Session J-17

Test Case Extraction and Test Data Generation from Design Models

Xiaojing ZHANG, Takashi HOSHINO

NTT Cyber Space Laboratories
Tokyo, JAPAN

Agenda

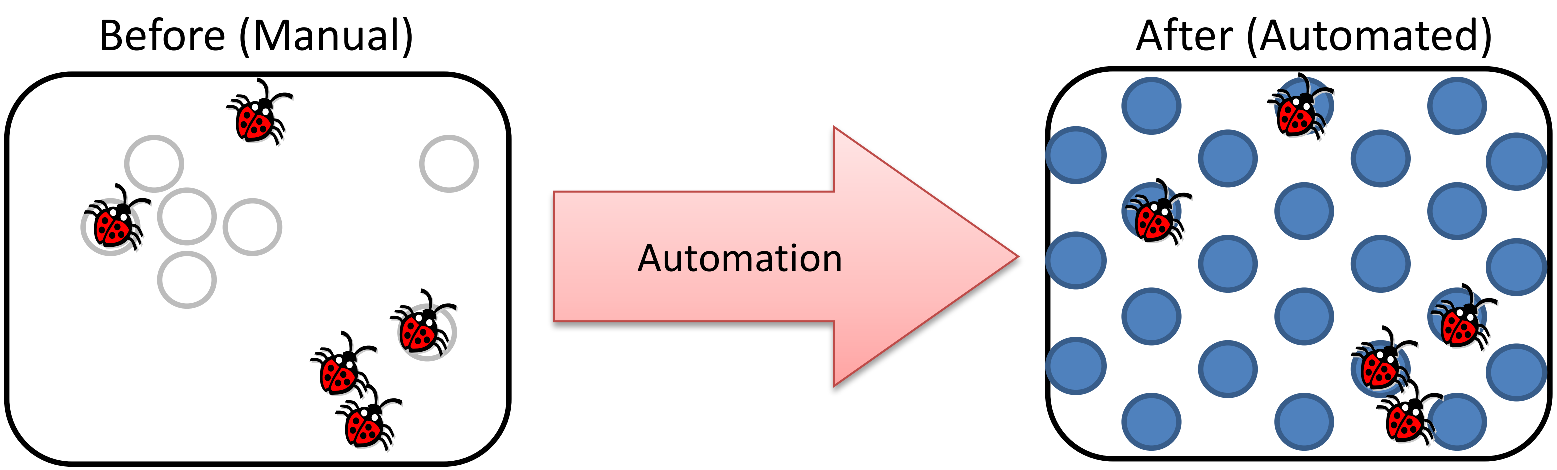
- Background and motivation
- Generation approach
- Tool implementation
- Evaluation results
- Future work and conclusion

Background

- We need better quality assurance!
  - Software defects become a public concern.
- Why testing?
  - The last check on the final product before release.
- What's testing?
  - A confirmation of whether the product is developed just as one intended.
- How to improve testing?
  - Quantity
    - Test density = Number of test cases / Size of the SUT.
  - Quality
    - Structure coverage = Elements tested / Total number of elements.
    - Input space coverage = Inputs used for testing / Entire input space.

Goal in this paper

- **our goal is to...**
  improve "test design", with low cost by automation
  - for both unit test and integrated test
  - Test design:
    - Extracting test cases and test data for test execution
    - Needs huge effort to achieve high density and coverage



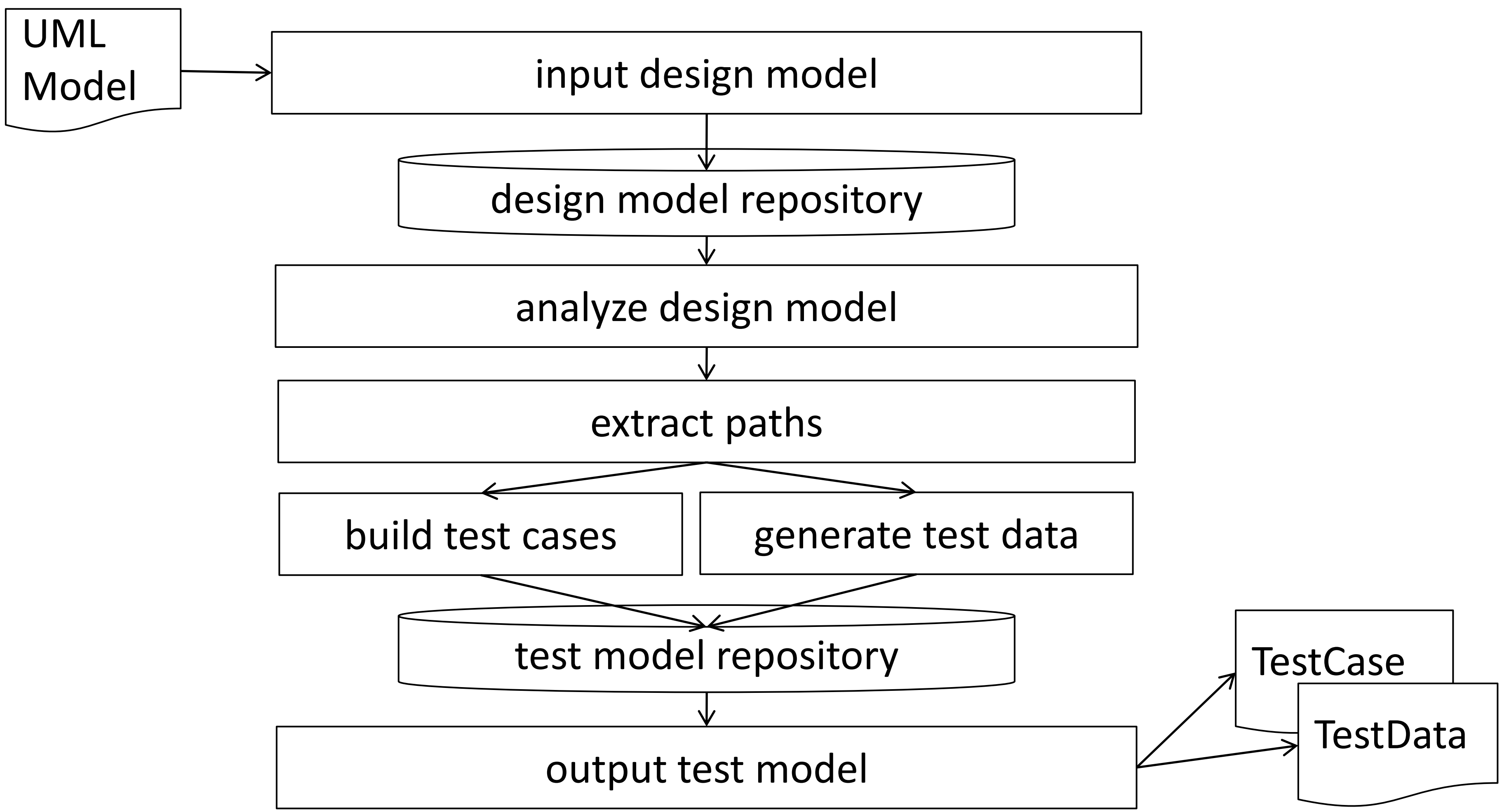
Motivation

- Related literatures' limitations
  - Source code as input. Can we assume it's 100% right?
  - Original design notation. Not welcomed by users...
  - Primitive data type like integer or string
- Our Motivation
  - software design as input
    - The formalized "intentions" of the customers or designers
  - familiar notations
    - UML 2.0 activity for behavior, class for data structure
  - test data with structure
    - More variations have to be considered when the test data has hierarchical or repetition structure

