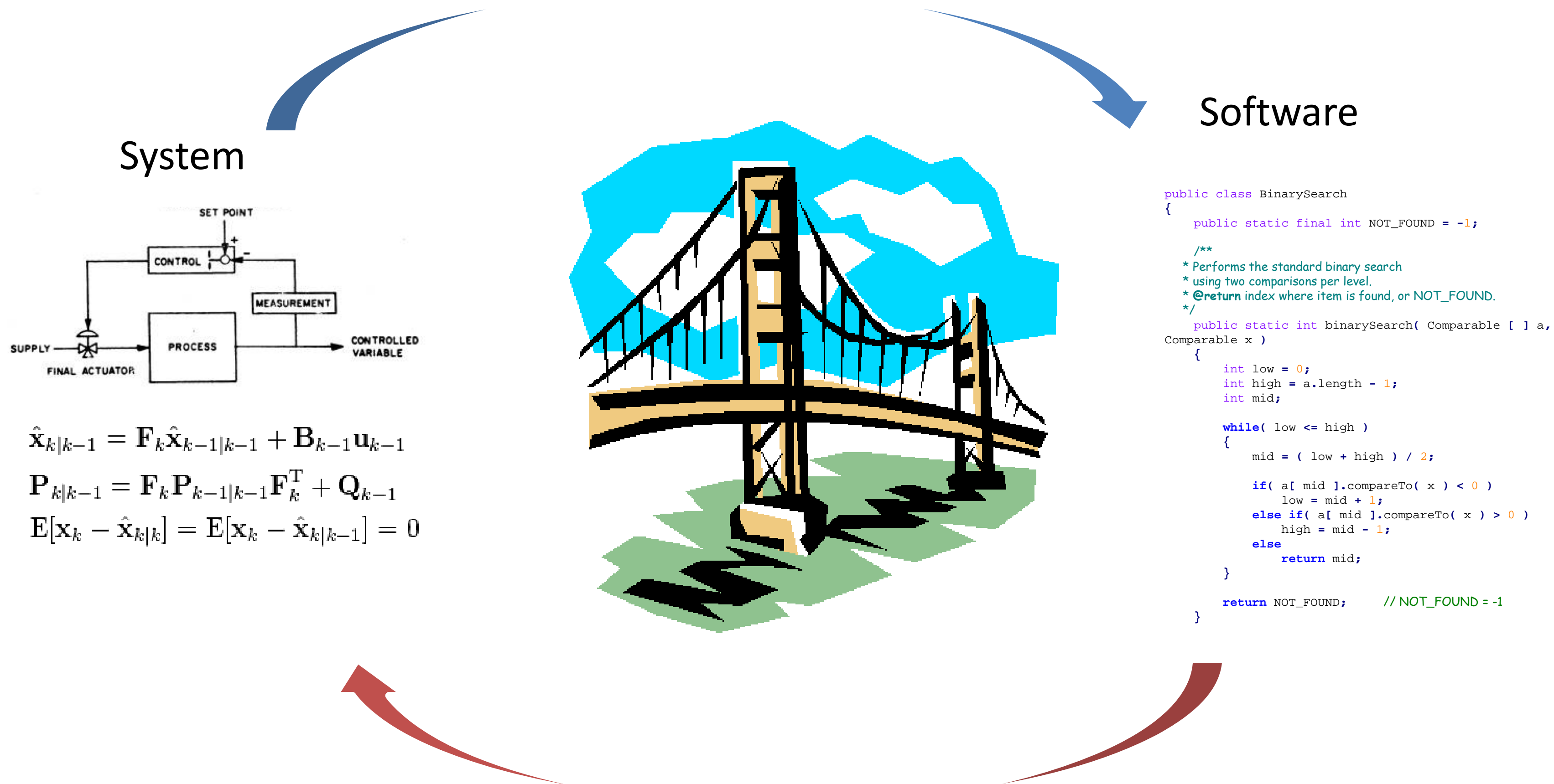
Identifying Implicitly Declared Self-Tuning Behavior through Dynamic Analysis

Hamoun Ghanbari, Marin Litoiu
Department of Computer Science
York University

ICSE 2009 Workshop
Software Engineering for
Adaptive and Self-Managing Systems (SEAMS)

