From Requirements to Code in a Snap

Michał Śmiałek

Warsaw University of Technology

MODELS Tutorial

04.10.2010

The presenter…

Michał Śmiałek: ME-HOW ☺

What I do/did (since around 1991)

- Professor of CS at the Warsaw University of Technology (Poland): responsible for the CS course curriculum at EE; UML, Requirements, Design, MDSD
- Coach in many commercial SE courses (> 150 editions, >3000 hours)
- SE consultant at Infovide-Matrix (top Polish IT consulting company)
- Project manager in SE projects
- SE methodology consultant (RUP certified trainer/coach)
- Software/business analyst
- Programmer

Tutorial outline

Introduction – requirements and MDSD

- Requirements modeling and transformation "snaps"

Programming at the requirements level

- Clarification of use case semantics
- Application logic vs. domain logic
- Programming understandable by "ordinary people"?

From requirements to code

- Automatic transformation of requirements (use case scenarios) into application logic and domain logic code

Software evolution as requirements evolution

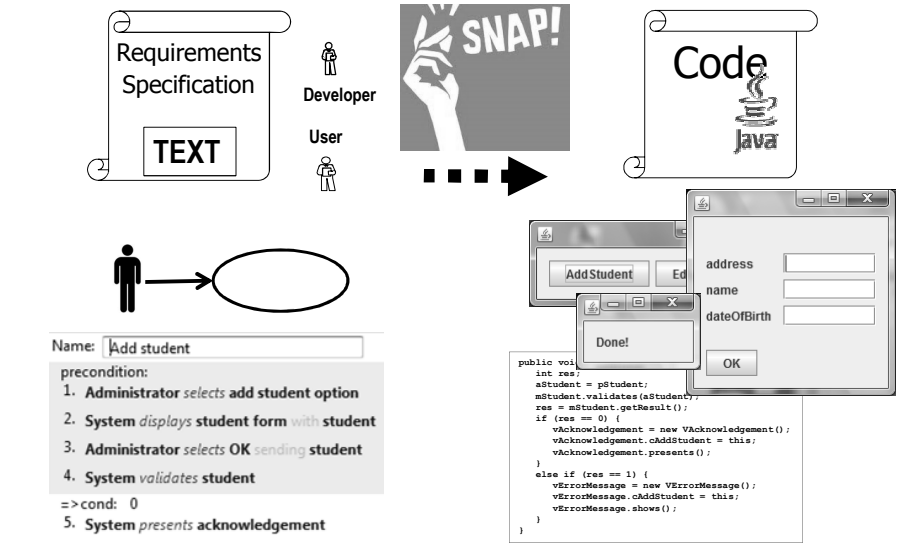
- Changes in requirements "programs" drive changes in the system

Summary and conclusion

3 From Requirements to Code in a Snap Michał Śmiałek

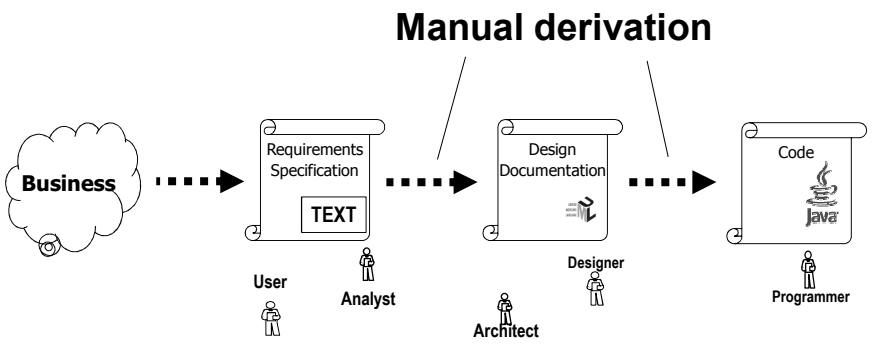
:: INTRODUCTION – REQUIREMENTS AND MDSD::

SD project manager's dream?



