Towards an Ontology of Software

Nicola Guarino
ISTC-CNR Laboratory for Applied Ontology
Trento, Italy

Joint work with Xiaowei Wang, Giancarlo Guizzardi, and John Mylopoulos

A motivating example

- Microsoft Word celebrated its 30th anniversary last year.
- Of course, *it* changed a lot in these years.
- Still, we say it is the same software. Why?
- Indeed, software changes all the time! (And we know very well how *costly* software changes are!)
- To address problems caused by software change, we need to understand what software change is.
- Only by providing the ***identity conditions*** of software, we can start to answer certain questions about software change in a formal way.

Main goals

An ontology of software

- Clarify software related concepts in a requirements engineering framework

Software as a bridge between abstract and concrete

An ontology-driven software configuration management system

- Provide a a solid semantics for ***software change rationale***

Different software notions

- **General notion** [Osterweil, 08]

  Software is something ***non-physical*** and ***intangible*** used to ***manage*** and ***control*** tangible entities (e.g. recipes).

- **Specific notion of *computer software***

  Four kinds of entities are discussed in the literature:

1) **code**, a set of computer instructions;
2) **copy**, physical *embodiment* of a set of instructions;
3) ***medium***, a physical body which manifests the embodiment
4) ***process***, the result of processing a software copy by executing its code.

L. J. Osterweil, "What is software?," Autom. Softw. Eng., vol. 15, no. 3–4, pp. 261–273, 2008.

Software as Artifact

- Irmak [2013] states that software is an ***abstract artifact constituted*** by code, but different from code (and also different from copy, medium, or process).
- ***Constitution relation*** [Baker 2004]: when a certain aggregate of things exhibits an emerging essential property, a new entity (co-located with the previous one) comes into being
  - e.g, a statue is constituted by a lump of clay
- What is the emerging essential properties of artefacts?
  - having a ***proper function*** ascribed, as a result of an ***intentional process***
  - note: the artefact is *not* required to perform its proper function

N. Irmak, "Software is an Abstract Artifact," Grazer Philos. Stud., vol. 86, no. 1, pp. 55–72, 2013.
Baker, L.R.: The ontology of artifacts. Philos. Explor. 7, 99–111 (2004)

Our contribution

- We also treat software as an ***artifact***, but attempt to answer Irmak's open question "what are the *identity conditions* for software?"

- We ground our answer in the practice of software engineering (and specifically, requirements engineering)
- As a result, we distinguish different kinds of software artifacts on the basis of their different identity conditions:
  - ***Software product***
  - ***Software application***
  - ***Software system***
  - ***Software program***

Code vs Program

- A piece of ***code*** need not be an ***artifact***
  (think of a monkey, randomly pressing a keyboard)

- A ***program must*** *be an* ***artifact***
  We need to have a ***purpose*** for it.
  (i.e., at least a ***functional specification***)

What is a Bug

We can *NOT* say a code has a bug, as long as it is accepted by a computer. The computer just loyally parses the code and executes the instructions.

We *CAN* say a program has a bug, as the execution result of the program could be something other than its specification.

What is a Bug

- *Program1*: print the value of variable a
- *Code1*: Int a=0, b=1; print b;
- *Code2*: Int a=0, b=1; print a;

- **Both codes are correct for he computer.**
- **For the human**, the program is buggy when it is constituted by *Code1*, and it becomes correct when *Code 1* is substituted by *Code 2*.