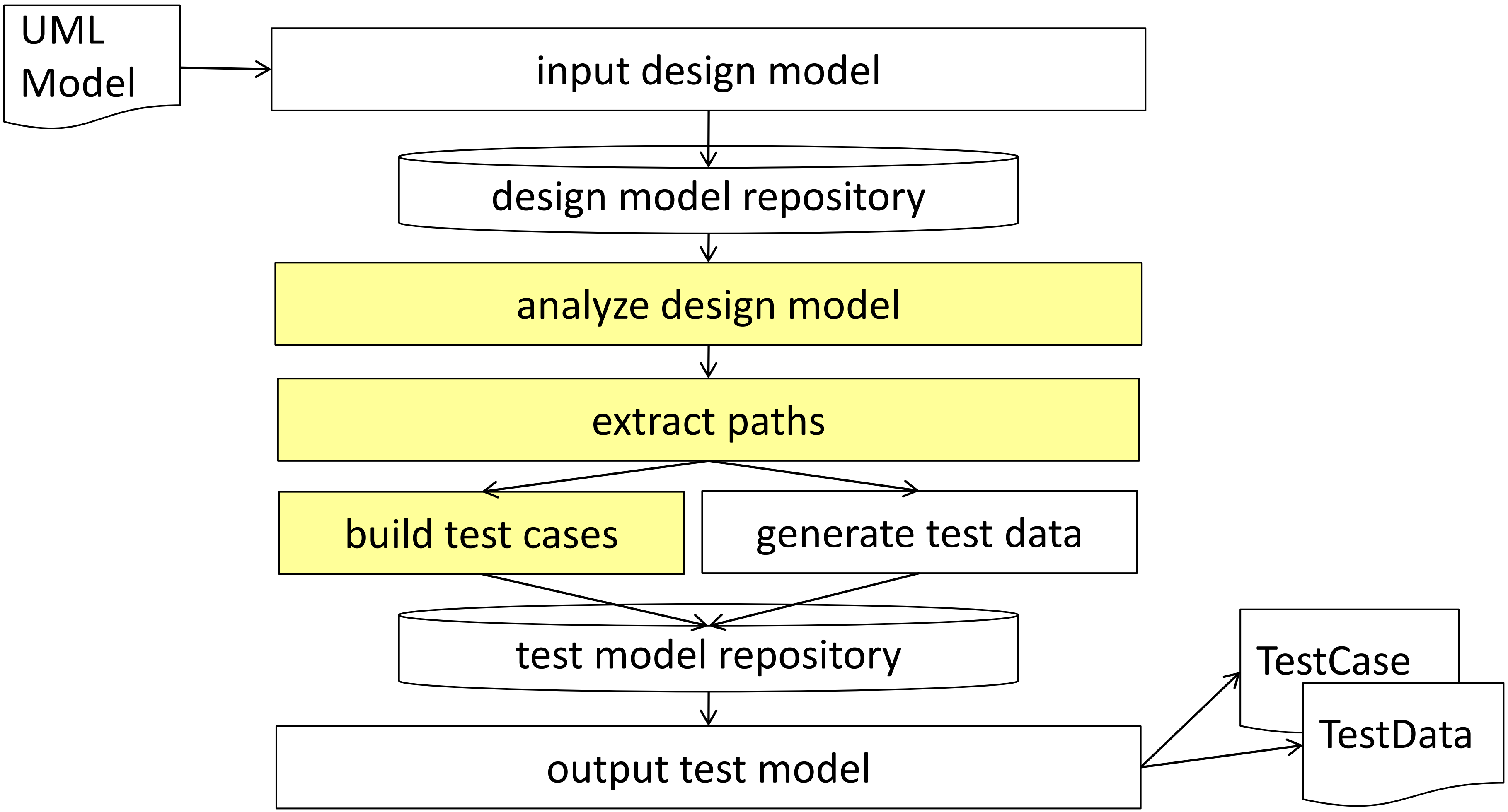
Agenda

- Background and motivation
- Generation approach
- Tool implementation
- Evaluation results
- Future work and conclusion

Generation approach: An Overview



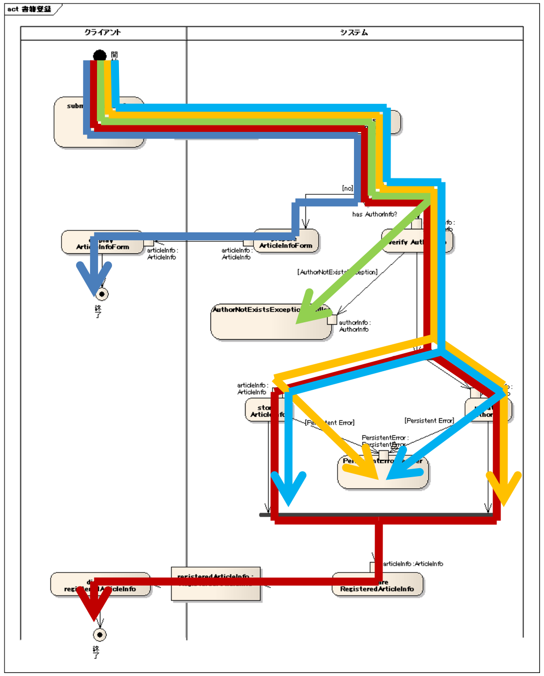
Test Case Generation



Generation approach: Test case (1/2)

1. Decompose UML model into activities

2. Extract “paths” from UML Activity
   - simple depth first search algorithm
   - If a loop exists,
     case of not passing the loop
     case of passing the loop just once



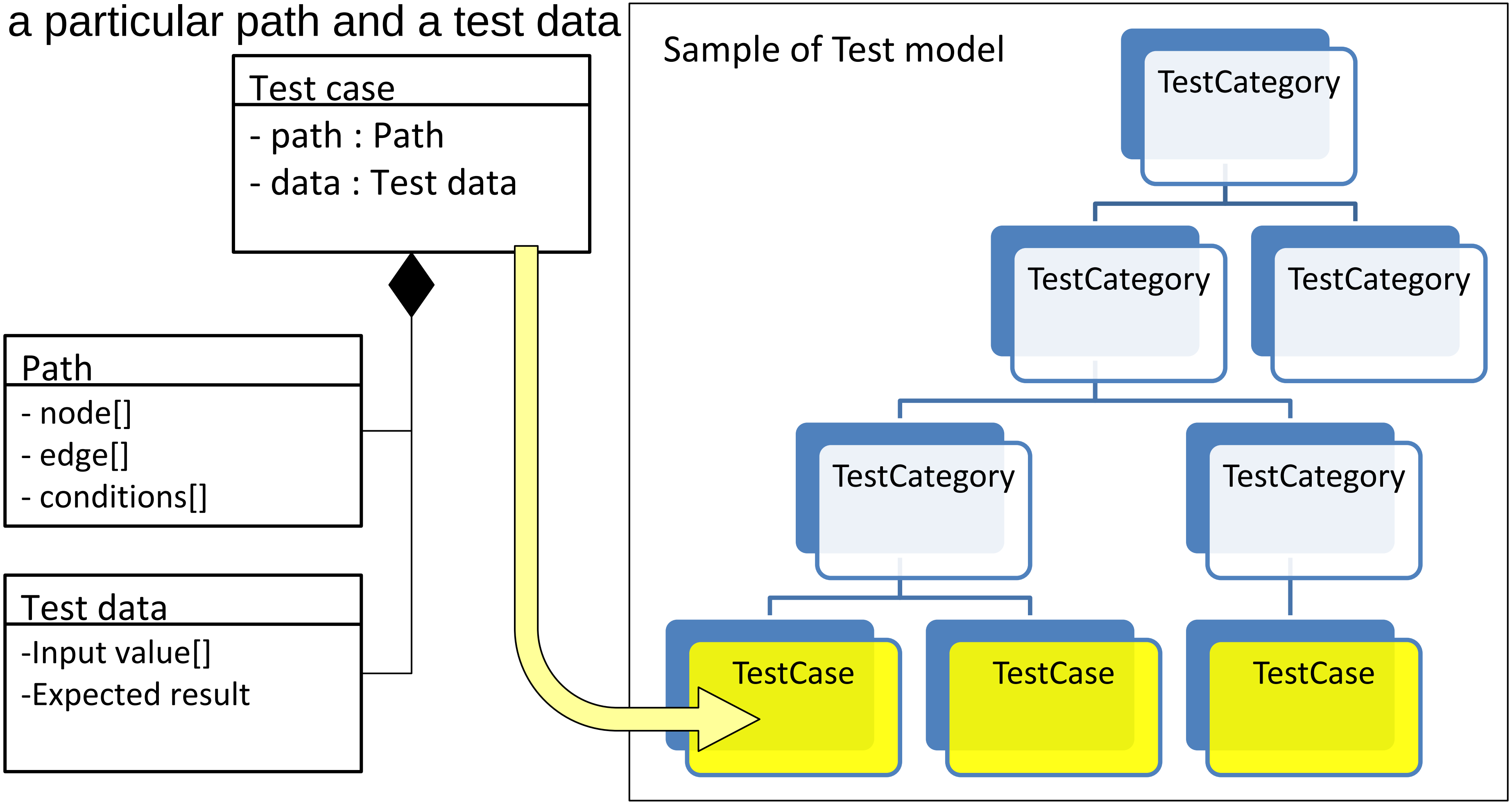
3. Make one test case for each path
   - a test case is a pair of a particular path and the test data

Generation approach: Test case (2/2)

4. Organize test cases as a test model

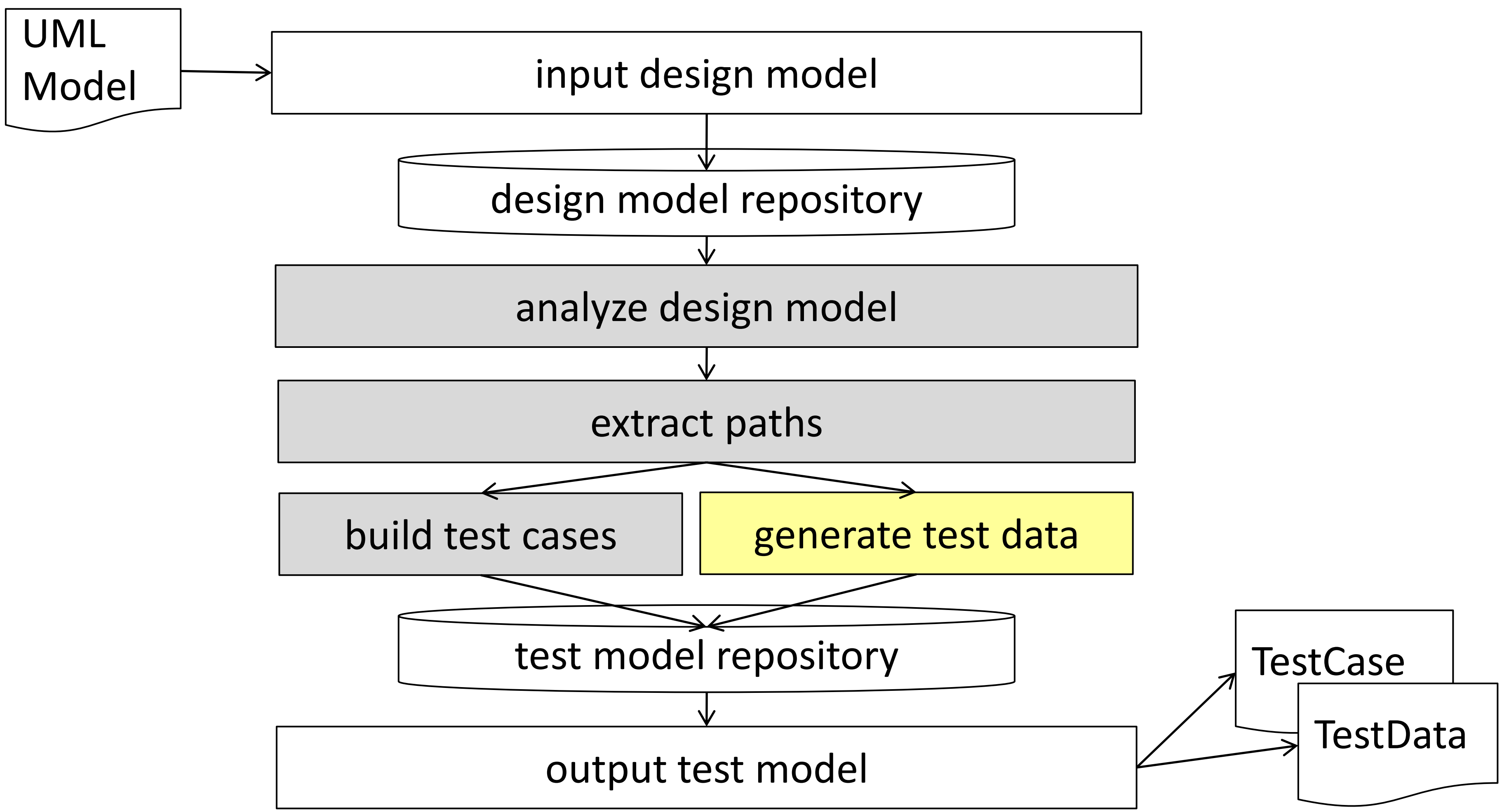
a test case is a pair of
a particular path and a test data

