Unified Software Method (USM) : Towards a Method of Measurement of the Necessary Changes to Software in Maintenance

Stéphane Mercier, A. Abran M. Lavoie, R. Champagne

International Workshop Software Measurement (IWSM 2006)
November 2-3, 2006 Potsdam, Germany

Agenda

- Introduction
- What is the USM about?
- The proposed measurement method
- A case study
- Conclusion

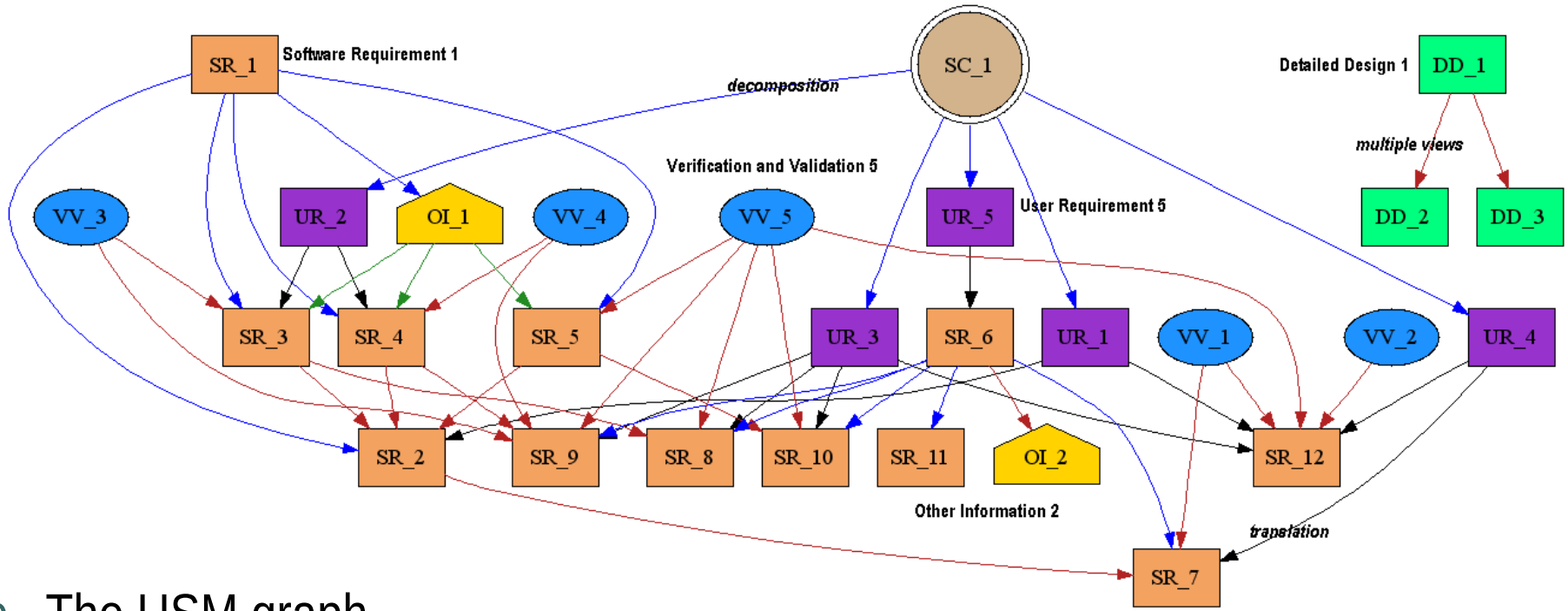
Introduction

- Back annotation problem
  - Almost impossible to achieve backward traceability
  - Creates undesired artificial software aging
- Software aging
  - Natural
    - Can be predicted and anticipated
    - Technical obsolescence, incompatibility with new technology, etc.
  - Artificial
    - Undesired and induced by human
    - Progressive degradation of synchronization, caused by successive modifications without appropriate updating of documentation
    - When reaching critical condition, rewriting the application becomes more productive then maintaining it

What is the USM about?

