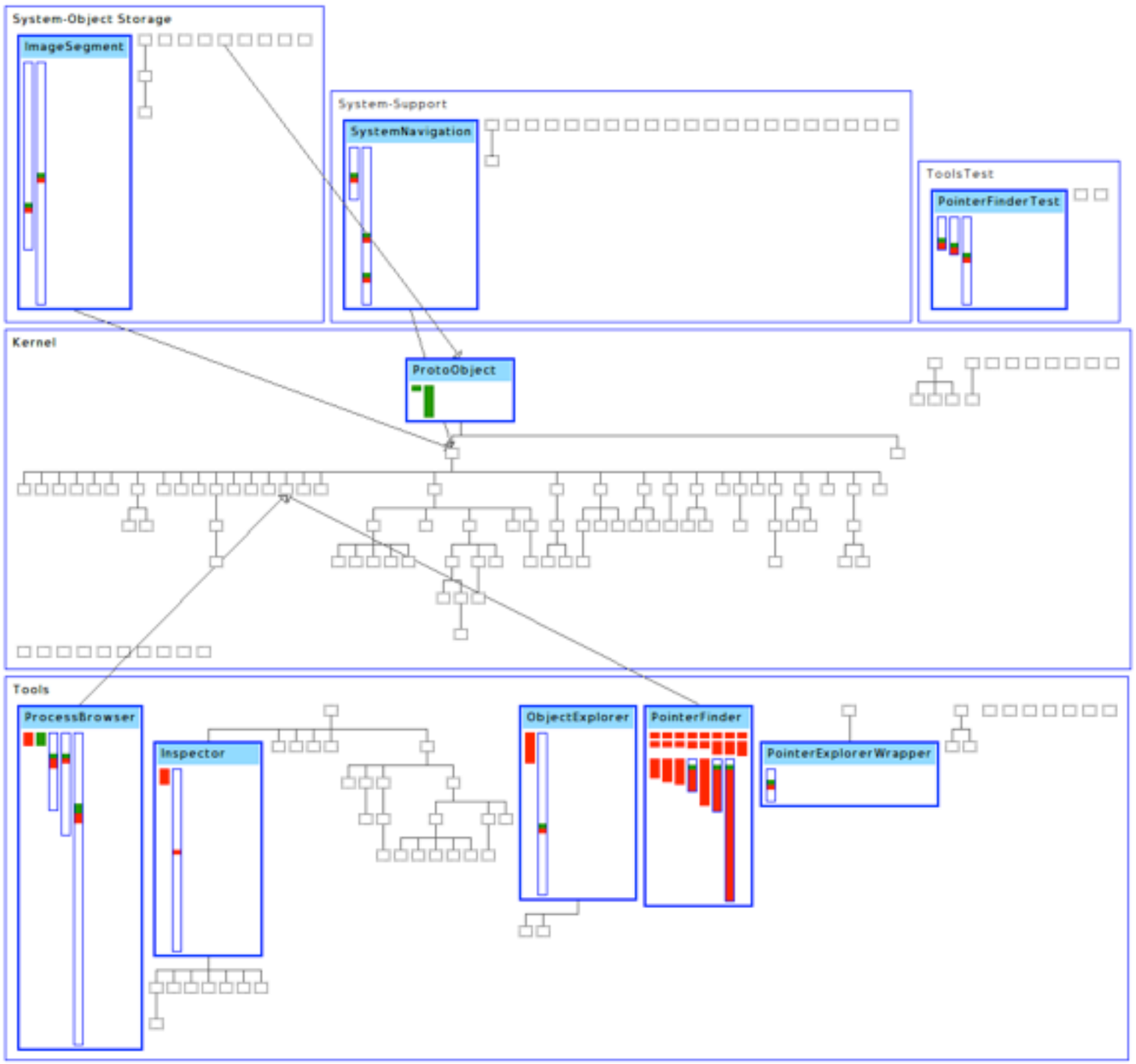
Manual tasks are needed

Dependencies between changes

Integrator is not the author of the changes