Can be categorized by test level or viewpoints



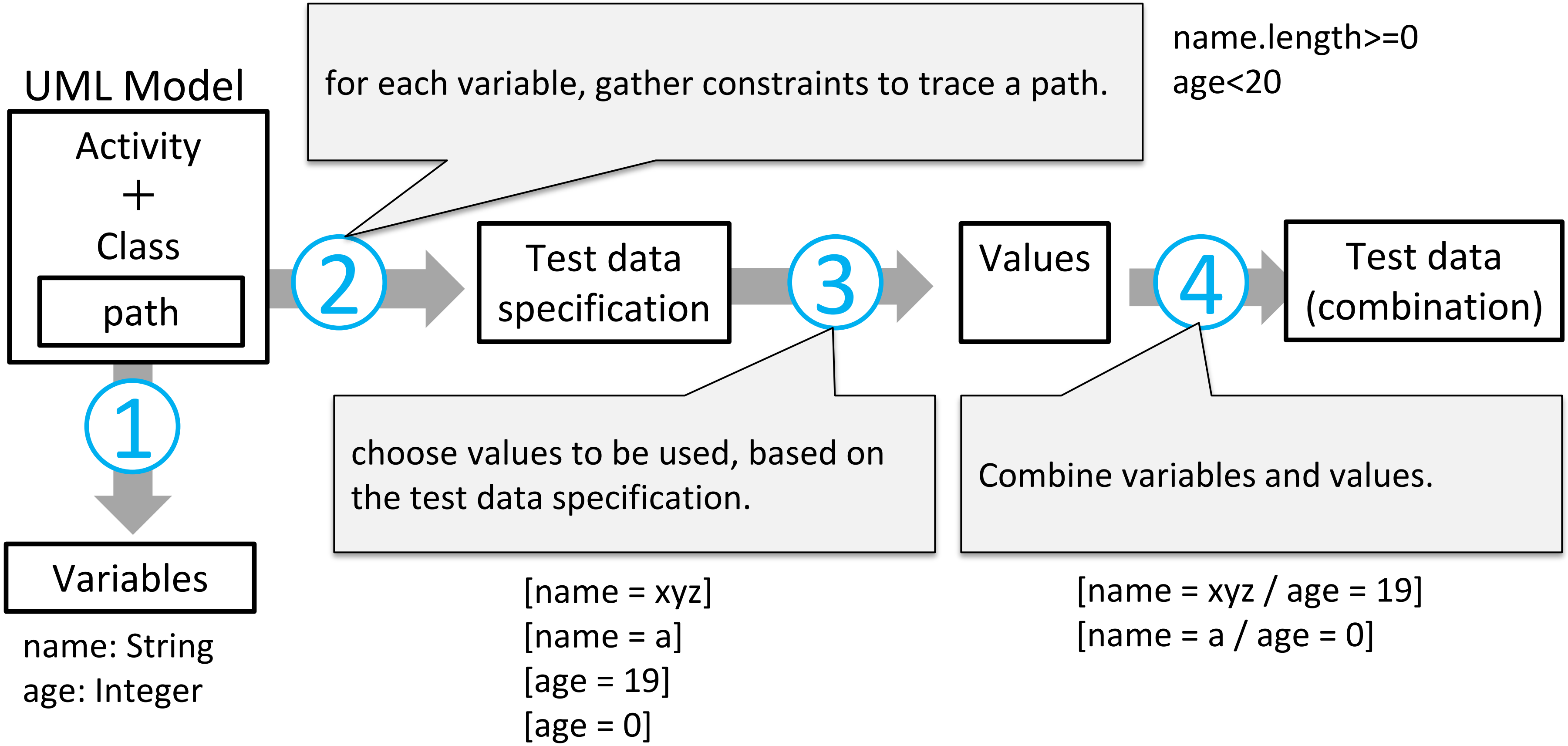
NTT

Test Data Generation



Generation approach: Test data (1/4)

- In order to achieve high “input space coverage”, we thoroughly obtain the <u>variables</u> and the <u>values</u> which constitute the test data.
- The generation process has 4 steps.



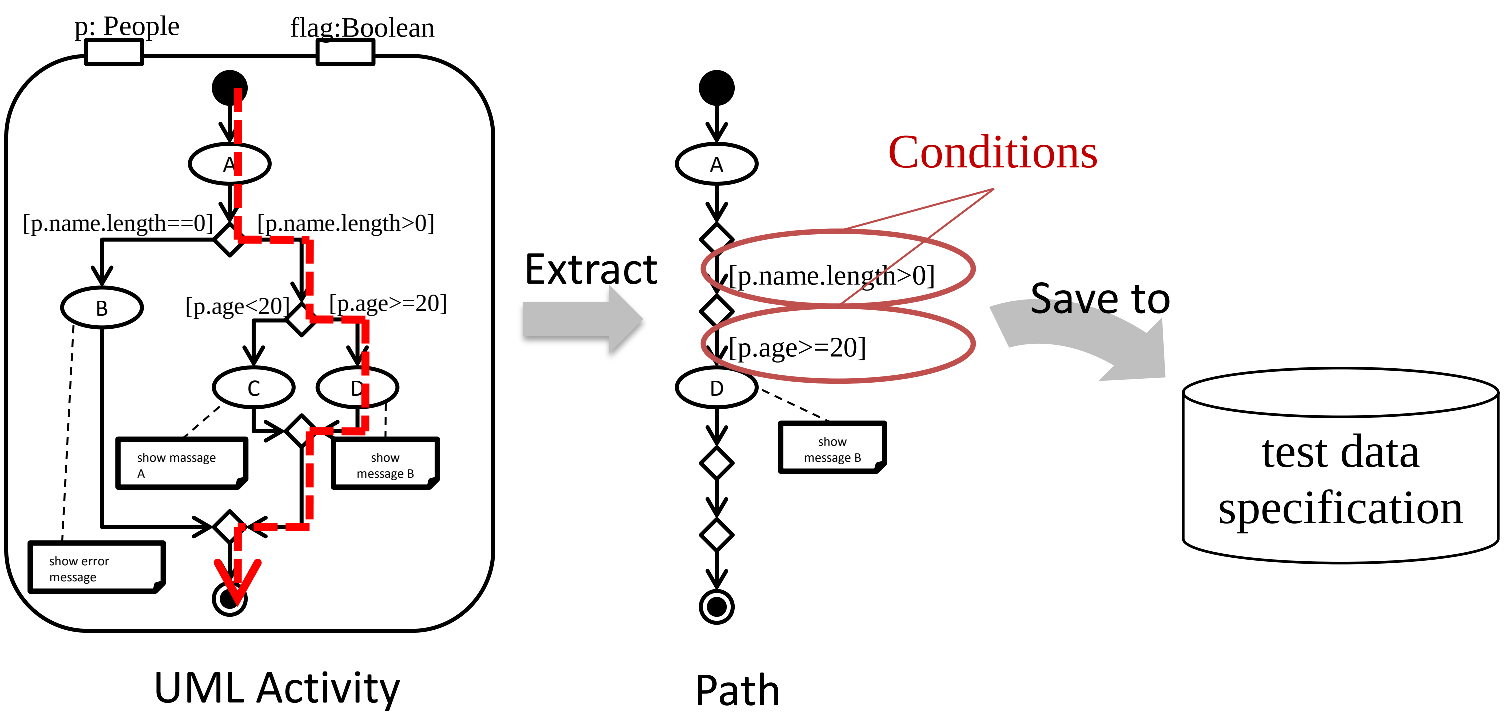
Generation approach: Test data (2/4)