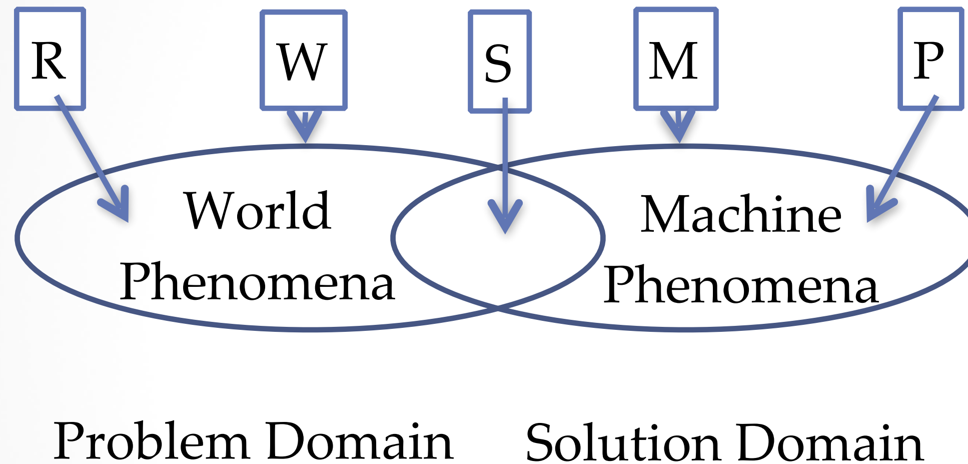
From Ontological Analysis to Software Engineering

- It is human *intention* that makes a program an artifact different from code; a program is an artifact *constituted* by code.

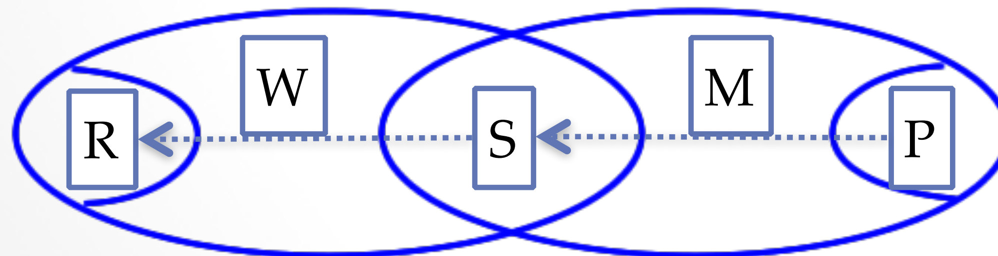
- Capturing the *intentions* in software artifacts requires looking at *Software Engineering (SE),* and particularly *Requirement Engineering (RE)*.

- So we answer identity questions coming from formal ontology by looking at *SE* practice.

Jackson and Zave's Theory: Different Kinds of Intentions in SE



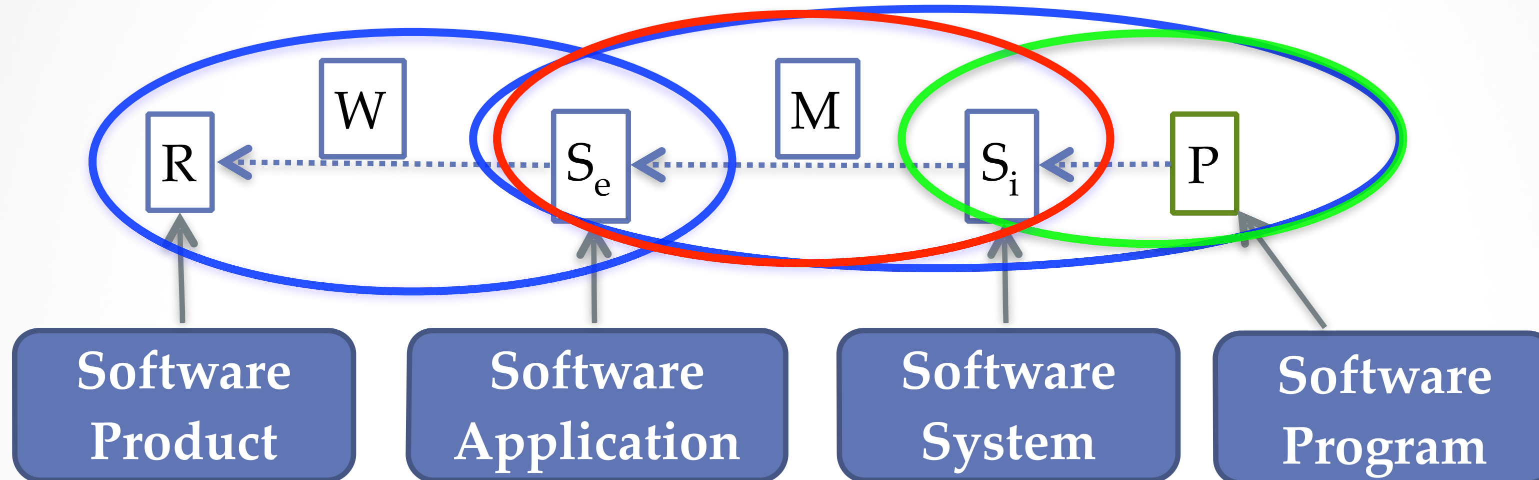
R: Requirements
W: World assumption
S: Specification
M: Programming platform
P: Program