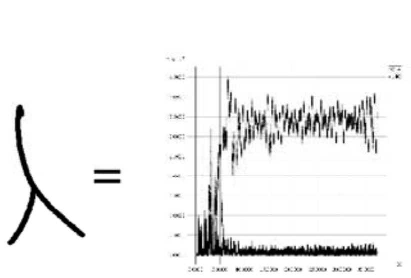
Software-system bridge

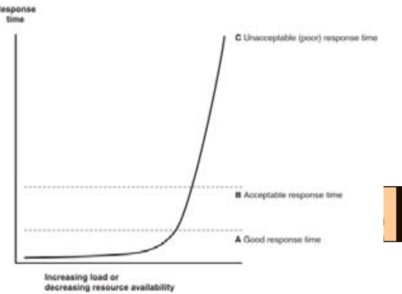


YORK U UNIVERSITÉ UNIVERSITY

A legacy software with hidden self-* capabilities

I have a software with 100000 LOC which employs some kind of autonomic self- behavior*



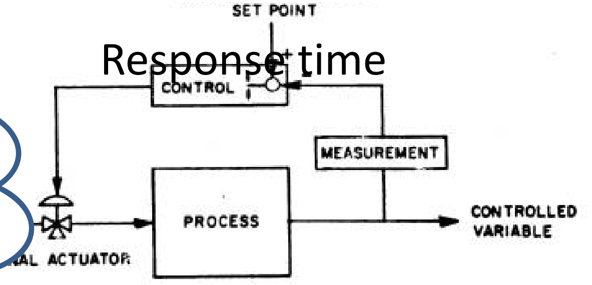


Arrival rate

Response time

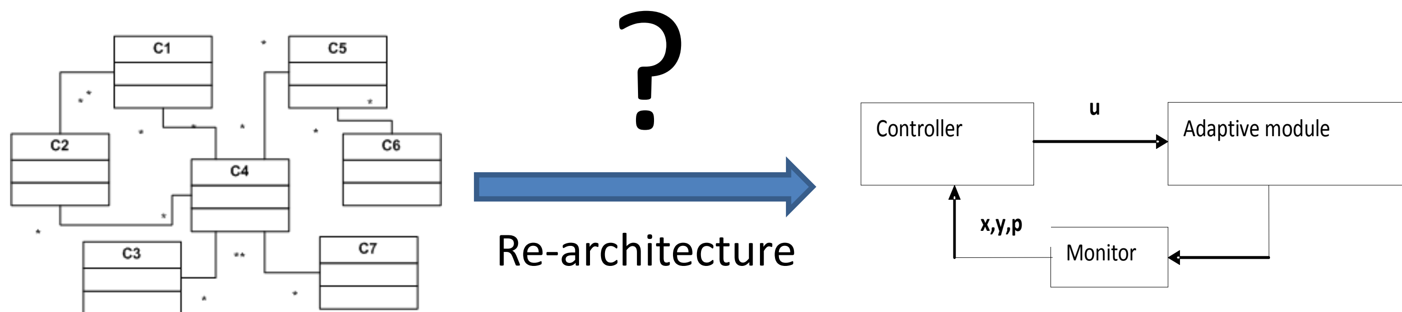
property by myself separately using:

```
...
if(responseTimeDrop >
    5% of averagaeResponseTime)
{
    workingThreads++;
}
...
```



YORK UNIVERSITÉ UNIVERSITY

Final goal



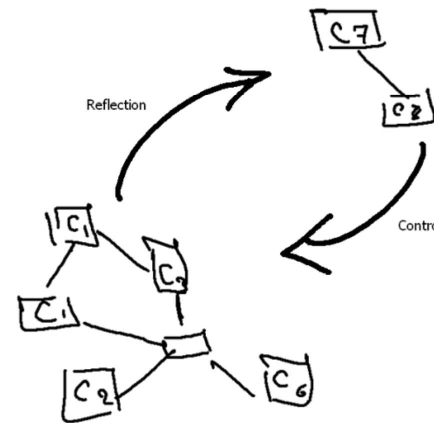
Integration of existing software into autonomic frameworks by:

1. Automating the identification of tuning parameters
2. Re-architecting the software to expose those parameters trough a separate control module.

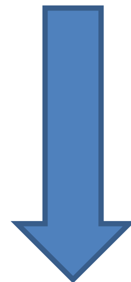
YORK U UNIVERSITÉ UNIVERSITY

Current approach

Self-tuning behavior is roughly considered to be composed of a *reflexive behavior* connected to a *control mechanism.*



How to find *control parameters* and *reflexive values* in software?