① Extraction of the variables

- activity parameter node

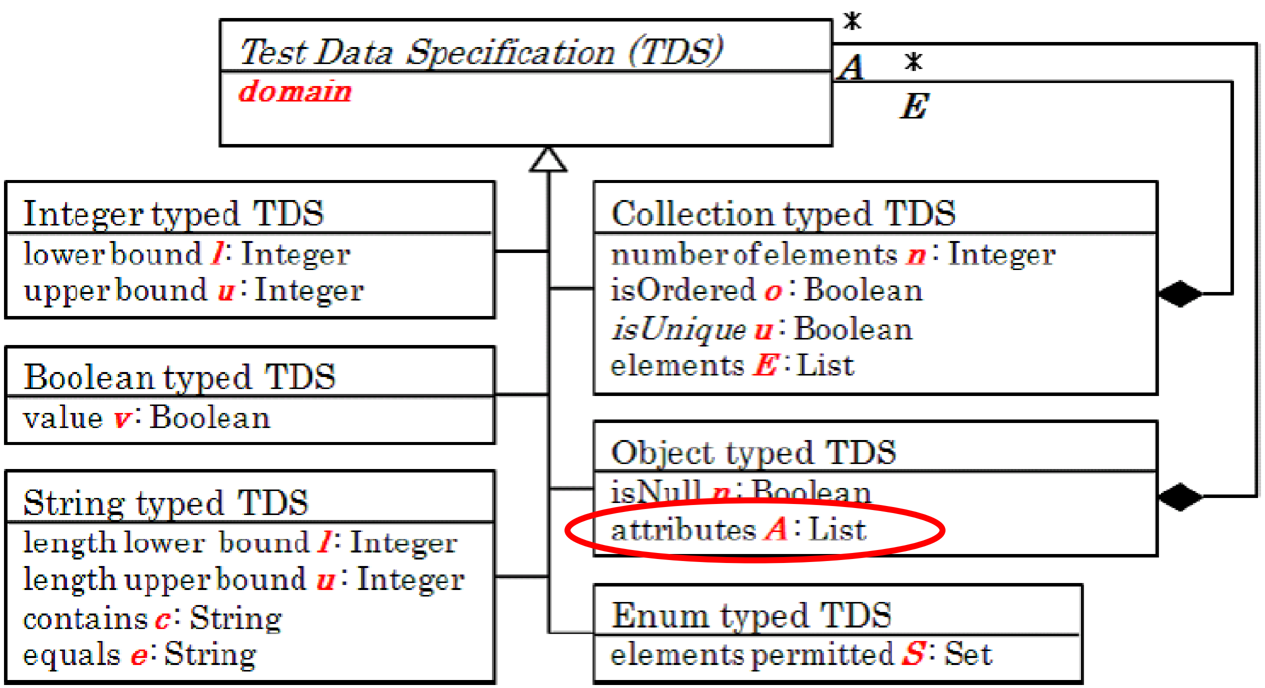
② Creation of the “test data specification”

- which hold the constraints for the variables.



② Generation of the “test data specification”

- Generated per each variable of each path
- Different “domain” for different data type.
- Domain of Object data type, contains specification of its own attributes:deal with hierarchy structure



Generation approach: Test data (4/4)

③ Generation of the values

- boundary value analysis
- normal values / abnormal values
- as much as one "abnormal value" in a "leaf node" means the entire object is an abnormal value.

④ Combination of the values

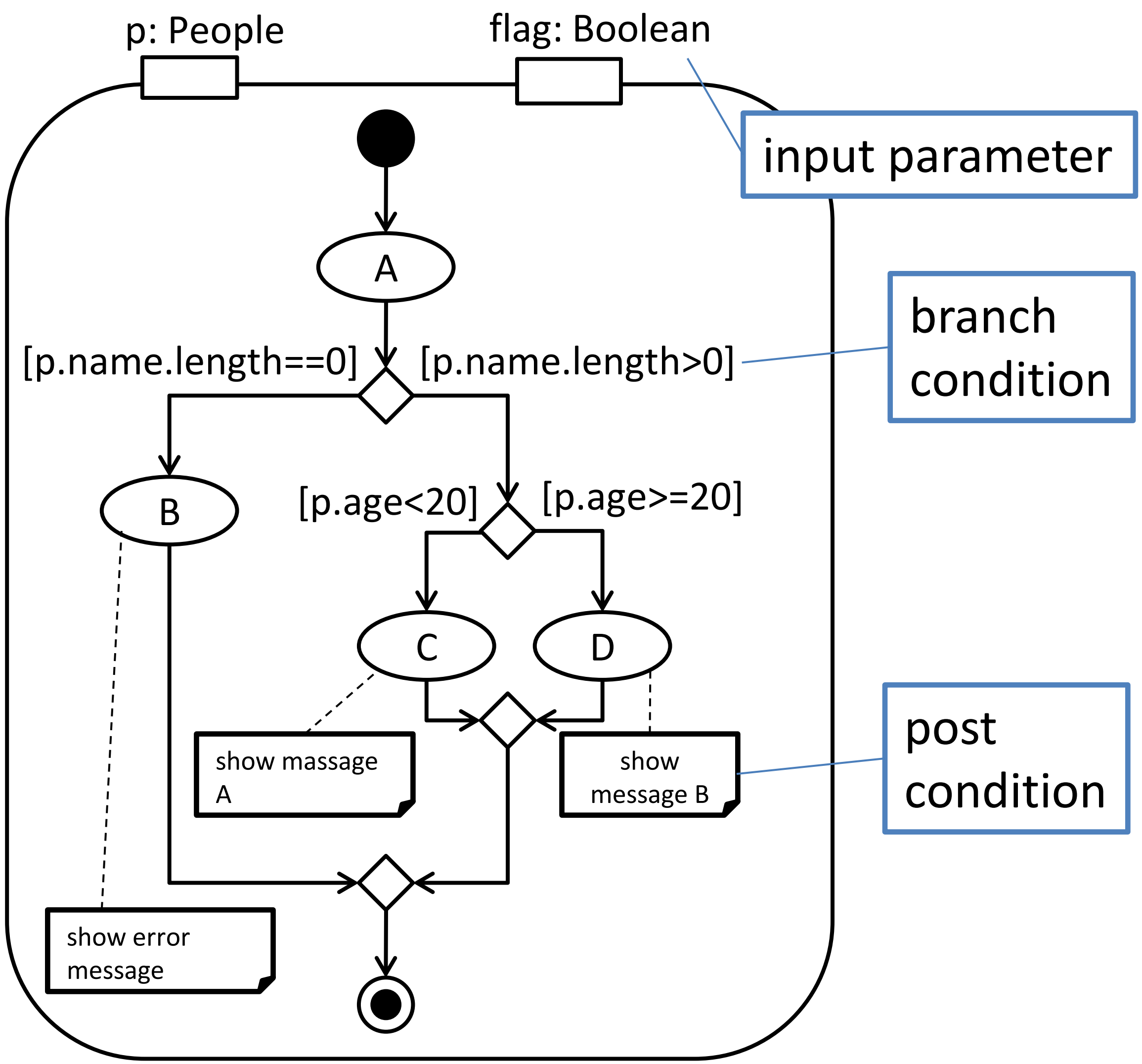
- Normal Inputs
  - Minimum combination cover all normal values
- Abnormal Inputs
  - Combination contains only one abnormal value

Generation approach: Restrictions

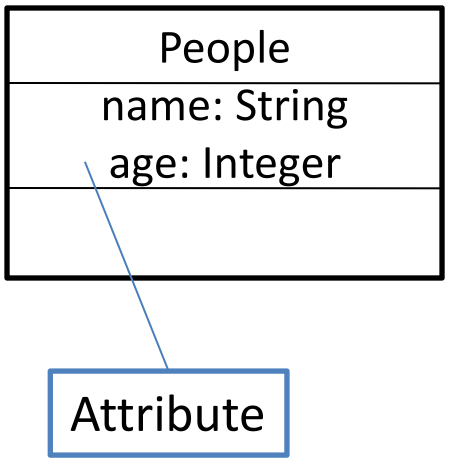
- Cannot handle <u>variables</u> except the input parameters
  - temporary variables, global variables
    - Need more data modeling.
- Cannot handle <u>conditions</u> describing:
  - the dependence between two or more variables.
    - Need dynamic domain reduction?
  - variables overwritten or changed.
    - Need symbolic execution or simulation?
- Inefficient <u>combinations</u>
  - just use all values generated is not good enough.
    - Need pair-wise methods?