- A solution to the back annotation problem
- Organizing information of software project in form of a graph
  - To offer complete traceability of the software project information
  - Without being constrained by a methodology or a particular process
  - To maintain the synchronization of the information
- USM information structure must be adapted to humans
  - *Be able to master complexity*
  - *Applying the Keep It Simple (KIS) principle*
    - *7 +/- 2 elements*
    - *Up to 4 parameters (relational complexity)*
    - *Max info in min time using min space with min printed element*

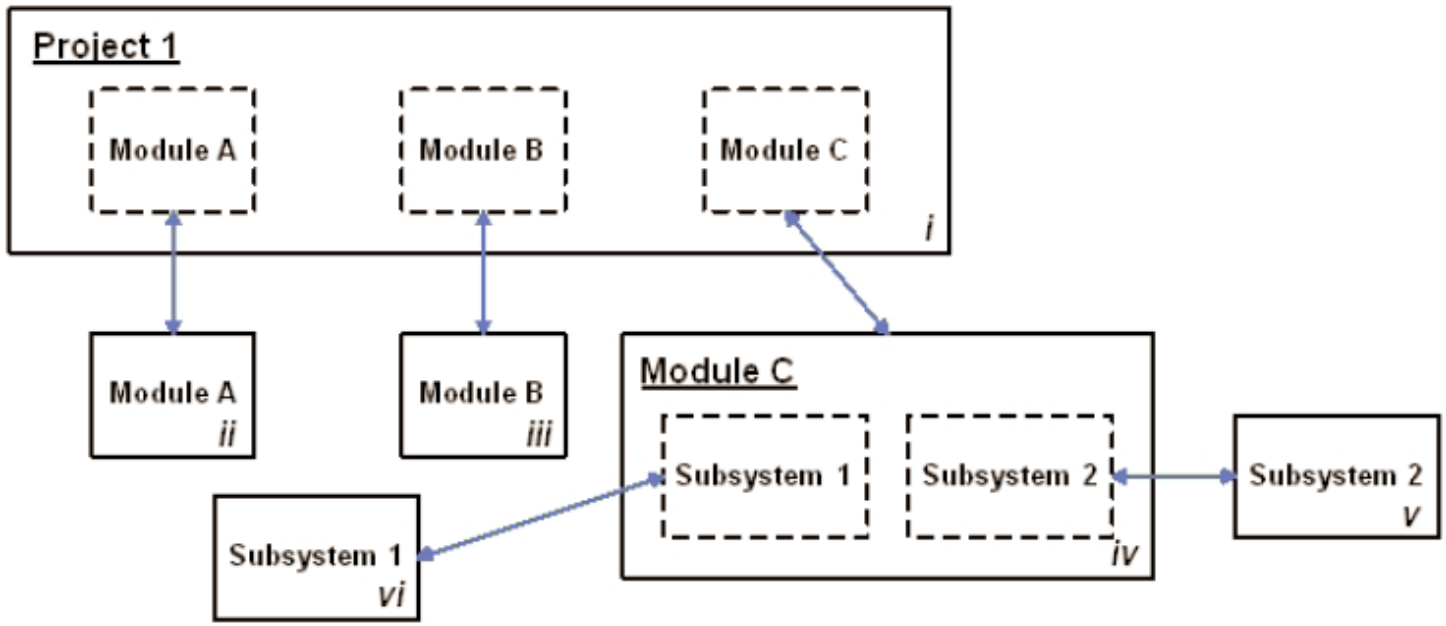
What is the USM about?



- The USM graph
  - Node: Address of the project information
  - Link: Traceability between the information