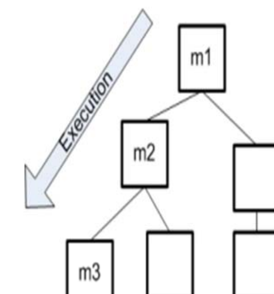


Mining program's state and behavior

1. Find the relationship between system's state and behavior
2. Classification of the results into the taxonomy of state-behavior signatures.

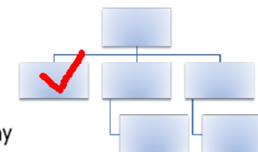
```
arrivalRate=4
socketTimout=7
isSynchronized=true
```

Corresponds to?



yes

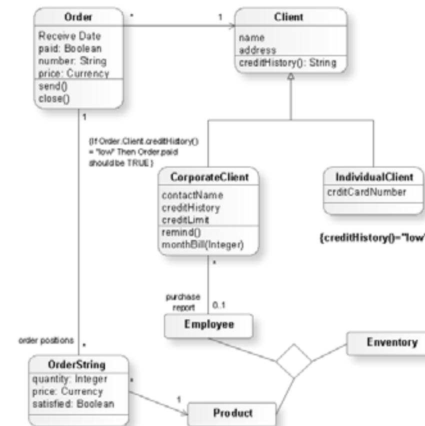
Classify the correspondence into the taxonomy



YORK UNIVERSITÉ UNIVERSITY

Related Work

1. STAC project: software tuning panels for autonomic control

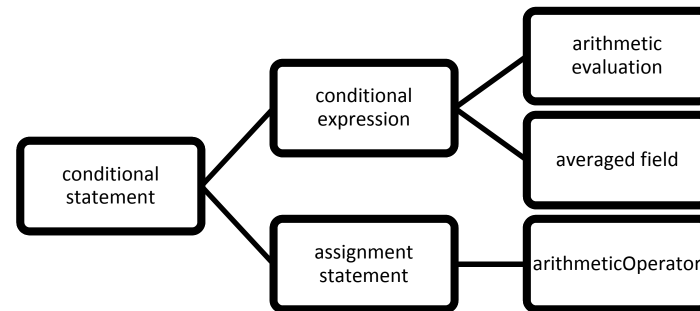


1. [3] N. Brake, J.R. Cordy, E.D. y, M. Litoiu, and V.P. u, "***Automating discovery of software tuning parameters***", *Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems*, ACM, Leipzig, Germany, 2008, pp. 65-72.

Static way

```
...
if(responseTimeDrop >
   5% of averagaeResponseTime)
{
    workingThreads++;
}
...
```

Convert to pattern



Structural pattern

YORK
UNIVERSITÉ
UNIVERSITY

Proposal:

Behavior is more expressive than structure

1. Instrument the program (AOP)
2. Transfer program's trace (data transfers/method invocations) into data model
3. Query the model
4. Classify the result