W, S |=R
M, P |=S

C. A. Gunter, M. Jackson, and P. Zave, "A reference model for requirements and specifications," Software, IEEE, vol. 17, pp. 37–43, 2000

Jackson and Zave's Theory: Different Kinds of Intentions in SE



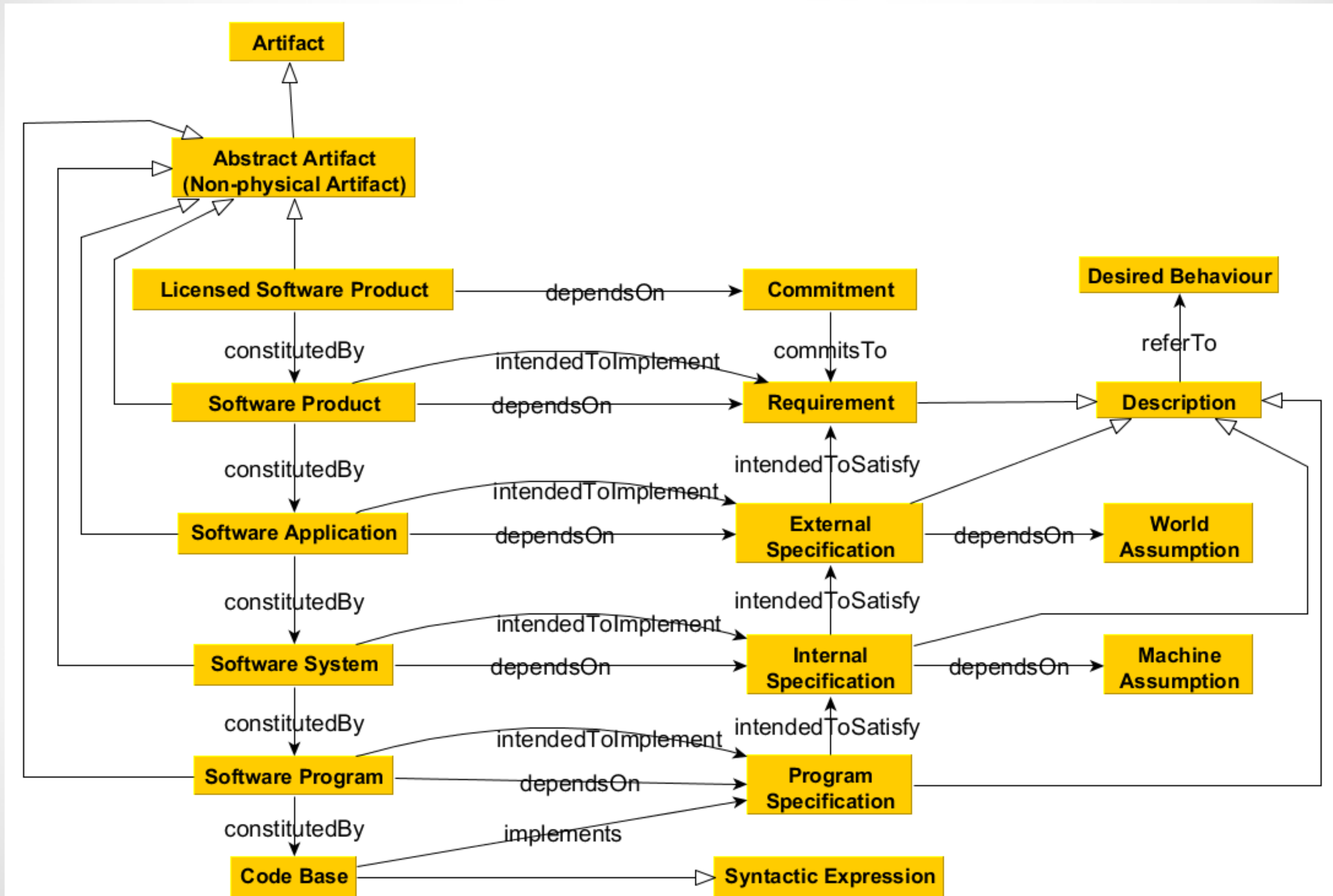
R: Requirement
W: World assumption
M: Machine assumption
S_e: External Specification
S_i : Internal Specification
P: Program Specification