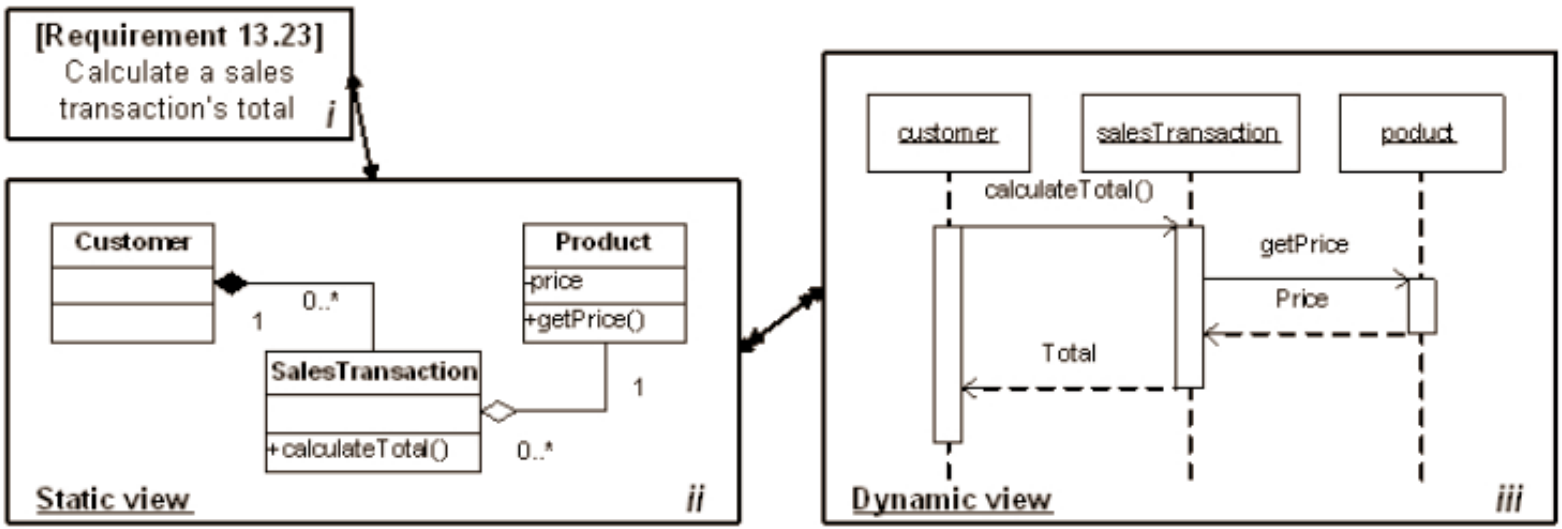
What is the USM about?

- Decomposition link
  - Managing the information complexity
  - Divide to conquer



What is the USM about?

- Multiple-view link
  - To have a better understanding of the information
  - Different representation gives different perspective
    - *What? Where? How? Who? Why? …*



Some maintenance measurement methods

- Measure that give useful information is about
  - Go, do the maintenance or
  - Stop, rewrite the application
  - But if you have people working on the project, they can answer more accurately to this Go/ Stop question

- Beware of methods that are mathematically unacceptable

$$MI = 171 - 5.2\ln(aveV) - 0.23aveV(g') - 16.2\ln(aveLOC) + 50\sin(\sqrt{2.4\,perCM})$$

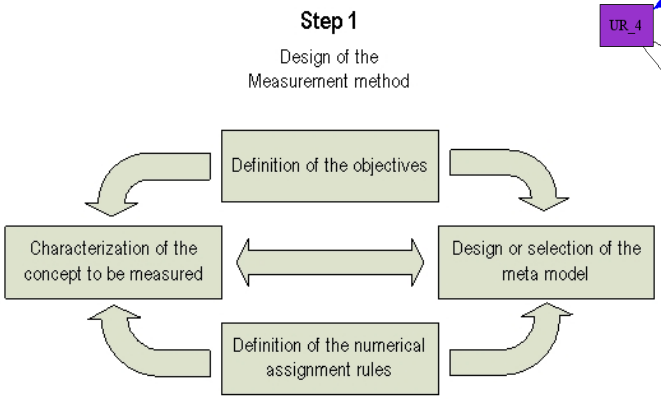
MI = ( ) – (HalsteadVolume) – (CyclomaticComplexity) - (LineOfCode) + {%Comment}