Software

Execute

Store

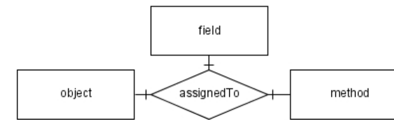
Query

Classify

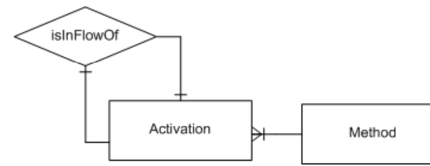
YORK
UNIVERSITÉ
UNIVERSITY

Data model

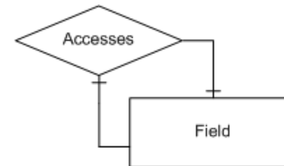
1. *Aliasing*



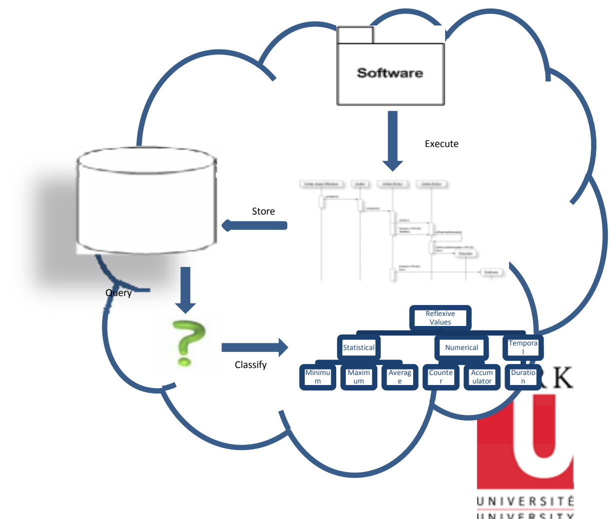
2. *Activation flow*



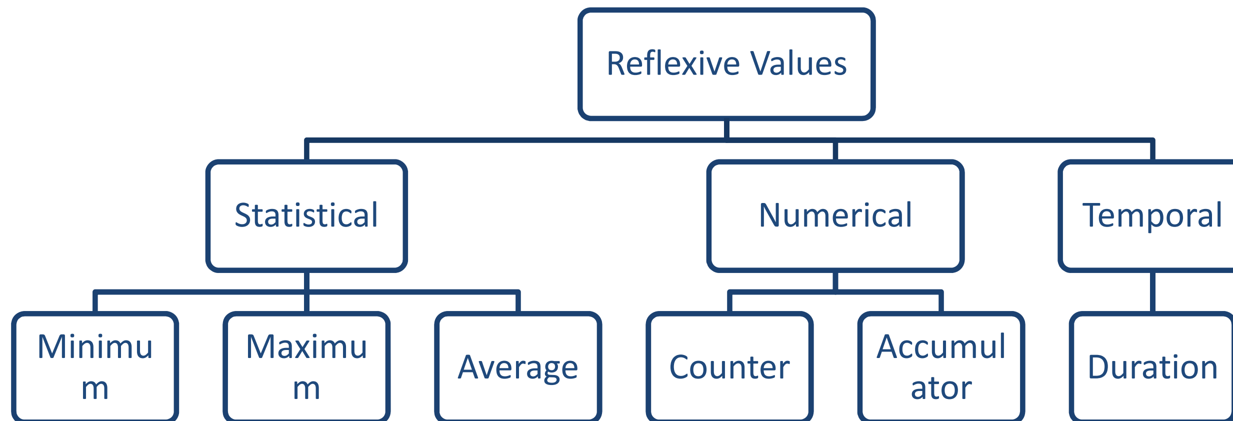
3. *Static reachability*



4. *Distance metric*



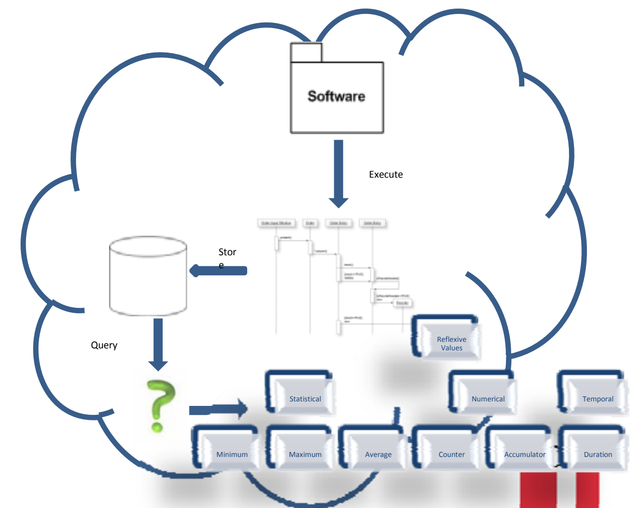
Reflexive value (sensor) patterns



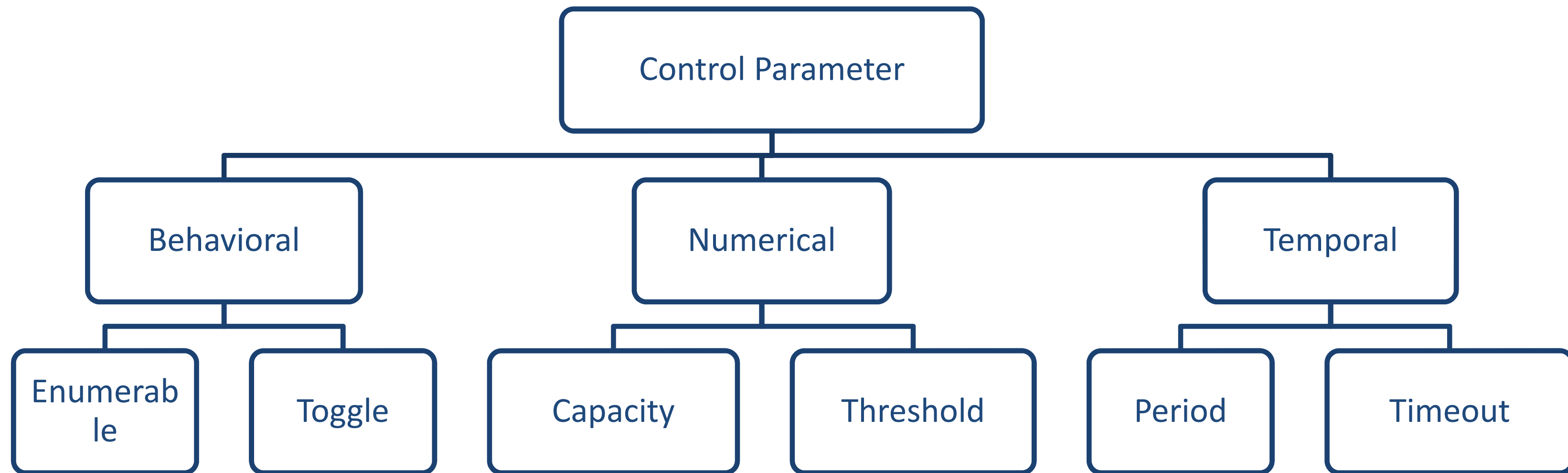
Counter: Measures how many times an event or action has occurred using discrete, constant increments (e.g., number of active connections).

Duration: Measures the length of time some event or action has been

Average: Measures the statistical average of some numerical or temporal parameter.



