Software as a Social Artifact

A Requirements Engineering Perspective

Speaker: Xiaowei Wang

Supervisor: John Mylopoulos, Nicola Guarino

University of Trento, Italy
ISTC-CNR, Italy
xwang@disi.unitn.it

Background and Motivation

- Software is used in every aspect of human activity
- Ambiguous terminology is used in the SE community
- Software is difficult to maintain and has a high failure rate the in the practice
- We try to answer the questions, what is a software, and how to identify the same software under several changes

State of the Art

- Software as a general concept [Osterweil, 08]

He characterizes software as something non-physical and intangible, a software instance could be executed to manage and control tangible entities

- (Computer) Software in common sense

Four concepts are ambiguous in the literature:

1) *code*, the instruction for a computer;
2) *copy*, the realization of instructions through hard media;
3) *media*, the hardware media itself;
4) *execution*, the execution process of the copy.

Understanding of Software

- Duality [Moor, 78] [Colburn, 00]

They believe that software is both the code with the "copy + media" (e.g. CD with code) or the code with the execution (running process or thread in a computer) at the same time

- Distinguishing the entities [Oberle, 2005]

SoftwareAsCode: an InformationObject, the expression with both syntax and semantics;

ComputationalObject: the physical realization of such code in a concrete hardware, but not the hardware;

ComputationalActivity: the result of the execution of a computational object.

Software as Artifact

- Software *is-A* (*Expression* and *Artifact*) [Kassel, 07]

  A program is considered as both a computer language expression and an artifact of computation

- Software/program as Artifact different from code, copy, media and execution [Irmak, 13]

  1) We share very much Irmak's intuitions;

  2) questions are left open by them:

  how *software changes*,

  what *identity conditions* for software are, and more.

Code vs Program

- A code could be an non-artifact

  e.g. the code could be the result of the input of a monkey randomly pressing the buttons

- A program must be an artifact

  e.g. the program must be created under human intention, such as a program created for sorting a list of numbers

What is a Bug

- To understand the different between code and program, we can check the meaning of 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 instruction.

We *CAN* say a program has a bug, as the execution result of the program is not intended by human.

What is a Bug

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

The computer doesn't know which piece of code is intended by the human.

We would like to say that *Program 1* is a buggy program when it is constituted by *Code1*, and Program 1 is correctted when *Code 1* is substituted by *Code 2*.

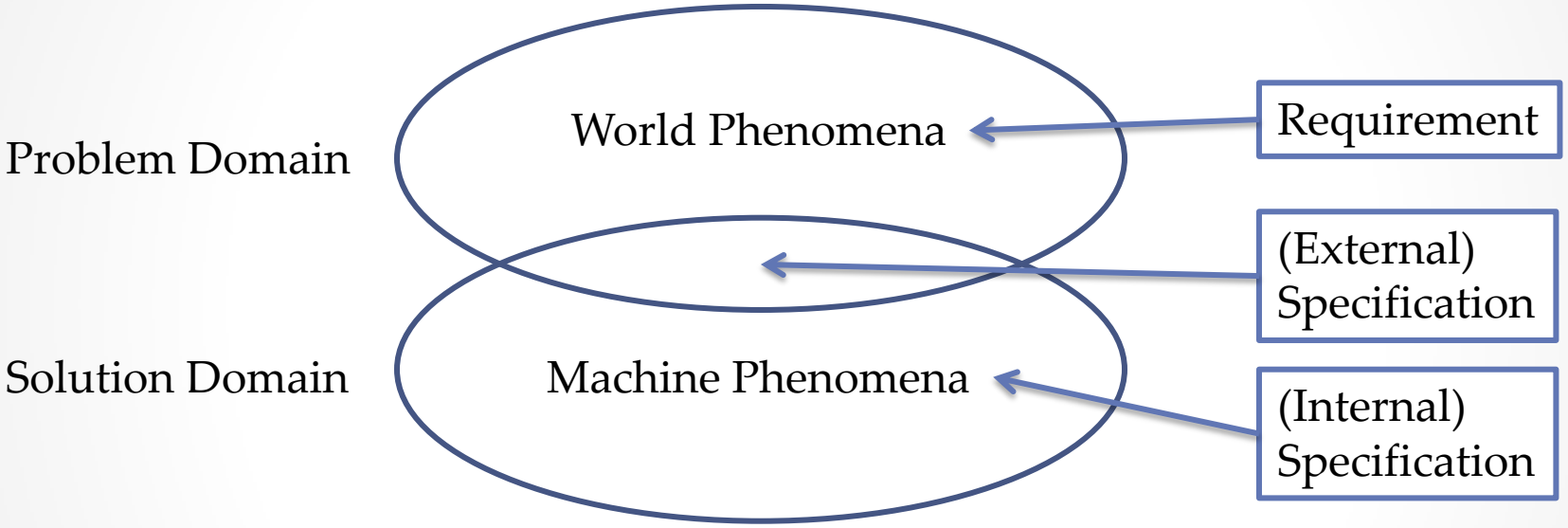
(**note:** detail artifact creation process is neglected here)