$W, S_e \models R$

$M, S_i \models S_e$

$P \models S_I$

A Preliminary Ontology of Software



HOW TO DO WHAT TO DO

The Specific Categories

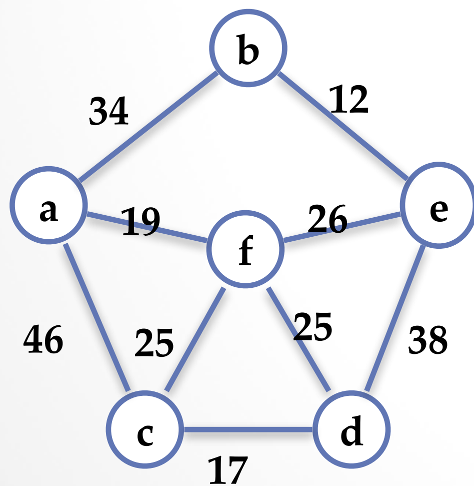
Code Base

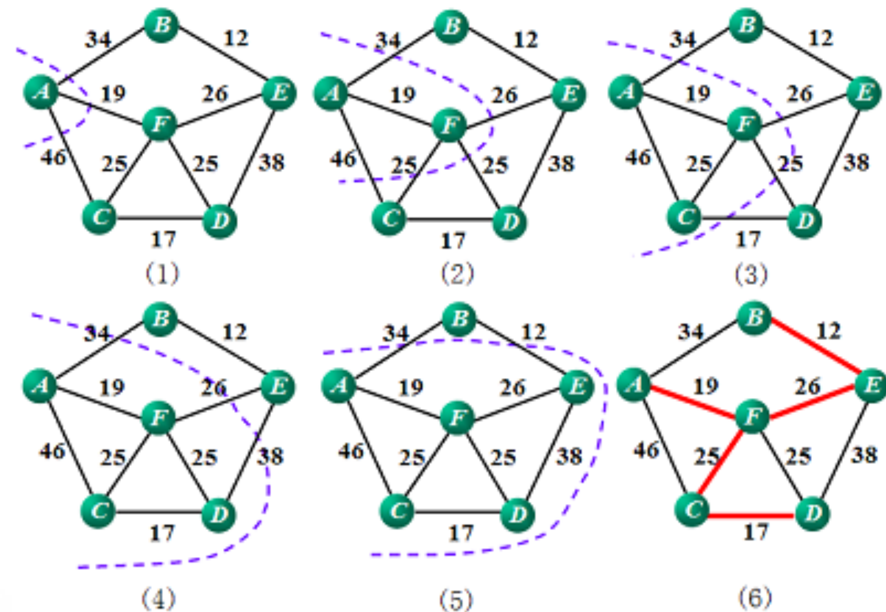
- **Nature:** Sequence of instructions
- **Identity criterion:** Syntactic Expression(a well-formed sequence of instructions in a Turing-complete language).