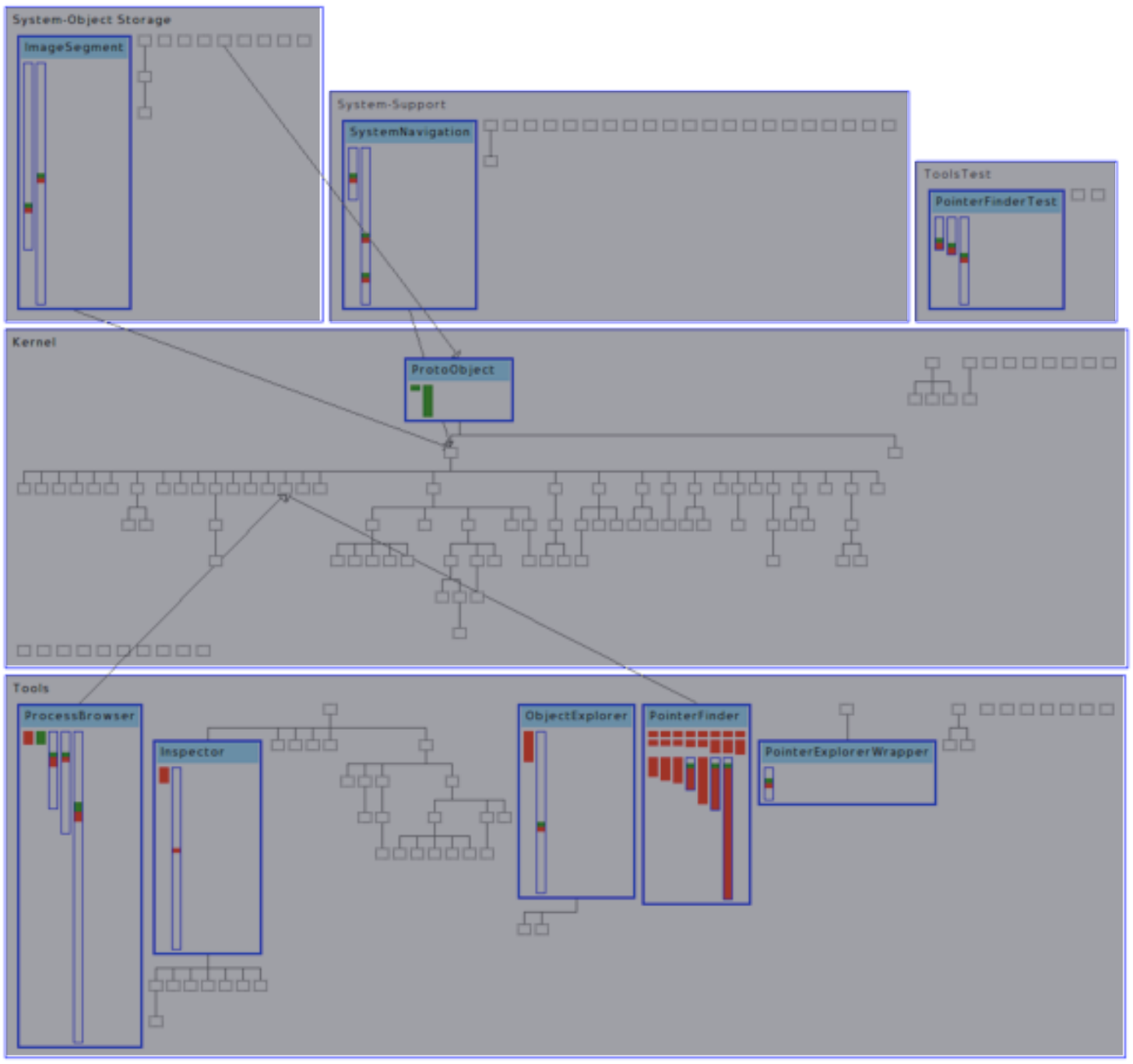
No guarantee that the system will work

Torch: Which changes?
Where? Who? What?



A set of changes, involving:

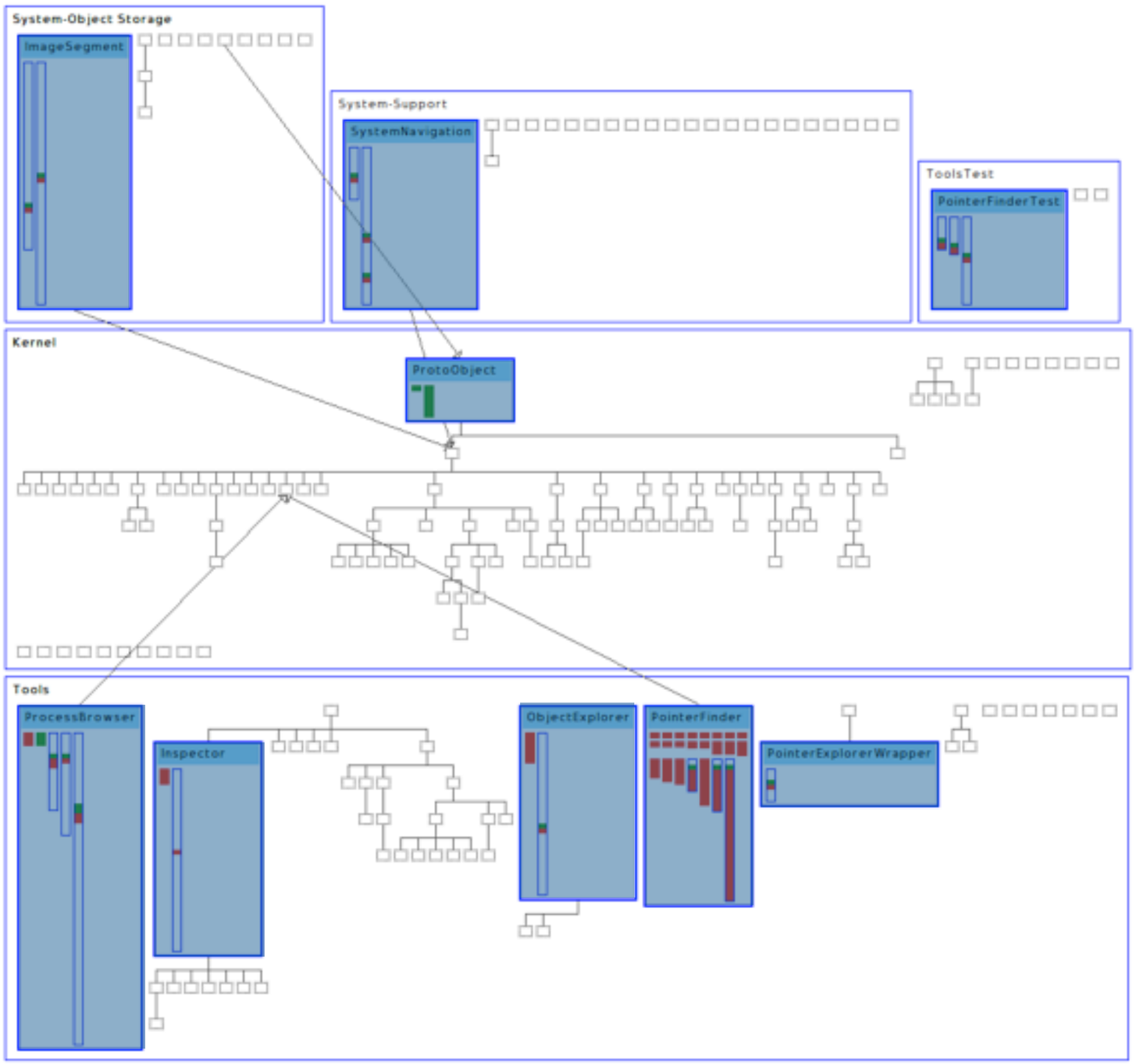
Torch: Which changes?
Where? Who? What?



A set of changes, involving:

5 packages,

Torch: Which changes?
Where? Who? What?