The Intention

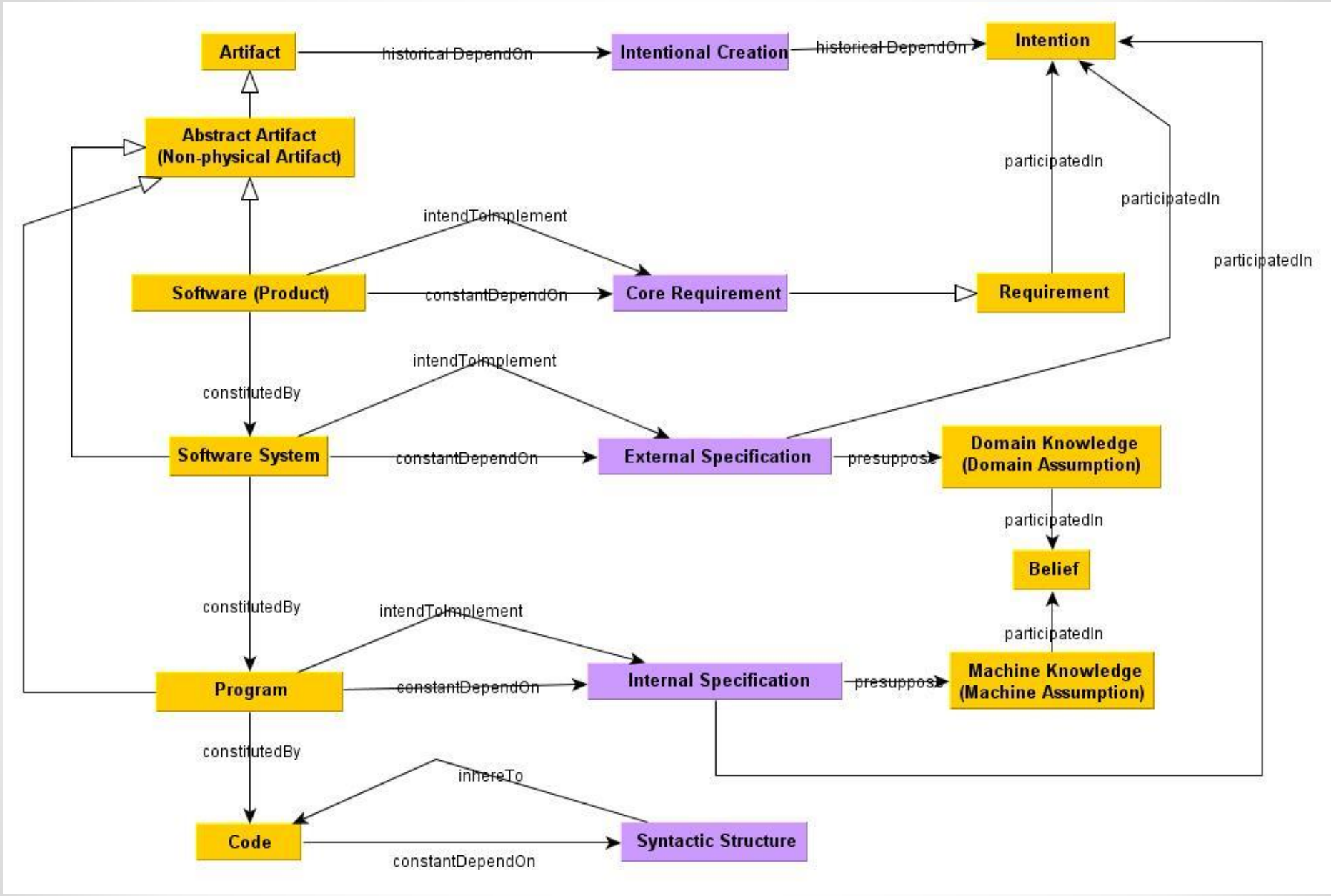
- It is the *intention* makes the program as an artifact different from a code, and Irmak stops here, leaving the identity of software/program as an open question.