- Two code bases are identical iff they are syntactically the same.
- New code bases are created from changes including *variable renaming*, *order changes* in declarative definitions, inclusion and deletion of *comments*, etc.

Software Program

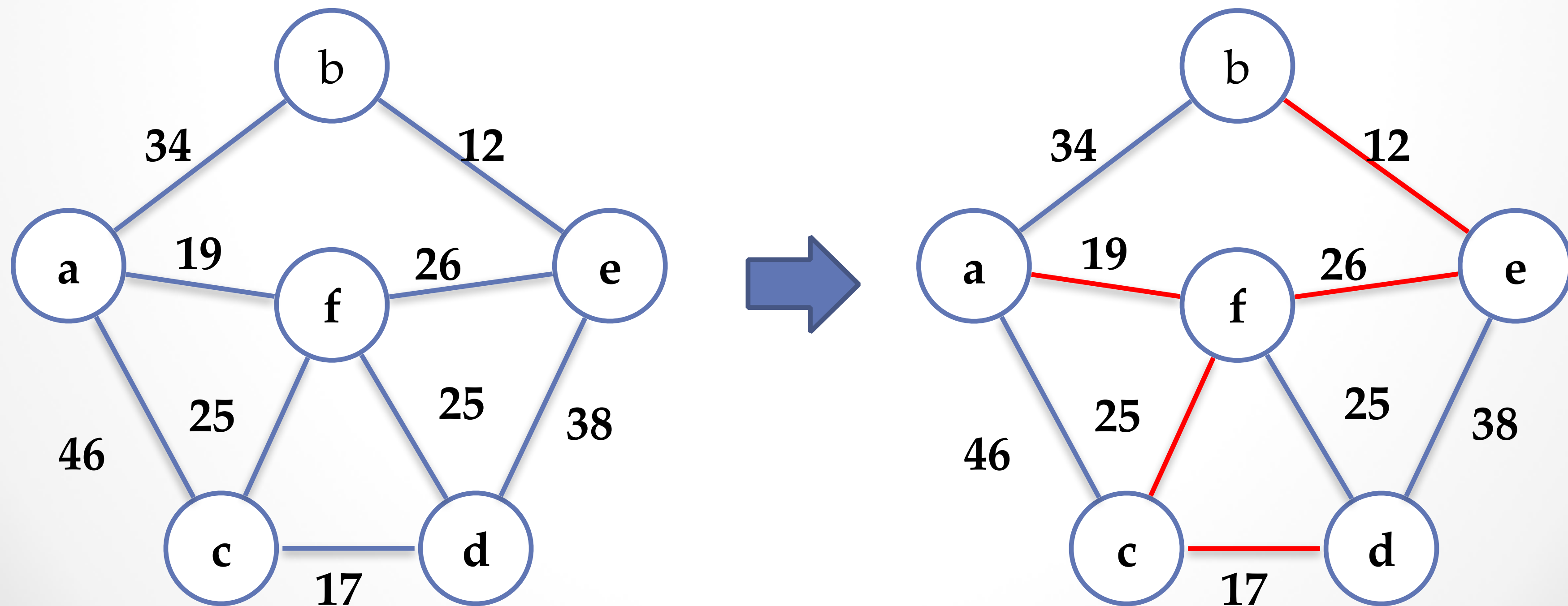
- **Nature:** Artifact constituted by a code base
- **ID condition:** Specified data structure, functional change in data structure, and algorithm *inside the computer* (Program Specification)
- **Example:** Minimum Spanning Tree (MST-Prim)