Example

UML Activity



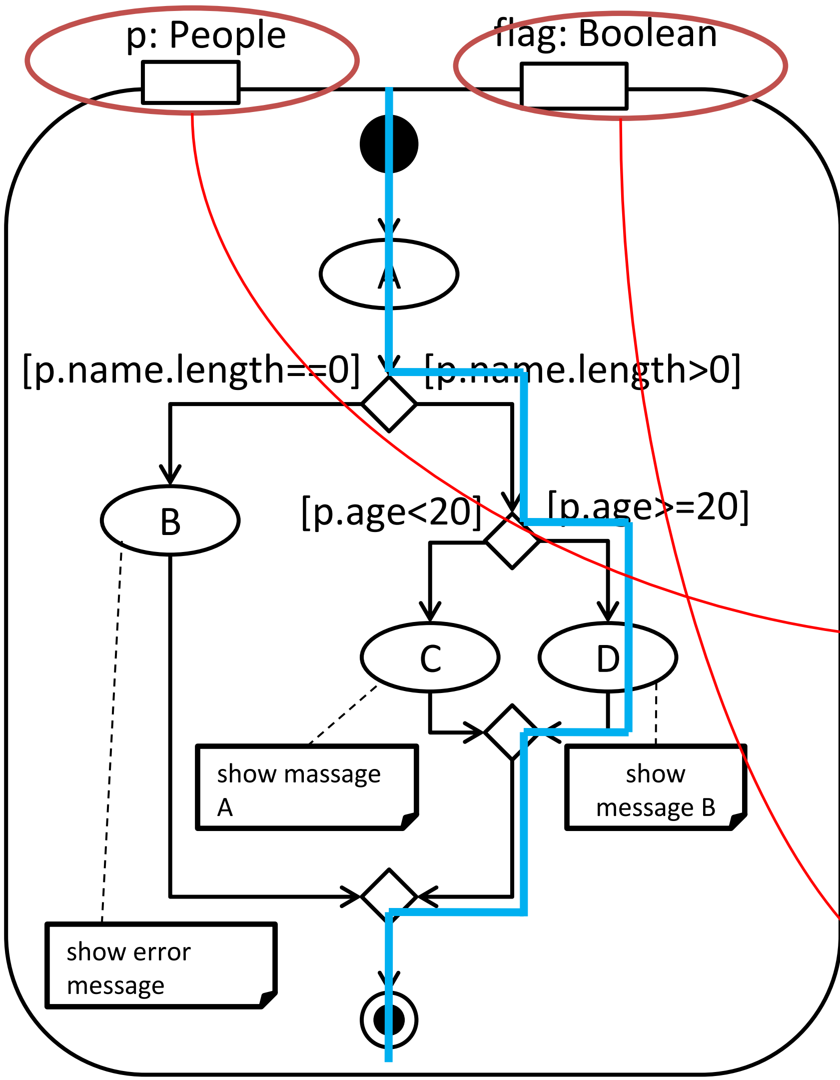
UML Class

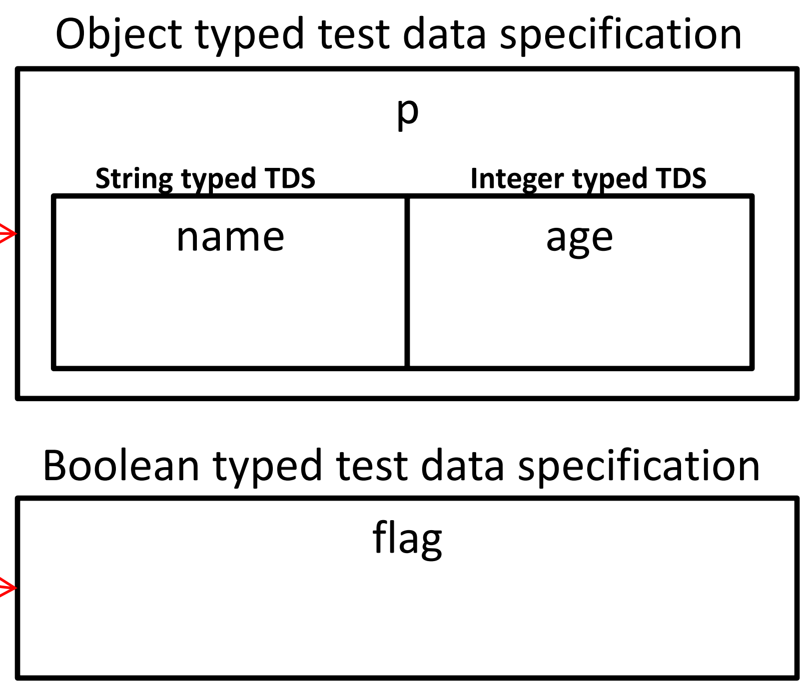


Example

①②

For each path,
Extract Input parameter as variables,
make and initialize a test data specification
depends on the variable's type

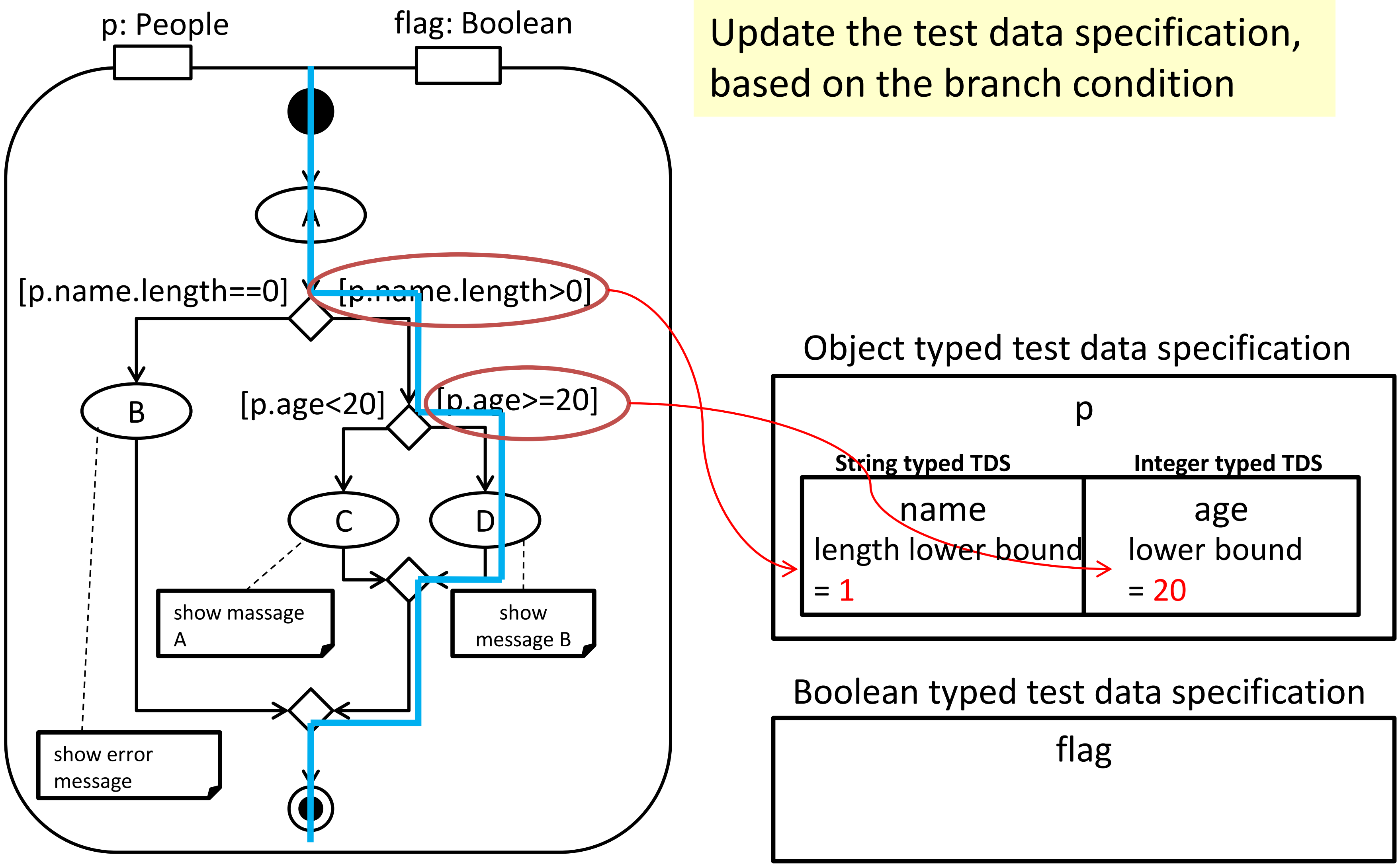




Example

②

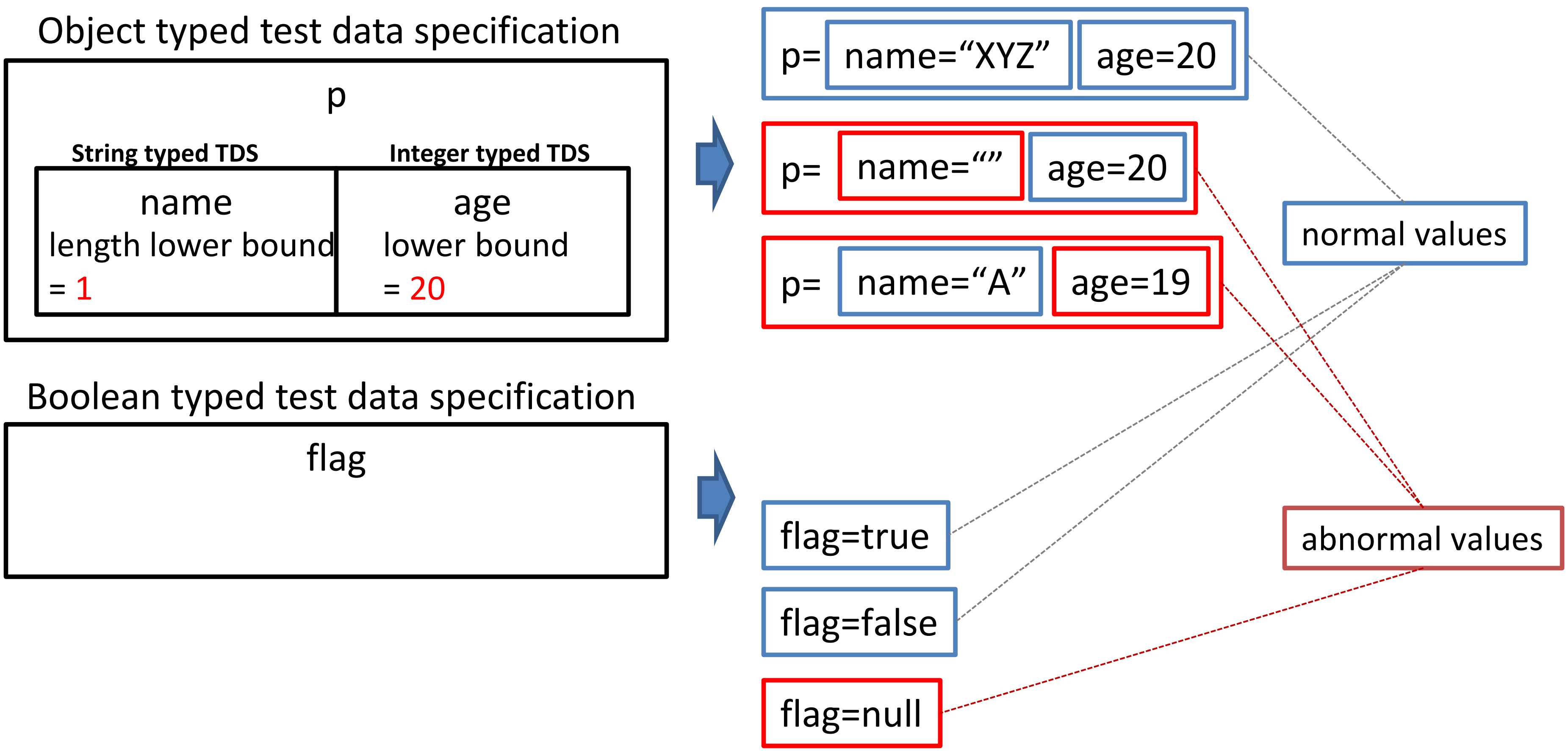
Update the test data specification, based on the branch condition



Example

③

Generate normal values and abnormal values from the test data specification.