Control (actuator) patterns

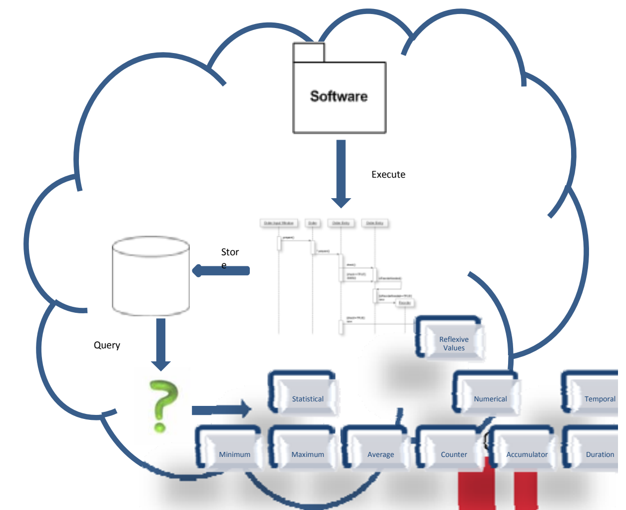


Capacity limit (to how large (i.e. memory footprint) a resource can grow (e.g. buffer limits).

Enumerable: Controls the behavior of the system from a finite set of states (e.g. thread priorities, algorithm selection).

Toggle: Enables or disables a behavior of the system through some (e.g. caching enabled).

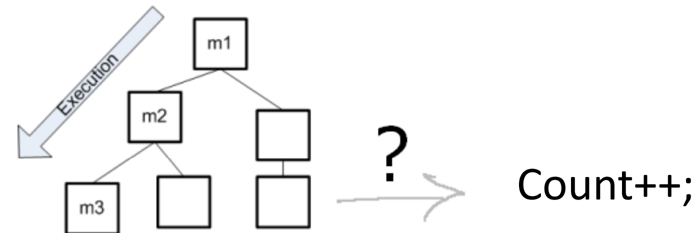
Timeout: the length of time an action has to complete before it will be interrupted (e.g., reading bytes from a network connection).



Mining reflexive patterns

1. Identifying counter pattern:

```
isCounter (fld1,counter_obj,fis):-
        isIncremented(m1, counter_obj,fld1),!,
        callStackAfterExecutionOf(m1,callStack),
        fis=frequentItemSetOf(callStack).
```