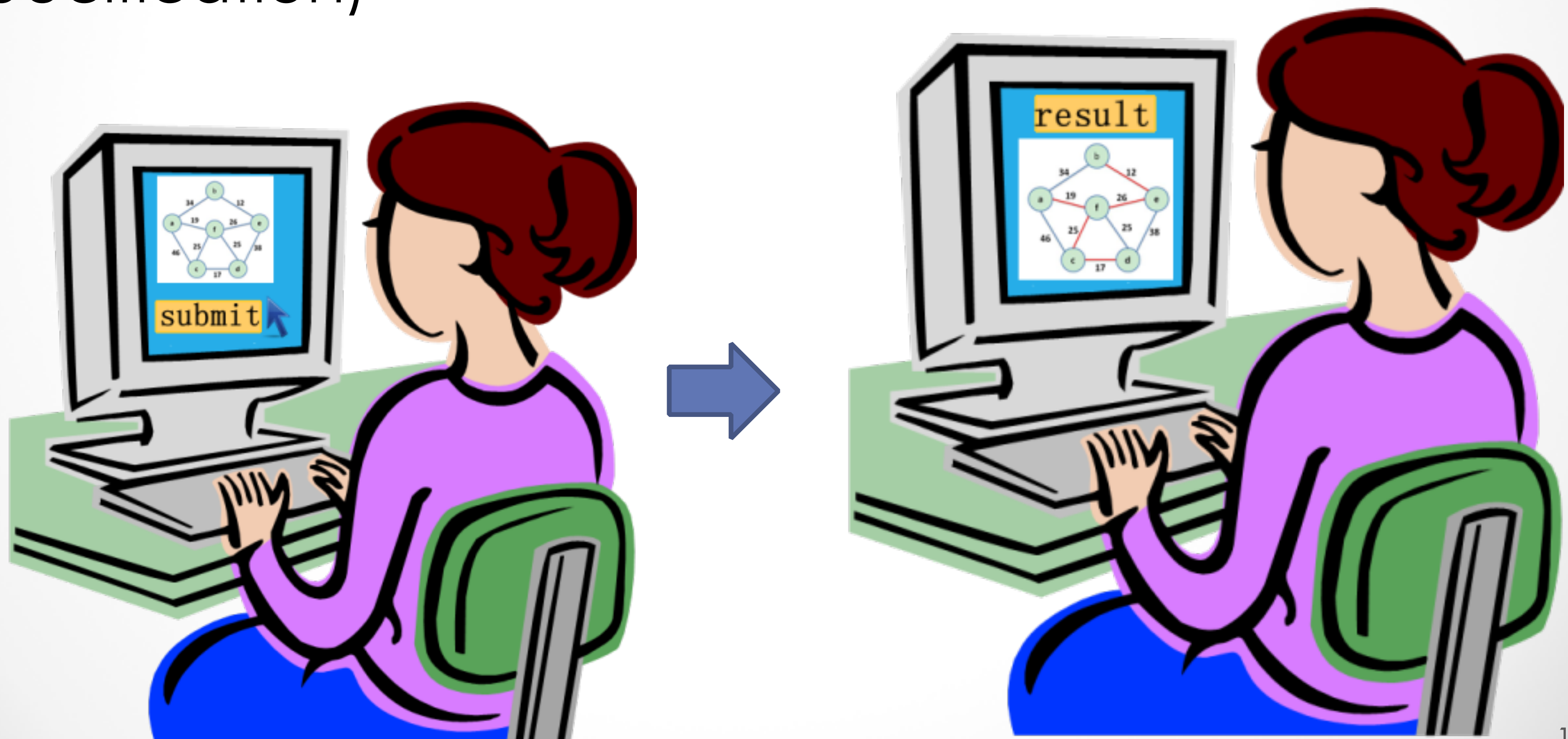
Software System

- **Nature:** Artifact constituted by software program
- **ID condition:** Specified functional changes in data structure *inside the computer* (Internal Specification)
- **Example:** Minimum Spanning Tree



Software Application

- **Nature:** Artifact constituted by a software system
- **ID condition:** specified behavioral constraints *at the interface* with the environment (external specification)