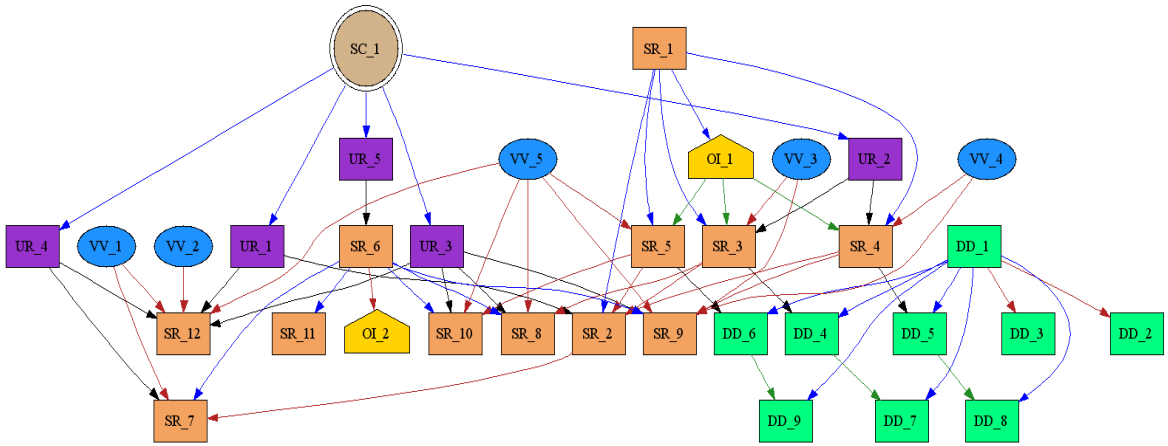
The proposed measurement method

- Measurement process [Jacquet & Abran 1997]
  - Step 1: Design of the measurement method
  - Step 2: Application of the rules of the measurement method
  - Step 3: Analysis of the measurement results
  - Step 4: Exploitation of the measurement results

Step 1 : Design of the measurement method

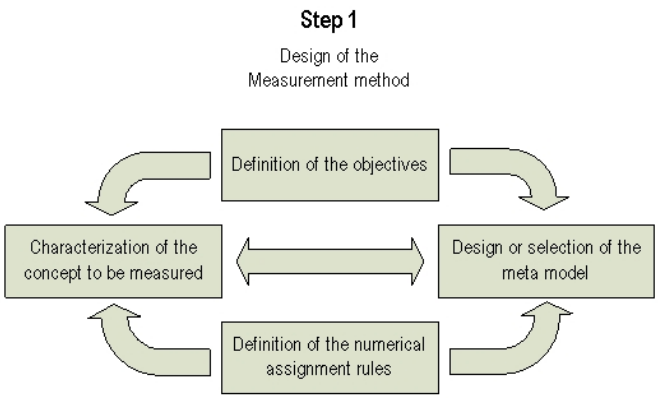
- Definition of the objectives
  - To identify information nodes requiring an update as a result of a modification or removal maintenance activity





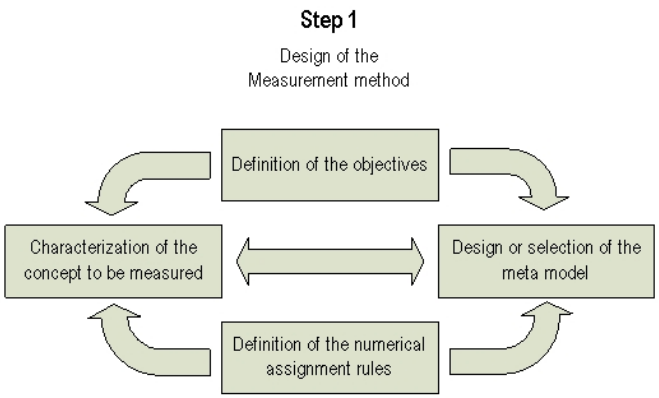
Step 1 : Design of the measurement method