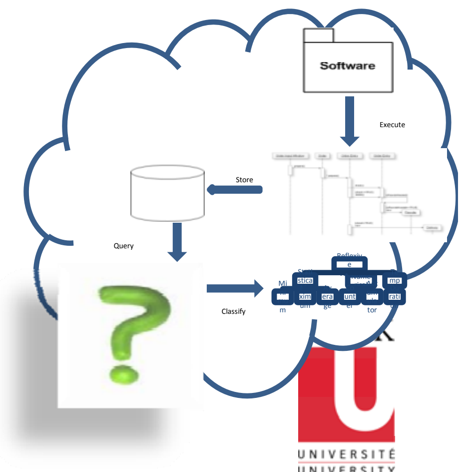
2. Identifying Accumulator pattern:

```
isAccumulator(accumulator,fld1,accumulated_obj,fld2)
        :-aliasedTo(m1,accumulator,fld1,obj1),
        aliasedTo(m2,accumulator,fld1,obj2),
        d=distance(obj1,obj2),
        aliasedTo(_,accumulated_obj,fld2,accum_val)
        ,equals(accum_val,d).
```

```
public class C1{
  public static int a=0;
}
```

→ C1.a=0;
t1 → acc=4
→ C1.a=1;
t2 → acc=5

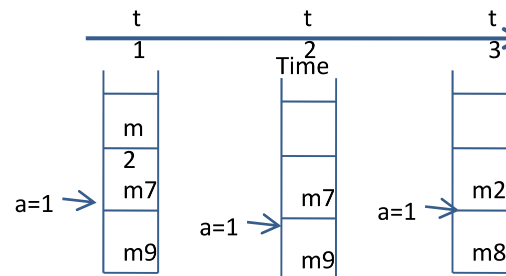
Execution

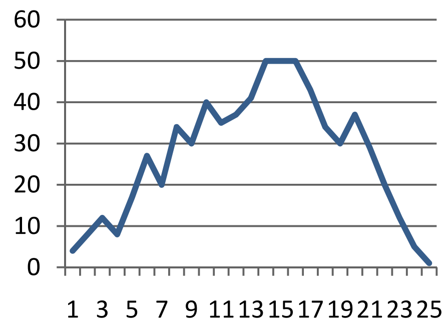


Mining control patterns

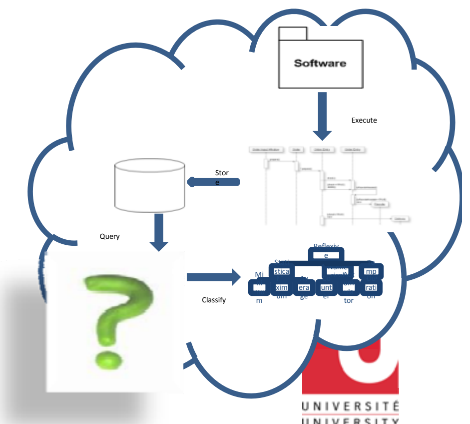
- Mining execution control patterns:
  1. Attach variables and call stack together in method activations
  2. Identify a common subset of call stack and variables values.
- Mining temporal/physical :

```
isToggling(fld1,container,fis):-
isLastValueOf(container,fld1,m1,fld_value),
callStackAfterExecutionOf(m1,callStack),
fis=frequentItemSetOf([fld_value|callStack]).
```






```
int maxSize=50;
public onArrival(Packet
packet){
    if (!bufferSize > maxSize){
        insert(packet);
    }
}
```



Mining self-tuning loop (reflection + control)

Example: a web server might increase the connection pool size if the maximum number of faults per second exceeds some amount.

```
{
 update(maxReqPerSec);
 boolean increaseSize=
    curFaultRate> maxFaultRate;
 if (increaseSize){
      maxFaultRate= curFaultRate +
      (curFaultRate -maxFaultRate)*0.1;
    conPool.setSize(curSize*1.2);
 }
}
```

We encode the full control loop:

- **reflexive parameter**: *maximum* number of faults per second.
- **controlled behavior**: increasing the size of connection pool.

isMaximumToggleLoop():-
  isMaximum(container1,maximum_fld),
  isToggling(boolean_fld,container2,partOfStack),
  hasCorrespondence(maximum_fld,boolean_fld).