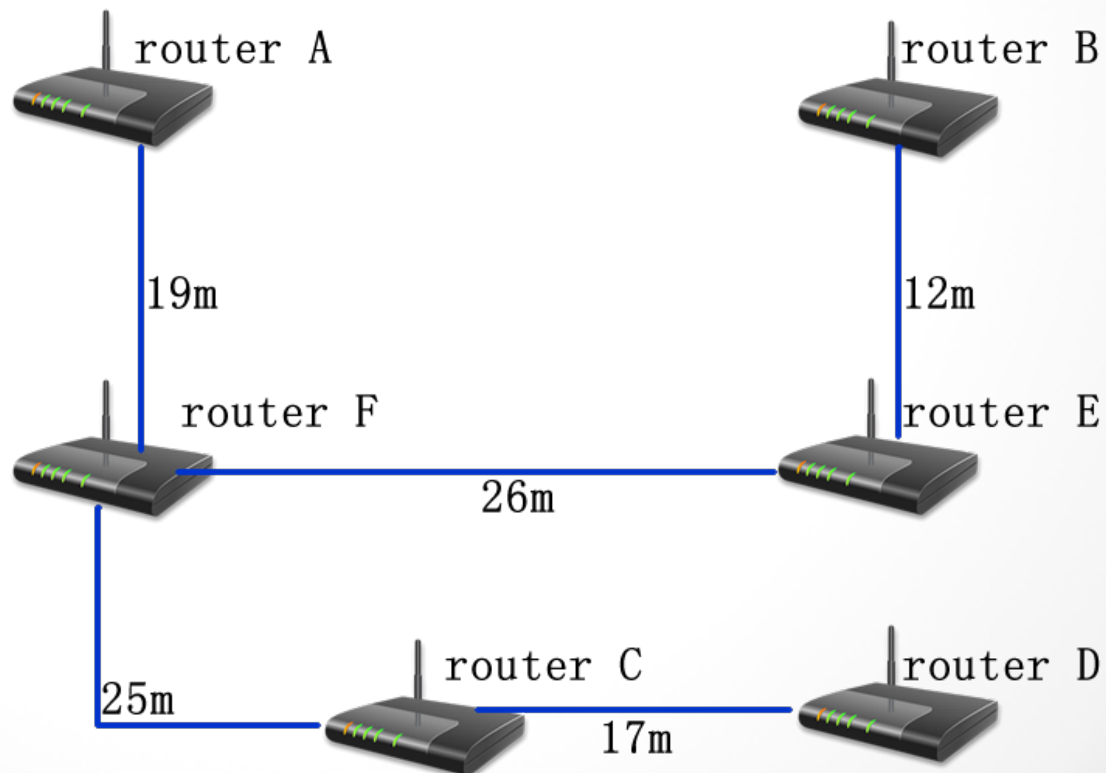


Software Product

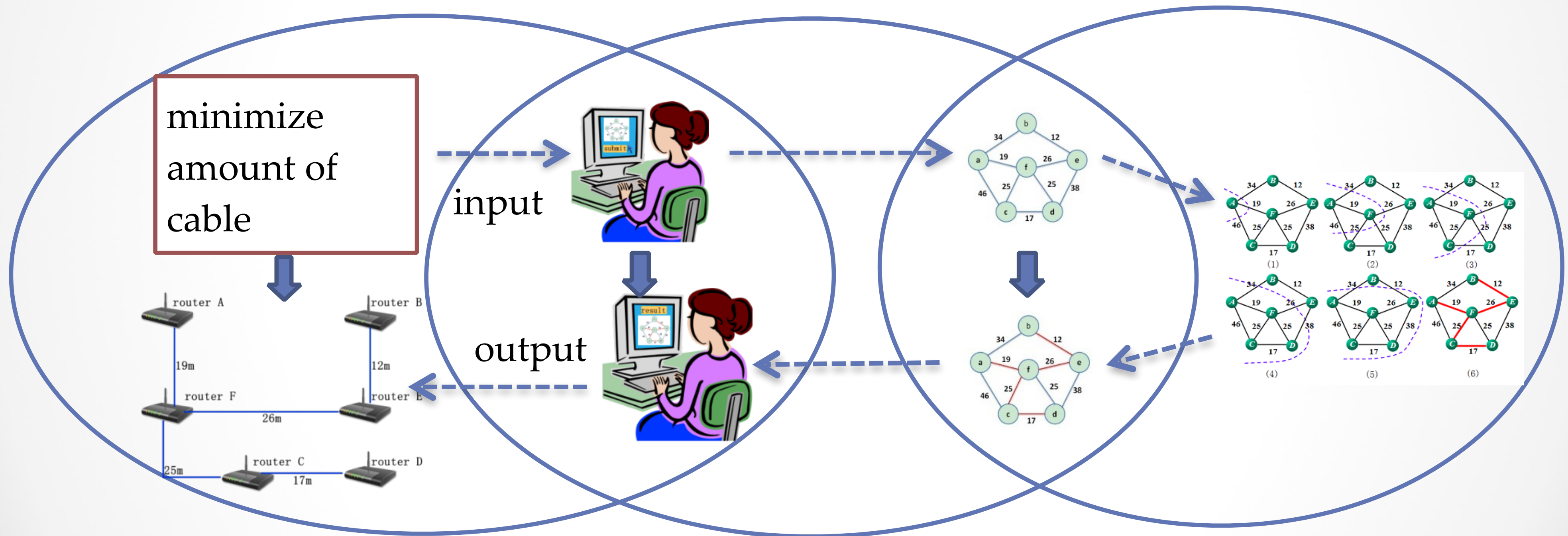
- **Nature:** Artifact constituted by a software application
- **ID condition:** specified (or just desired) behavioral constraints *in the external environment* (high level requirements)