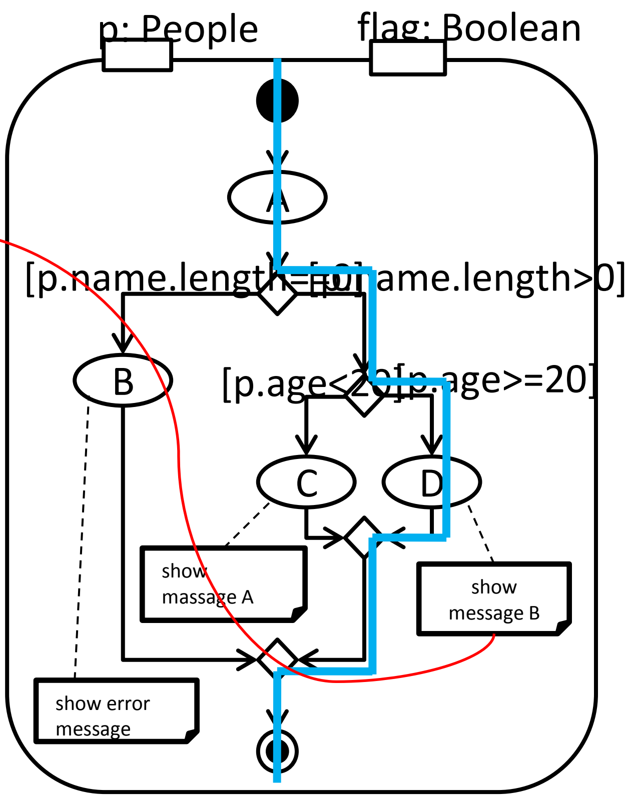


Example

④

input value is a combination of the values set to variables respectively.
expected result from user defined postcondition of the path.

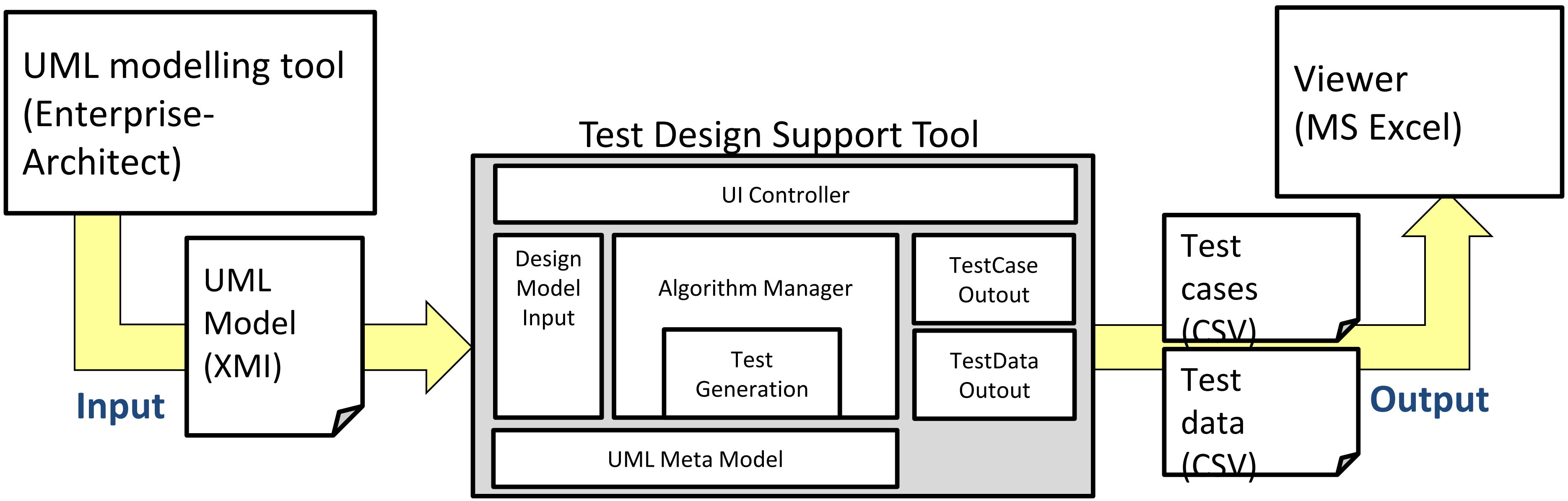
ID	Input value (Combination)	Expected result
1	p= name="XYZ" age=20 flag=true	show message B
2	p= name="XYZ" age=20 flag=false	show message B
3	p= name="" age=20 flag=true	indefinite
4	p= name="XYZ" age=20 flag=null	indefinite
	…	



Agenda

- Background and motivation
- Generation approach
- Tool implementation
- Evaluation results
- Future work and conclusion

Tool implementation



Sample

Test cases

root	EA_Model	DPC	Components	LineItem	setQuantityShipped	PathNo0	test_setQuantityShipped_PathNo0_DataNo0
							test_setQuantityShipped_PathNo0_DataNo1
							test_setQuantityShipped_PathNo0_DataNo2
					setLocale	PathNo0	test_setLocale_PathNo0_DataNo0
							test_setLocale_PathNo0_DataNo1
							test_setLocale_PathNo0_DataNo2
							test_setLocale_PathNo0_DataNo3

Test data

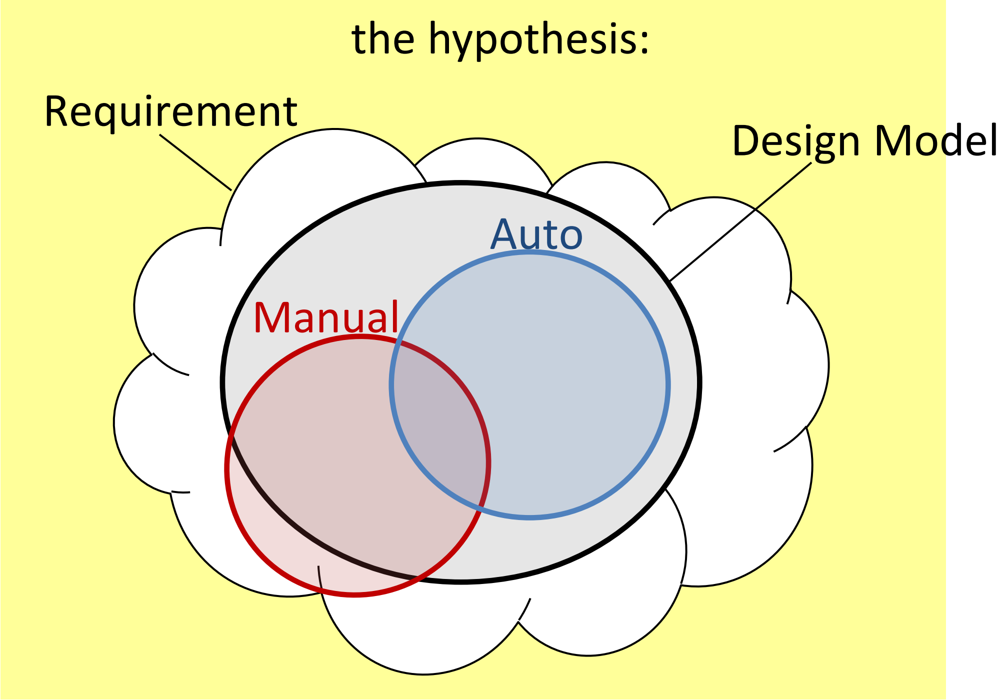
Path_Name	TestData_ID	status:OrderStatusNames	orderId:String	Expected_Status	Is_Normal	Test_Result
PathNo0	DataNo0	status:OrderStatusNames = APPROVED	orderId:String=任意文字列(999文字)	chagnedOrderが追加されていること。	TRUE	OK / NG
PathNo0	DataNo1	status:OrderStatusNames = SHIPPED_PART	orderId:String=任意文字列(1文字)	chagnedOrderが追加されていること。	TRUE	OK / NG
PathNo0	DataNo2	status:OrderStatusNames = DENIED	orderId:String=任意文字列(1000文字)	chagnedOrderが追加されていること。	TRUE	OK / NG
PathNo0	DataNo3	status:OrderStatusNames = PENDING	orderId:String=任意文字列(0文字)	INDEFINITE	FALSE	OK / NG
PathNo0	DataNo4	status:OrderStatusNames = COMPLETED	orderId:String=任意文字列(999文字)	INDEFINITE	FALSE	OK / NG

Agenda

- Background and motivation
- Generation approach
- Tool implementation
- Evaluation results
- Future work and conclusion

Evaluation: Overview

- Case study
  - One component of a shopping store application
    - small Java web application (about 4.8KLOC)
  - Manually build an design model in UML2.0
  - Compare the artifacts made from the same model,
    - Manually derived by one average skilled developer
    - Automatically generated with our proposed method
- Viewpoints:
  - manpower cost
  - SUT coverage, test density



Evaluation: Results (1/3)

NTT

- Manpower cost
  - By automation,
    about 52% manpower can be reduced

		Manual	Automatic
Man-power	for test design(minute) for other overheads(minute)	2330 0	0 1123 *
Unit man-power	per testcase (man-minute) per KLOC (man-hour)	12.3 8.1	3.1 3.9

* The cost of retouching existing "roughly sketched" UML models to a more detailed level, enough to generate test cases.
 This assuming to be 50% of the overall cost of UML modeling.