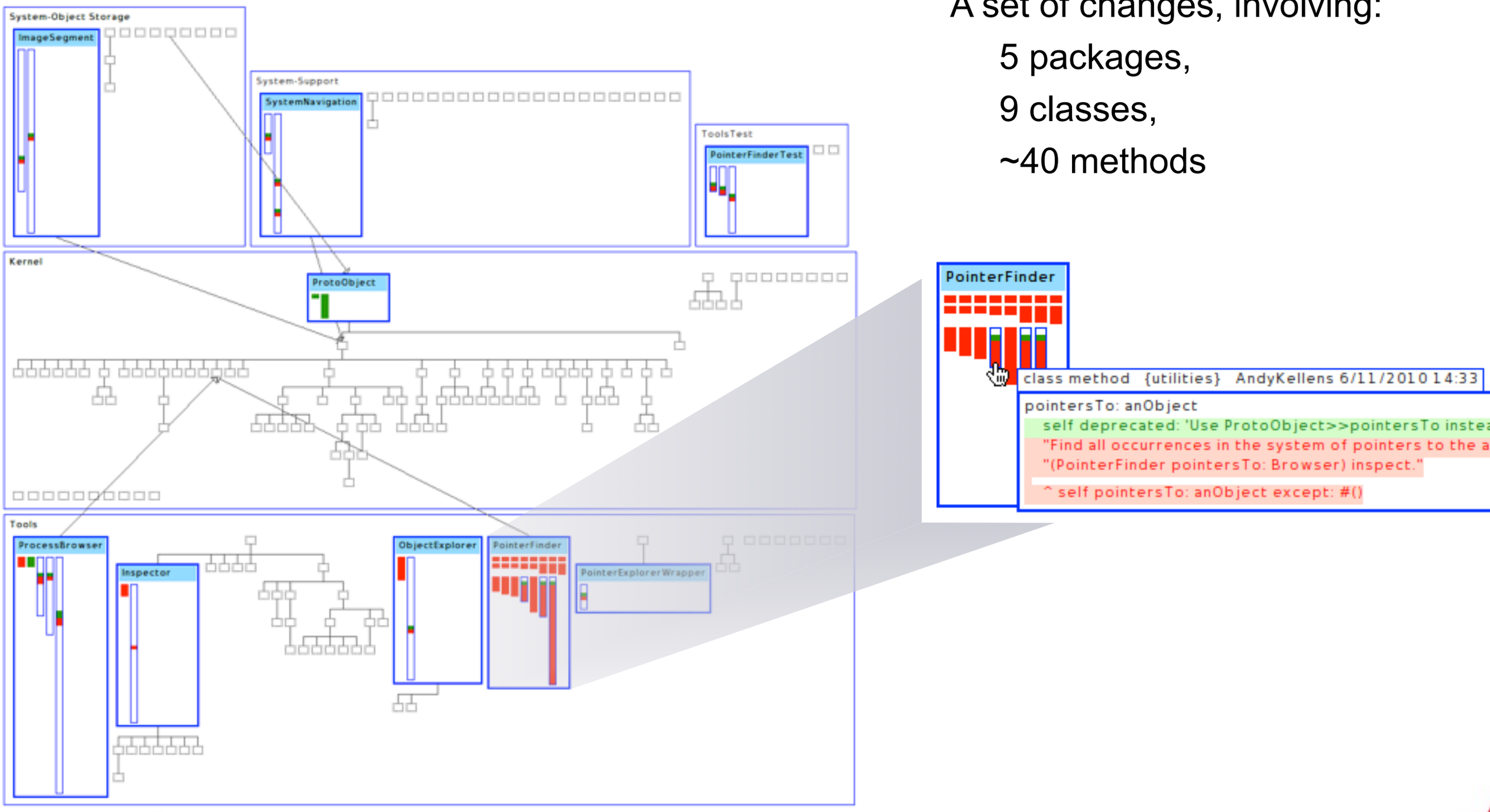


A set of changes, involving:

5 packages,

9 classes,

Torch: Which changes?
Where? Who? What?



A set of changes, involving:

5 packages,

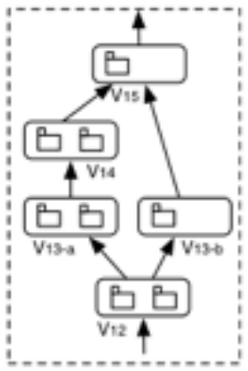
9 classes,

~40 methods

Streams of Changes:

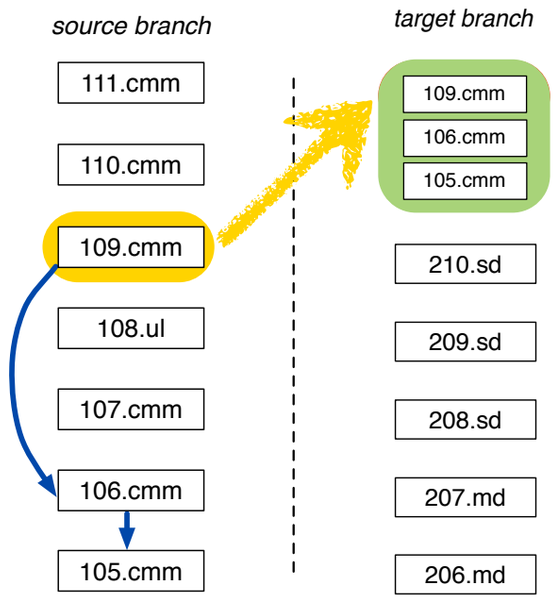
On what other changes does this change depend?