Example:

determine the most economic way to connect a set of routers (minimizing cable length)



Software as a Bridge between Abstract and Concrete



The social dimension: new kinds of software emerging

Software products usually come to the market in the form of ***service offerings*.**

- A *service* is a *social commitment* [Ferrario&Guarino 2009].
- *Service offerings* are *meta-commitments*, which are commitments to engage in specific commitments once a contract is signed (e.g. the delivery of certain services).

- Before the contract is signed we have another software entity emerging: a ***Licensable Software Product***.
- After the contract is signed, we have a ***Licensed Software Product***.

Towards ontology-driven software configuration management

Software Configuration Management

Dart [1991]: ***Software Configuration Management*** is "*a discipline for controlling the evolution of software systems*", and two basic notions about *version* are explain through our ontology.

- **Revision Process**
  - *From*: *Program p1* constituted by *Code Base c1* at time *t*
  - *To* : *Program p1* constituted by *Code Base c2* at time *t'*
- **Variant Process**
  - *From*: *Software system s1* constituted by *Program p1* at time *t*
  - *To 1)* : *Software system s1* constituted by *Program p1* at time *t'*
  - *To 2)* : *Software system s2* constituted by *Program p2* at time *t'*

Dart, S.: Concepts in Configuration Management Systems. Proceedings of the 3rd International Workshop on Software Configuration Management. pp. 1–18. ACM, New York, NY, USA (1991)

Accounting for Software Change Rationale

- The ontological distinctions above help to understand where (and why, more or less) software changes occur.

- These changes can be reflected in an ontology-driven versioning system.

  e.g. v 1.5.3.2 :

  - 1 - software application release number;
  - 5 – software system release number;
  - 3 – software program release number;
  - 2 – code release number.

Accounting for Software Change Rationale

- The ontology-driven versioning system above provides the possibility of developing new software ***versioning control tools*** describing software changes with a solid semantics.

- Traditional tools only focus on code changes, but according to our work, software could be ***consistently expressed and tracked at multiple abstraction layers*** (e.g. code, program, software system, software application, software product).

Accounting for Software Change Rationale

Refactoring refers to the creations of new codes, keeping the identity of the program;

Re-engineering refers to the creations of new programs, keeping the identity of the software system;

Software evolution refers to the creations of new software systems, keeping the identity of the software (product).

Conclusions

- We provided a preliminary ontology of software that establishes a link between a ***formal ontology of artifacts*** and the ***practice of software engineering***.
- Such ontology has layered structure based on the ***constitution*** relation.
- We are planning to exploit this results of a new generation ***software configuration management*** system.