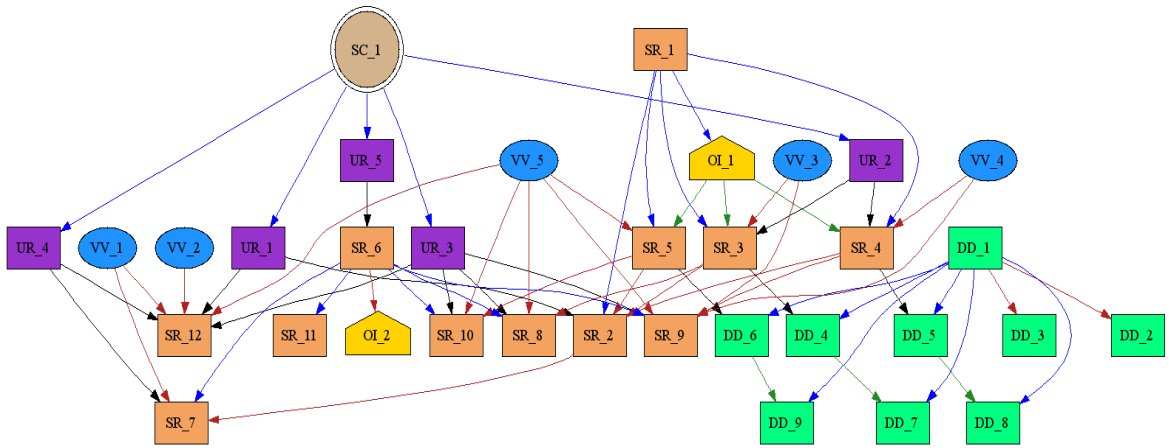
- Characterization of the concept to be measured
  - Because keeping synchronization is mandatory in the USM
    - All nodes which are either directly or indirectly influenced by a maintenance context considered must be identified
  - Associated to this measurement result we can also obtain
    - The node count related to the maintenance context
    - The Node proportion related to the maintenance context



Step 1 : Design of the measurement method

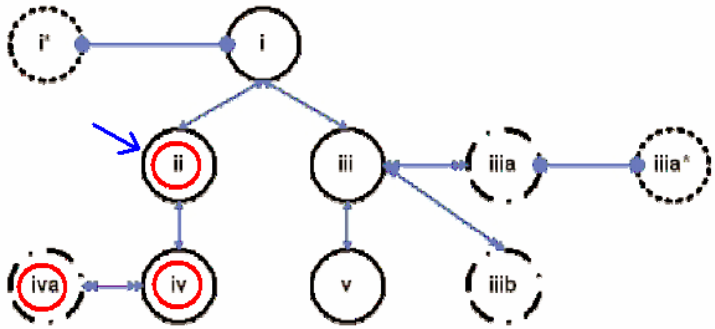
- Selection of the meta-model
  - Using a USM graph where
    - Nodes that represent project information is the concept to be measured
    - Links are essential to the measurement mechanism but are not considered in the measurement result