successive versions

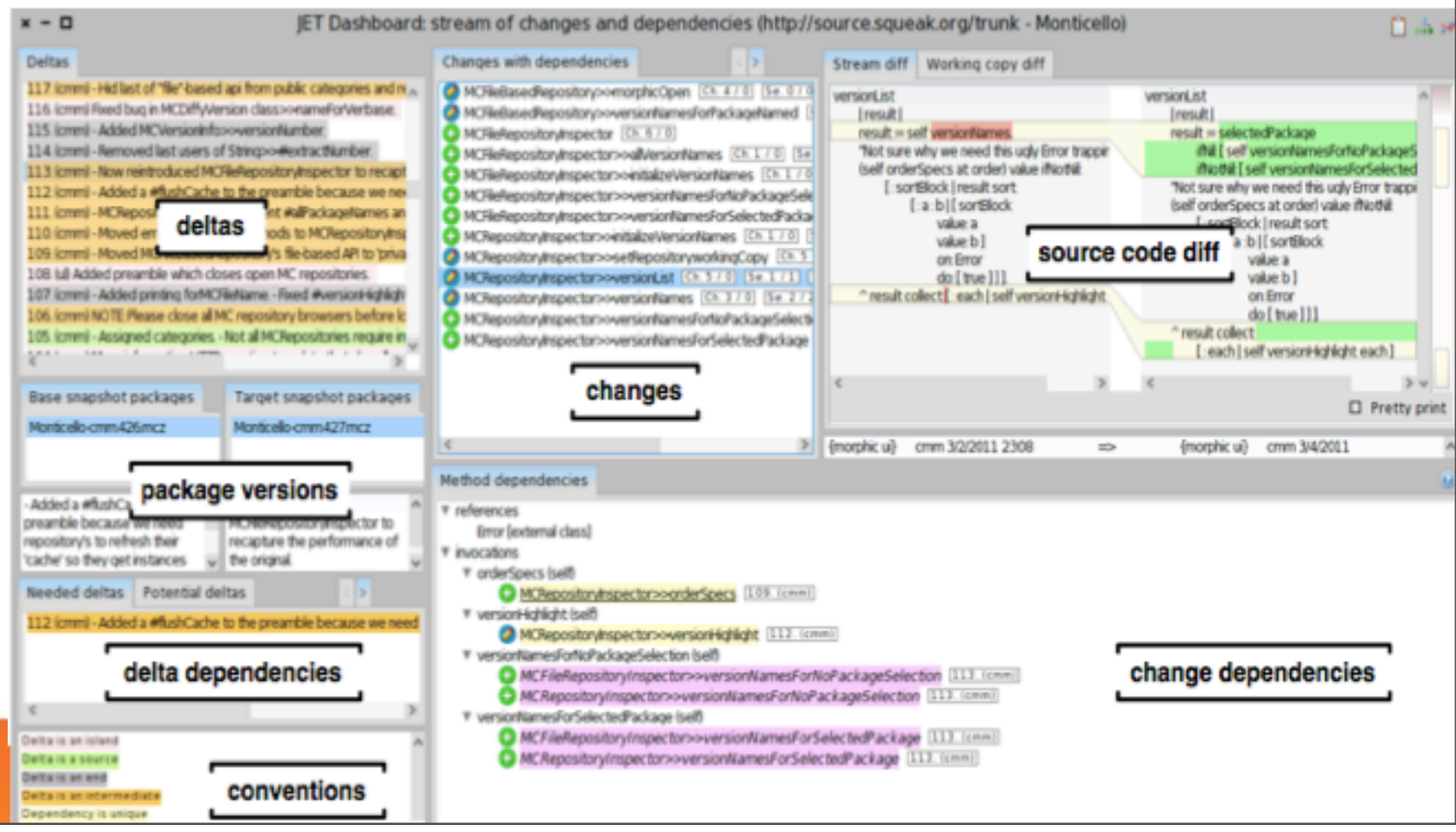


common ancestor analysis

dependency analysis



characterization of dependencies and deltas



Challenges

- How can we abstract from details but still access them and scale?
- FAMIX (language independent metamodel)
- OMG AST meta model

Maintenance is important

- http://rmod.lille.inria.fr
- http://www.synectique.eu