- To answer this question, we dive into the concept of intention based on *Zave and Jackson's* proposal about *requirement* and *specification* in view of *world* and *machine*.

Zave&Jackson's Theory



This is a general view about the World&Machine, when the machine is a computer-driven machine, we can start to discuss about computer software.


Artifact
historical DependOn
Intentional Creation
historical DependOn
Intention
Abstract Artifact
(Non-physical Artifact)
participatedIn
participatedIn
participatedIn
intendToImplement
Software (Product)
constantDependOn
Core Requirement
Requirement
intendToImplement
constitutedBy
Software System
constantDependOn
External Specification
presupposes
Domain Knowledge
(Domain Assumption)
participatedIn
Belief
constitutedBy
intendToImplement
participatedIn
Program
constantDependOn
Internal Specification
presupposes
Machine Knowledge
(Machine Assumption)
constitutedBy
inhereTo
Code
constantDependOn
Syntactic Structure

Code

- **Description:** Sequence of instructions, expression according to a programming language.
- **ID criteria:** Syntactic Structure
- **Explain:**

Two codes are identical if and only if they exactly have the same syntax.

New codes created from the changes including variable renaming, order changes in declarative definitions, inclusion and deletion of comments, etc.

- (**notes:** Relations among the code, copy, media and execution were discussed in previous slides)

Program

- **Description:** Artifact constituted by code
- **ID criteria:** intentional creation, internal specification
- **Explain:**

By checking the meaning of a bug, we have already known that the internal specification as the content of the intention identifies a program, focusing on the phenomena inside a machine, whose constituent could be different codes.

Another example shows the program historically depends on the intentional creation: different programs with same code and same internal specification developed by Microsoft (MS) or by a individual student. As two creation events raised.

Software System

- **Description:** Artifact constituted by program
- **ID criteria:** intentional creation, external specification
- **Explain:**

The software system historically depends on an intentional creation, but the constituent is program, so the constitution relation is chained together.

By pointing to different kinds of specifications, we move our abstraction layer up to another level that the software system doesn't care about the internal machine states, only focuses on the interactions between the World and Machine.

Software (Product)

- **Description:** Artifact constituted by software system
- **ID criteria:** intentional creation, core requirement, commitment, social structure
- **Explain:**

Following the constitution chain, by assuming a software (product) is an artifact constituted by software systems (indirectly constituted by code), we move the abstraction layer of the concept to the top level, focusing on world phenomena.

Besides the identity criteria of the intentional creation, the core requirements of a software system are essential which can't be changed for keeping the identity of the software.

Commitment

- Besides the intentional creation and core requirement, we still need to consider the constant dependence to a social commitment to the core requirements.

- e.g. Skype doesn't change after it is purchased by MS, as MS inherits the commitment to ensure the core requirement is fulfilled. To the user of Skype, there is no different before and after this purchase.

Social Artifact

- In other words, a software system generally depends on a social structure which consists of several social roles.
- For example, a software could be developed by single agent and used by the same agent, which meaning the intentional creation, requirement, and the commitment only refer to a single agent.
- But by bring the social structure concept, we can derive that there are always a developer role and user role for the software, although it is developed and used by the same person.

Practical usage

- A simple usage of our work could be a refined terminology for different kinds of software changes:

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

2) *re-engineering* refers to the creations of new programs, keeping the identity of the software system;

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

Practical usage

- If we propose a versioning number criterion according to our software abstraction layers, the significance level could be determined

- e.g. v 1.2.3.4:

  1 - requirement number;

  2 - external specification number,

  3 - internal specification number,

  4 - code number.

Practical usage

- As another practical result of this approach, the refined versioning number method provides the possibility of developing new software *versioning control tools* with *high level semantics* describing software changes.
- Traditional tools only focus on the changes in the codes, but according to our work, software could be *consistently expressed and tracked in multiple abstraction layers* (e.g. code, program, software system, software product).

Conclusion

- Provided an answer to the question "what is the identity criteria of software"
- We treated a software as a social artifact
- Several concepts are clarified in a core ontology of software: code, program and software system and software product
- These concepts were organized into a consistent abstraction layer structure

Future work

- An ontology of software expressed in OWL or other description logics with full axioms and thermos
- An ontology of software evolution
- Tools and methodology for software maintenance

The end

Thanks!

QUESTIONS?