Step 1 : Design of the measurement method

- Definition of the numerical assignment rules



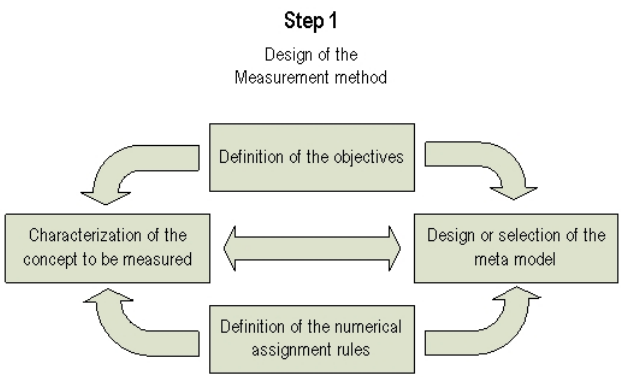
- Which ones ?
  - The red ones

$$ni = \sum i_m$$

- How many ?
  - 3

$$\%ni = \frac{\sum i_m}{\sum i} * 100$$

- How much ?
  - 30%



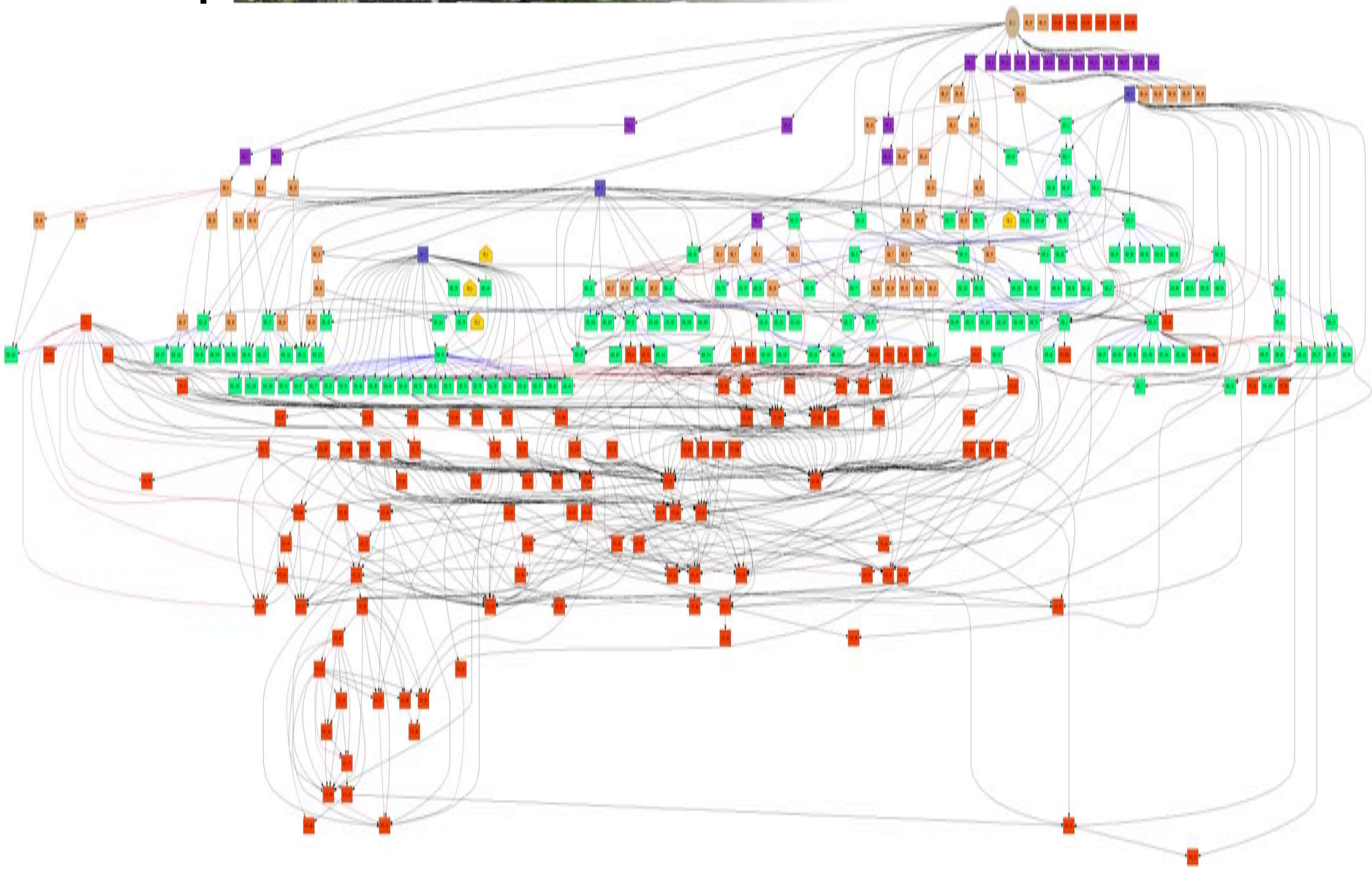
A case study

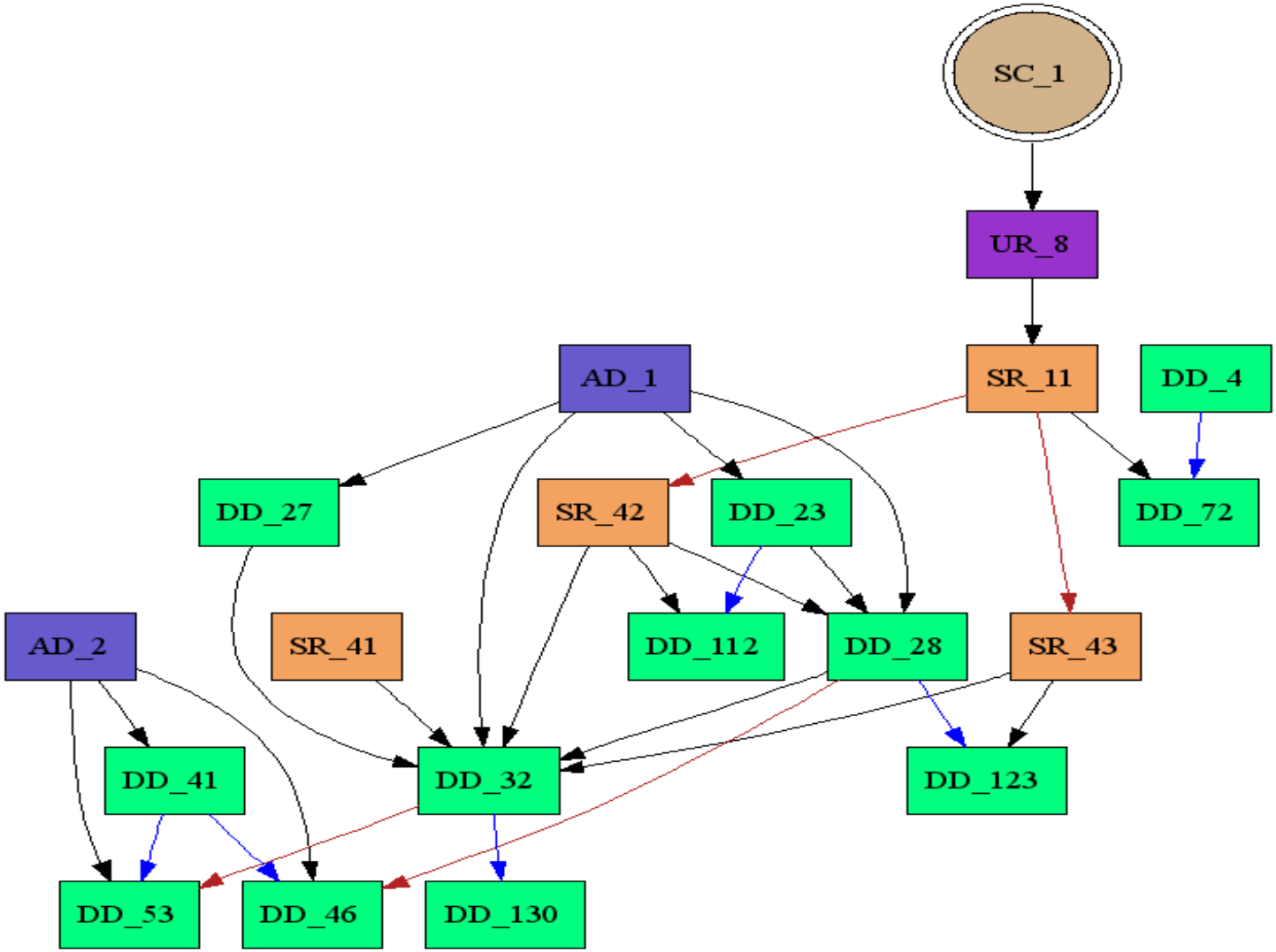
- Some statistics
  - 10 features
  - 22 use cases
  - 2 functional requirements
  - 3 non-functional requirements
  - 1 architectural diagram
  - 64 classes (design)
  - 18 .jsp files
  - 71 .java files

Another way to represent statistics

- Elements of information
  - 20 nodes associated to user requirement
  - 50 nodes associated to software specification
  - 3 nodes associated to architectural design
  - 136 nodes associated to detailed design
  - 120 nodes associated to code
  - 4 nodes associated to other information