Evaluation: Results (2/3)

- Test density and SUT coverage
  - By automation,
    test density increased to 1.9 times
    and the SUT coverage can be improved

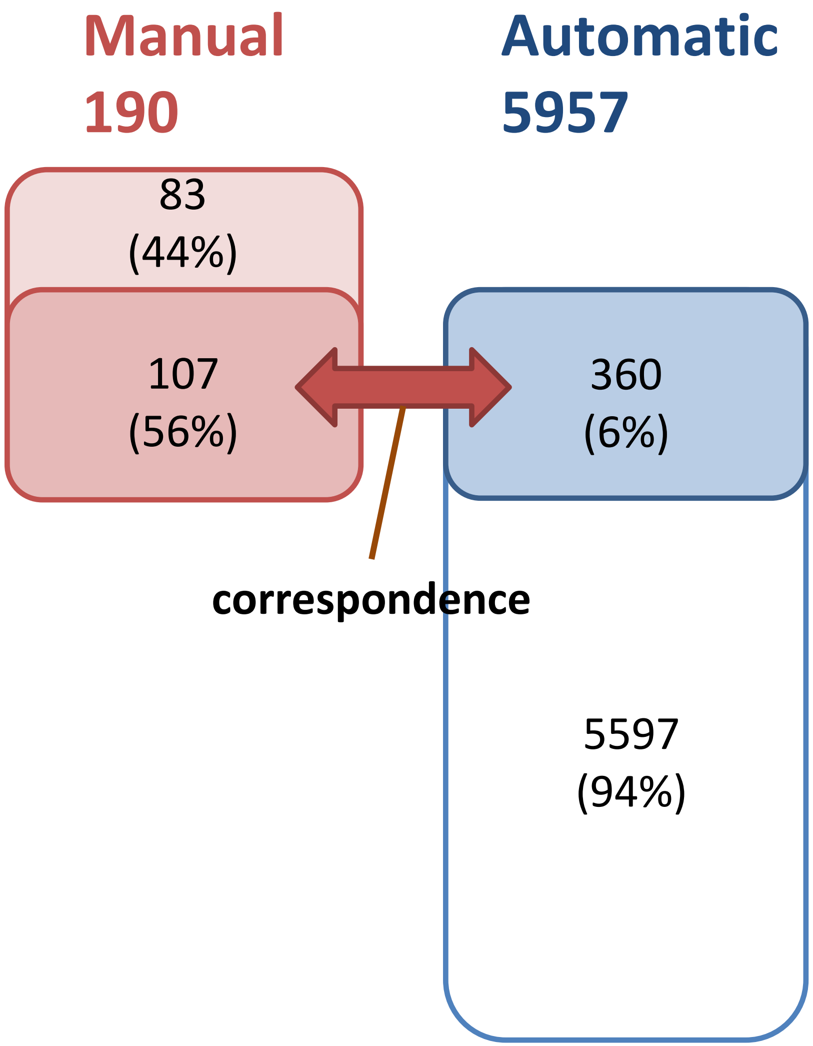
Test density (testcase per KLOC)			39.6	75.0
SUT Coverage	Structure		covered all paths	covered all paths
	Input space	values	covered representative values only	covered all boundary values
		combinations	without clear rules	covered all values

- Number of test cases
  - Good:

    56% of manual test cases can be generated by the proposed approach
  - Bad:

    the technique could not correspond to 44% of the manual derived test cases.

    the number of generated test data is too large to be used in test execution.



Evaluation: discussion

ⓐ Matching test cases

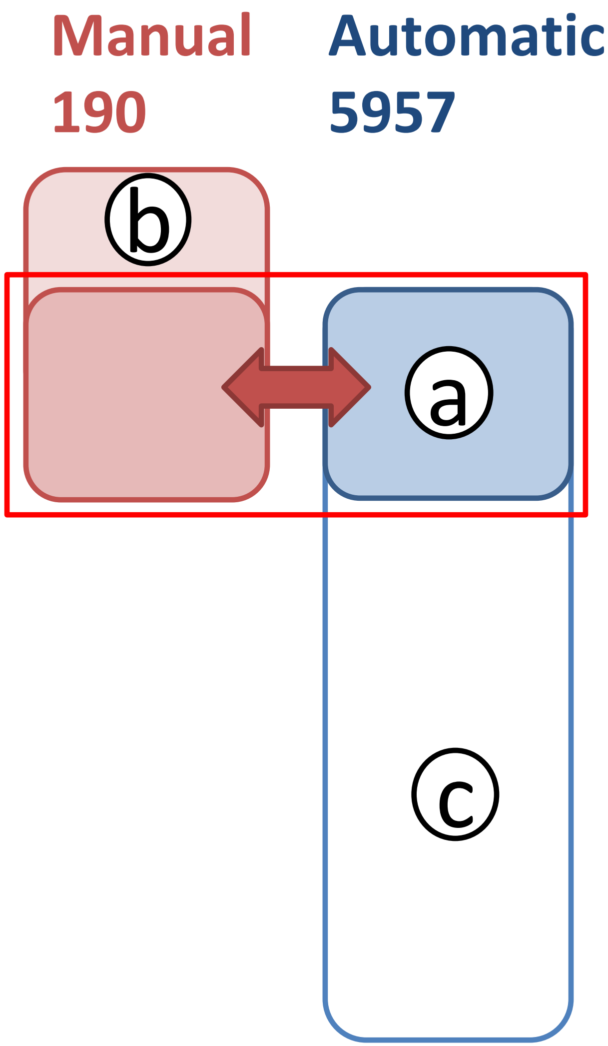
- Most of automatic test cases with normal inputs
- improvement in input space coverage
  - more variation of value
  - more exhaustive combination

ⓑ Test cases created only by the expert

- difficult to formalize semantics of the software
- nested structure of activities

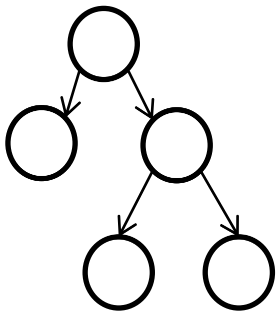
ⓒ Test cases created only by tool

- most of test cases with abnormal inputs
- The value of object data type has more variation than human usually consider



Evaluation: discussion

- What Next?
  - ⓑ How to deal with the knowledge not described in the UML model?
    - decide a notation and describing them in the UML model.
    - Or, extract from information formalized outside of the UML model.
  - ⓒ “many more” does not mean “better”.
    - It is necessary to narrow down the number of test cases to a practicable level
      - For example, for object data type, Not all combination of “leaf level” abnormal values But only the important cases



Agenda

- Background and motivation
- Generation approach
- Tool implementation
- Evaluation results
- Future work and conclusion

Conclusions

- The proposed method extract test cases, and test data with hierarchical structure, from UML activity and class models.

- our goal was to... improve “test design”, with low cost by automation
  - By introducing automatic generation, for A PART OF the test design work, higher SUT coverage and test density can be achieved with fewer man-power.
  - Though further improvement is required.
    - Need to apply other methods to solve the restrictions.
    - Need methods to handle the outside of UML model

Future Works, Potentials, (for discussion…)

NTT

- Vision: Toward totally automated testing

  (Integrate "test design automation" with "test execution automation")

    - large number of test cases will not be problem.
- Gaps!
  - "Free Style" Modeling is still difficult for normal developer.
    - Convert from "ordinary" design document template
  - Still need human to check the detailed test result.
    - Test oracle
- Before we reach the utopia…
  - Test case selection/prioritization. For manual test execution
  - Effectiveness evaluation. Mutation Testing, more large scale software
- Never forget the development process
  - Explain the merit of modeling and automated generation.