YORK
UNIVERSITÉ
UNIVERSITY

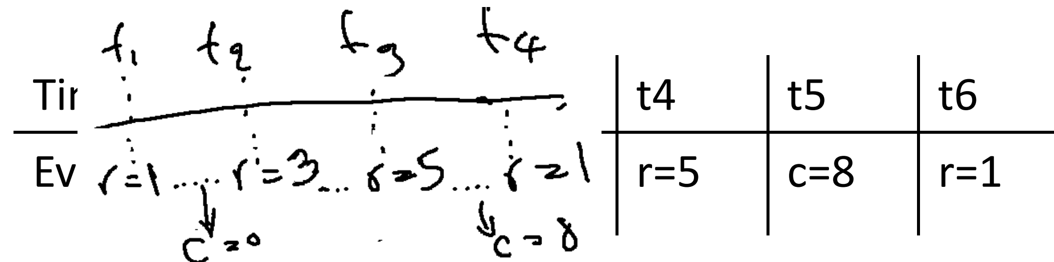
The Predicate hasCorrespondence()

Consider this static structure:

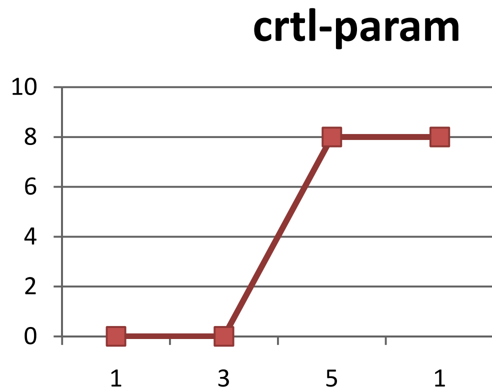
```java
public class C1{
    public int reflexiveParam;
}

public class C2{
    public int controlParam;
}
```

And This execution:



Correspondence(r,c)={<1,0>, <3,0>, <5,8>, <1,8>}



crtl-param

Could we find a regression? → Yes.->hasCorrespondence(c,r)=true

YORK UNIVERSITÉ UNIVERSITY

Example:
A behavioral pattern: Toggle

- An example of a toggle named disableUploadTimeout is mentioned.
- This toggle controls whether timeouts are employed when reading from a socket.
- Whenever the `disableUploadTimeout` is false `setSoTimeout` is called.

```
class Http11Processor {

public boolean disableUploadTimeout = false;
pusblic int keepAliveTimeout;
public int soTimeout;
public boolean keptAlive;
//...
public void process(Socket socket) throws
IOException {
  //...
  if (!disableUploadTimeout && keptAlive) {
    if (keepAliveTimeout > 0)
          socket.setSoTimeout(keepAliveTimeout);
    }
    else if (soTimeout > 0) {
          socket.setSoTimeout(soTimeout);
    }
  }
  //...
  if (!disableUploadTimeout) {
    int timeout = 0;
    socket.setSoTimeout(timeout);
  }
}
}
```