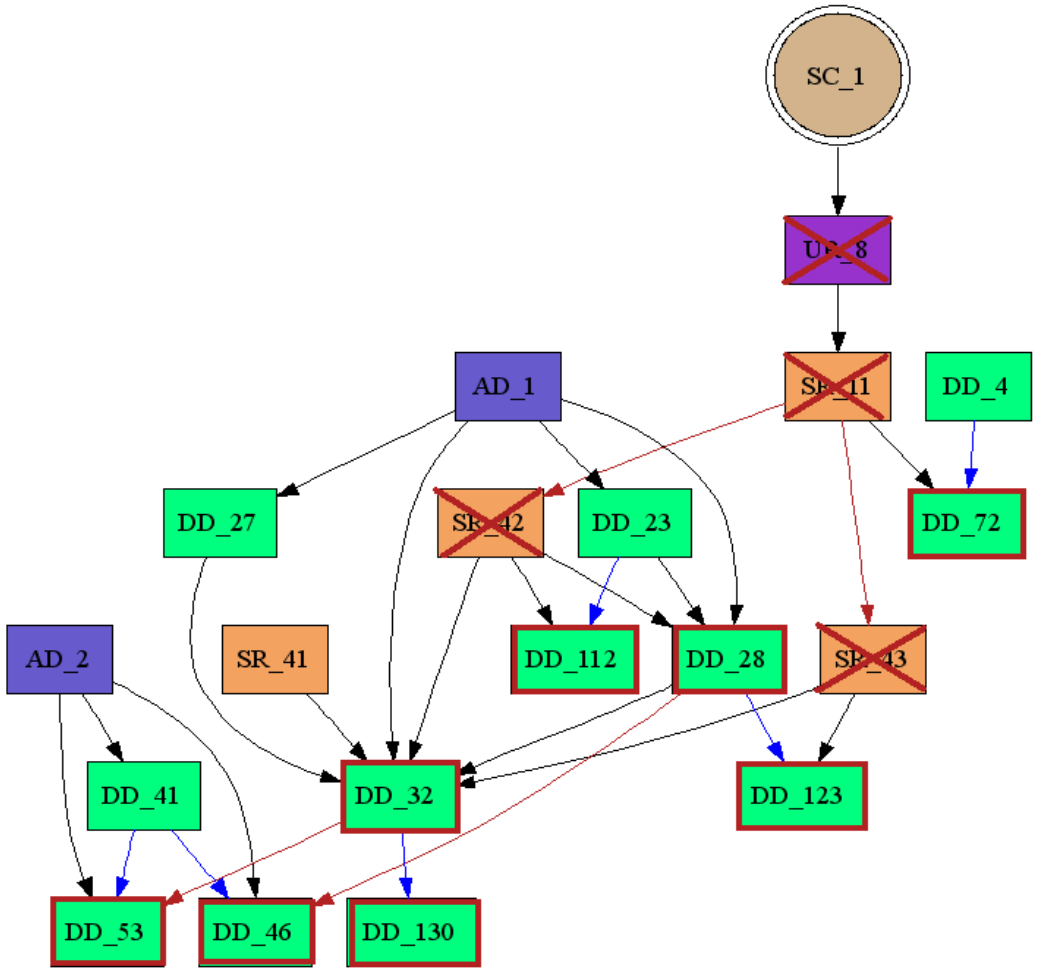
- 333 nodes




SC_1
UR_8
AD_1
SR_11
DD_4
DD_27
SR_42
DD_23
DD_72
AD_2
SR_41
DD_112
DD_28
SR_43
DD_41
DD_32
DD_123
DD_53
DD_46
DD_130

Application of the numerical assignment rules

- How many ?
  - *12*
- How much ?
  - *3,6%*
- Which ones ?
  - *red ones*



Conclusion

- Knowing exactly which elements are influenced directly or indirectly by the maintenance activity under investigation may enable this measurement to extend the life expectancy of the software
  - by allowing the software engineer to do clean maintenance that does not leave behind unused code, one of the main causes of artificial software aging

- Nodes represent information that is not of the same nature and the same size
  - If we join COSMIC-FFP, LOC, etc. value to nodes
    - The obtained measurement value is more accurate
    - A good estimated figure will be easier to achieve
    - Better maintenance management

Thank you!

stephane.mercier.5@ens.etsmtl.ca
aabran@ele.etsmtl.ca
mlavoie@ele.etsmtl.ca
rchampagne@ele.etsmtl.ca
http://www.etsmtl.ca