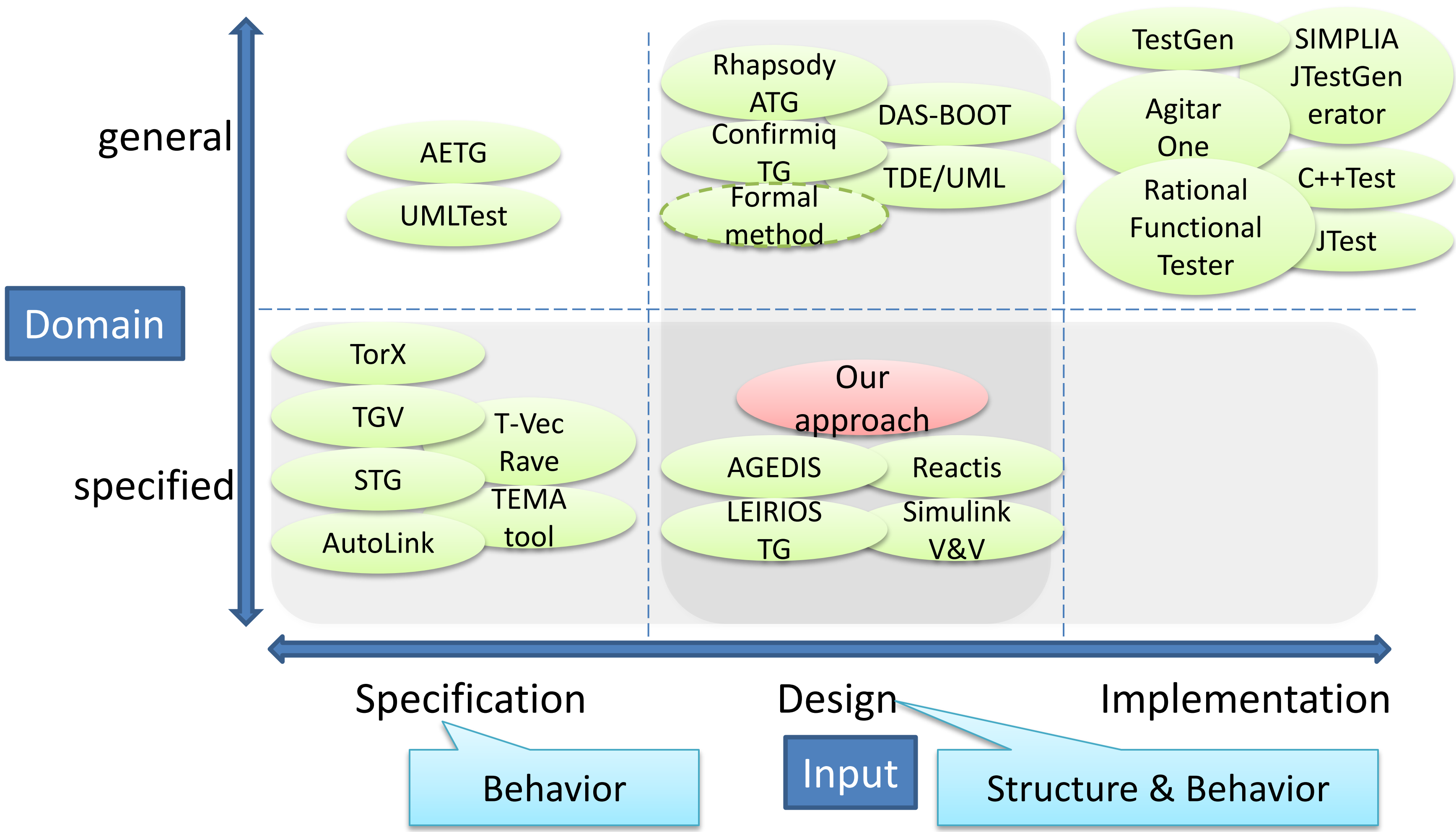
Xie xie!

Questions or comment?

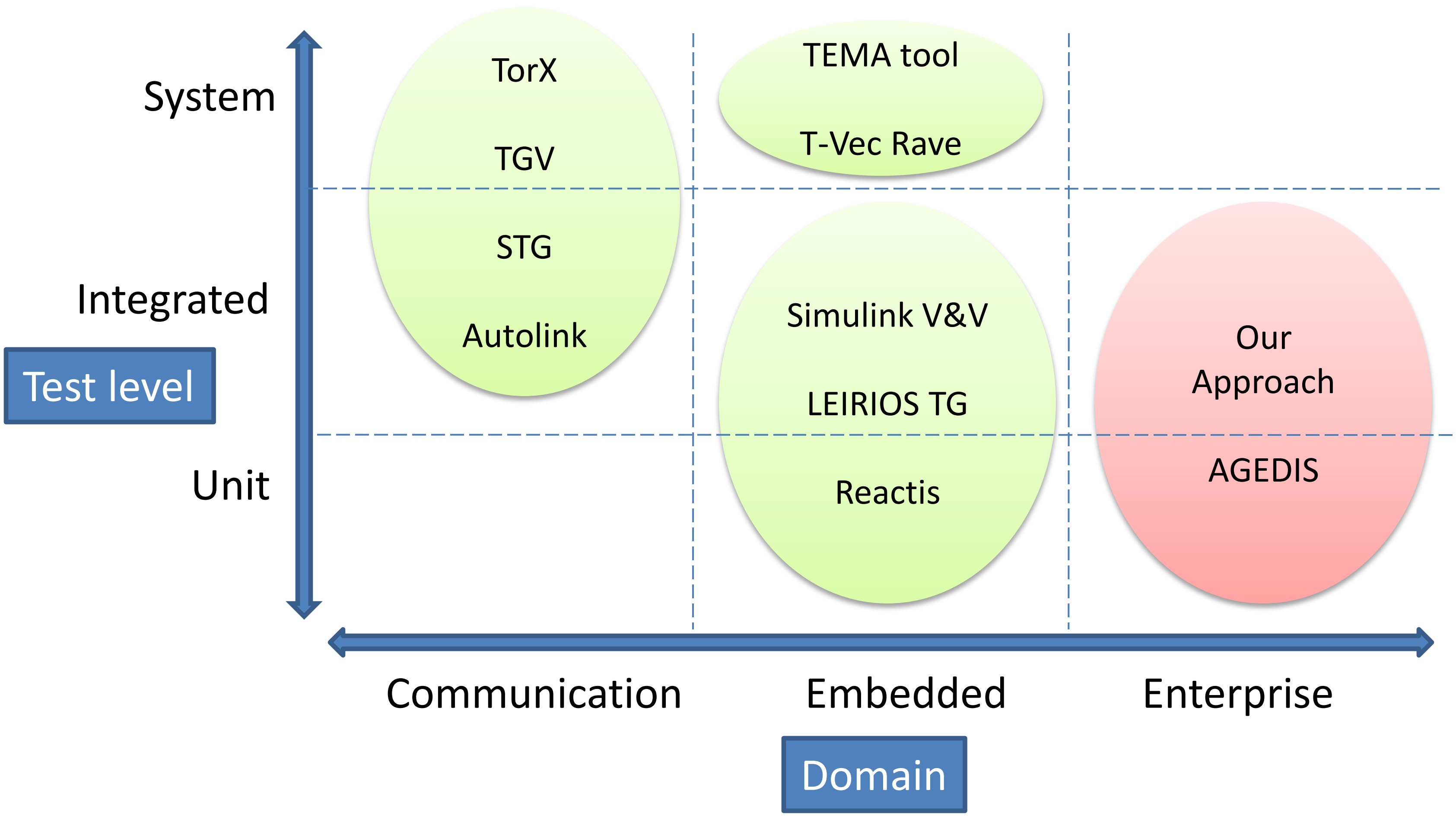
zhang.xiaojing@lab.ntt.co.jp

- Extra slides

Position Map



Position Map (target domain)

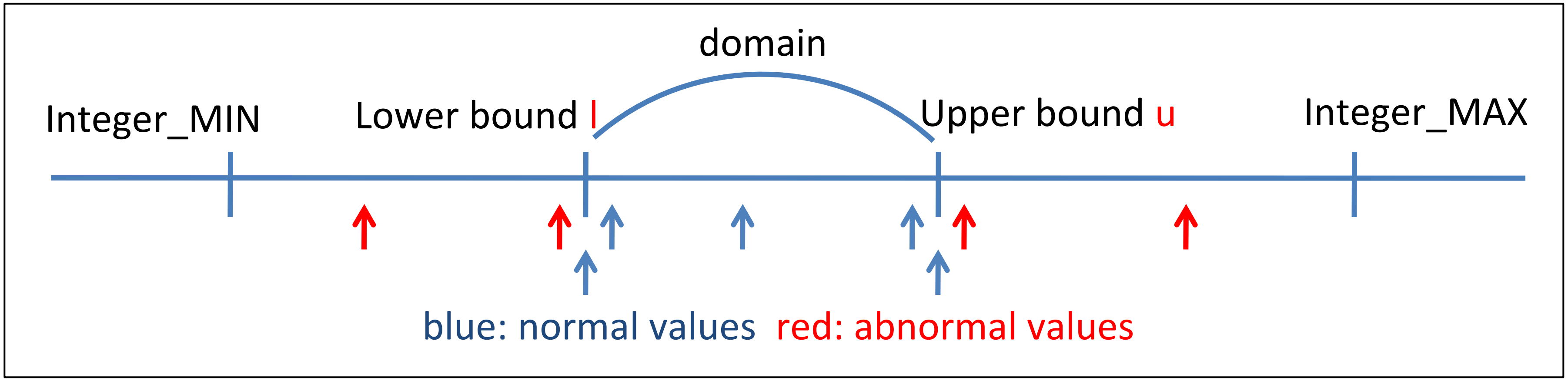


Generation technique: Test data (4/6)

③ Generation of the values

- equivalent classes division and the boundary value analysis
- extract not only the values which are inside of the domain as normal values, but also the values which are outside of the domain as abnormal values.
- For object data type, expanding the object hierarchically, if, as much as one “abnormal value” exists in a “leaf node” then the entire object is assumed to be an abnormal value.

Example: from Integer domain



Generation technique: Test data (5/6)

NTT

④ Combination of the values

- <u>Normal Inputs</u> consist of normal values
  - minimum combination which can use all values generated.
- <u>Abnormal Inputs</u> contains only one abnormal value
  - make the fault localization easier when failure occurs

Example

Values for variable a, b, and c

Normal value	a1 a2 a3	b1	c1 c2
Abnormal Value	a4 a5	b2	c3 c4

Combination

Test data : Normal Inputs

ID	a	b	c
1	a1	b1	c1
2	a2	b1	c2
3	a3	b1	c1

Test data : Abnormal Inputs

4	a4	b1	c1
5	a5	b1	c2
6	a3	b2	c1
7	a1	b1	c3
8	a2	b1	c4