is true

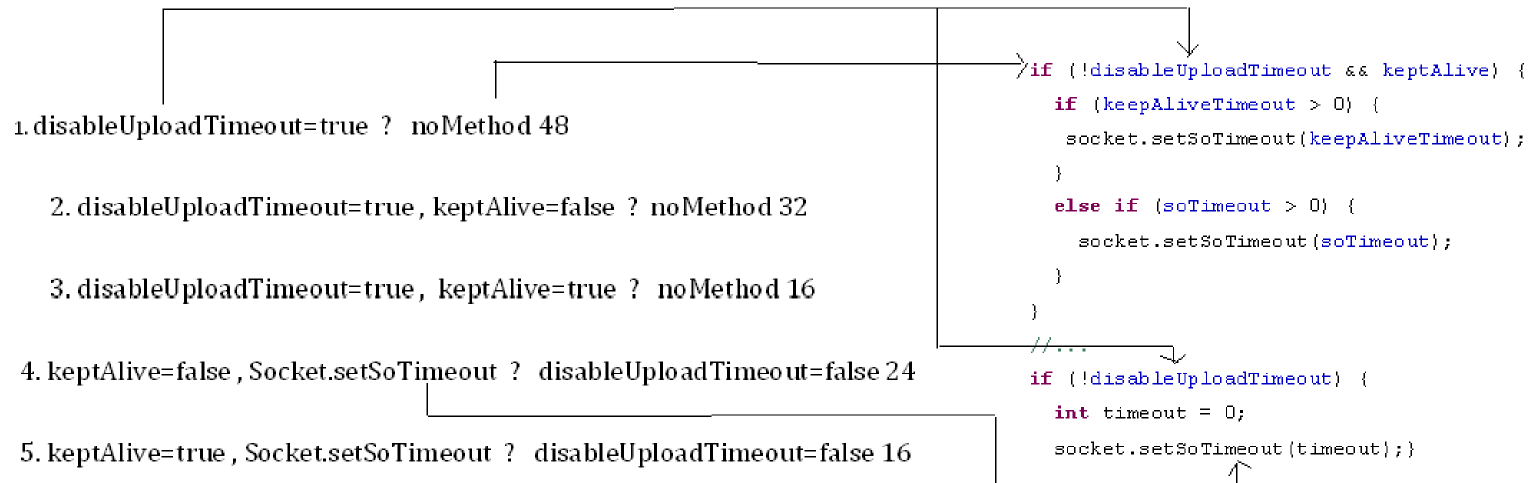
is false

is false

is false

Example:
A behavioral pattern: Toggle

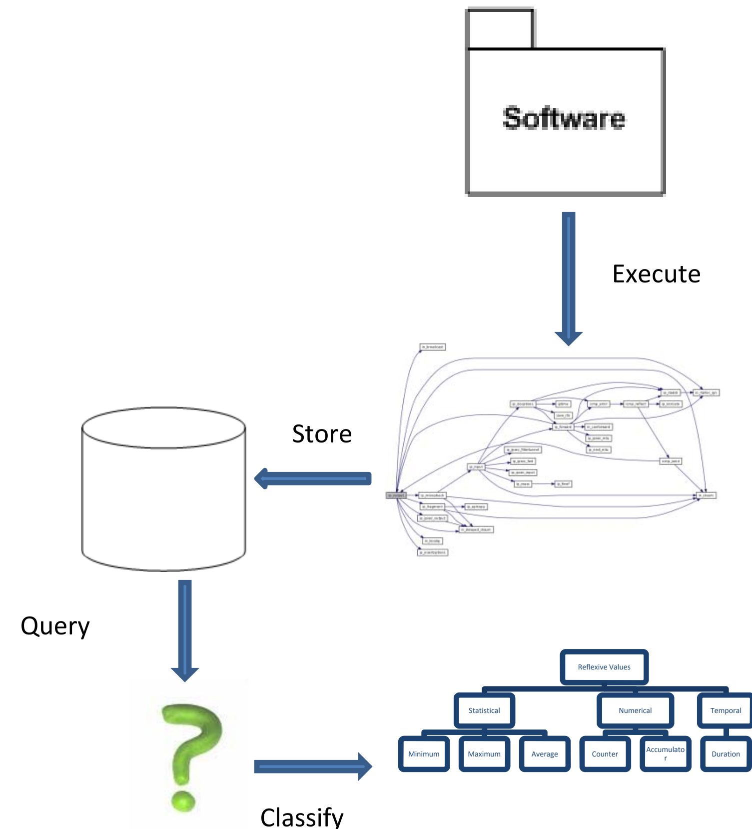
- applying association rule mining proves presence of toggle behavior:
  - method Socket.setSoTimeout *correlates* with the value of disableUploadTimeout.



YORK UNIVERSITÉ UNIVERSITY

Conclusion

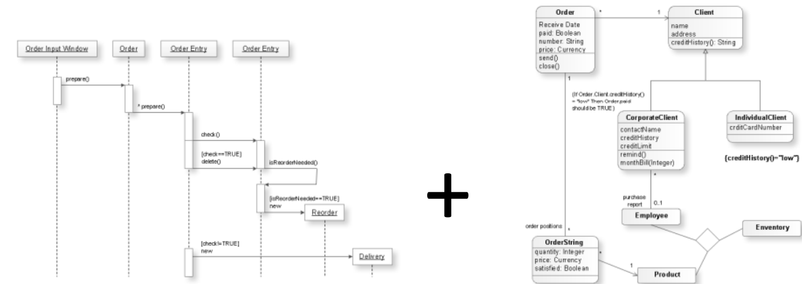
- we introduced a new techni que to identify tuning parameters using dynamic analysis.
- Tested on different types of tuning parameters: statistical, numerical, temporal.
- Our approach can be used as a filter for static analysis.



YORK U UNIVERSITÉ UNIVERSITY

Future Work

- Combining static and dynamic analysis could help addressing the scalability issue in larger applications
- Identifying the implicitly declared self-tuning behavior instead of just self-tuning parameters.
- Using sophisticated probabilistic approaches.



$$P(a) = \left[\frac{\sum_{i=0}^{a-1} \left(\frac{1-p}{p} \right)^i}{\sum_{i=0}^{a+b-1} \left(\frac{1-p}{p} \right)^i} \right]$$

YORK
UNIVERSITÉ
UNIVERSITY

YORK U UNIVERSITÉ UNIVERSITY