From business to code - traditional

Requirements specification in natural language → design specification derived from requirements → code derived from design models

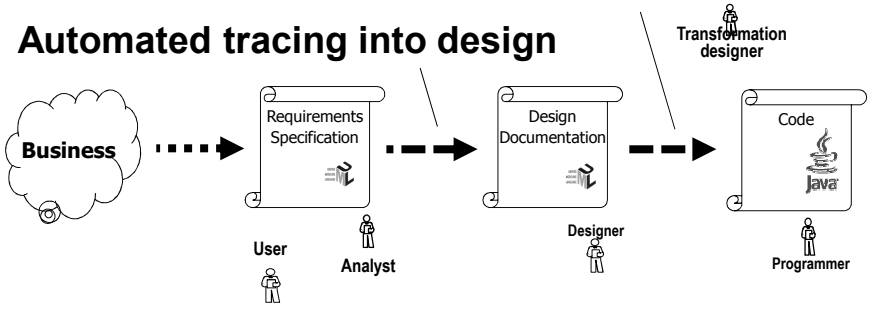


From business to code – model driven

Requirements spec. using semi-formal models → design specification traced from requirements → code transformed from design models

Semi-automatic transformation

Automated tracing into design



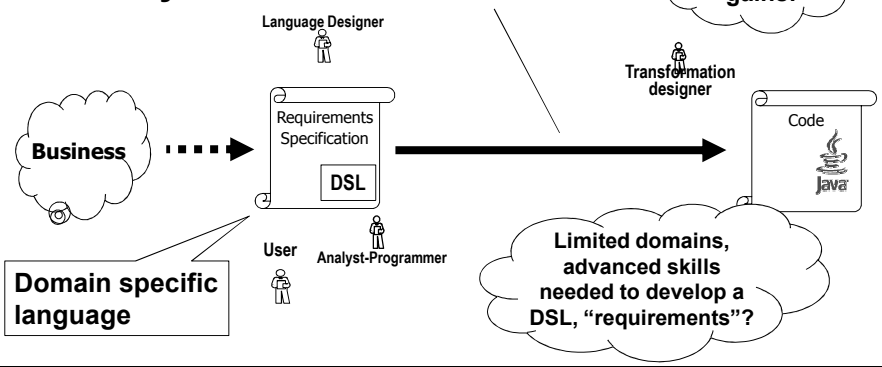
7 From Requirements to Code in a Snap Michał Śmiałek

From business to code – domain specific

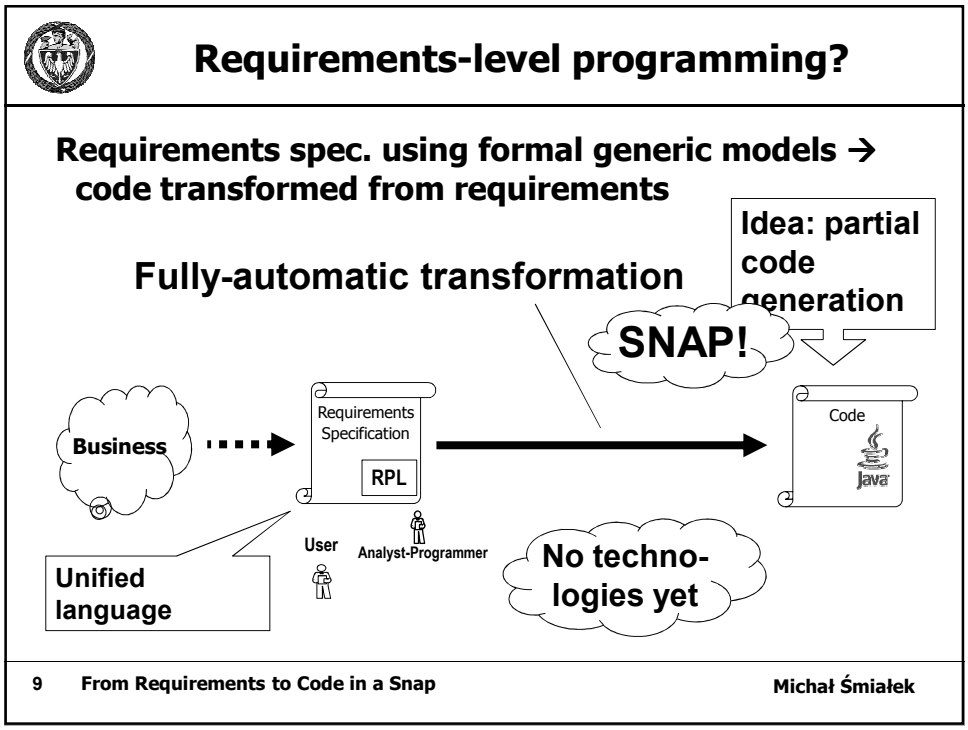
Requirements spec. using domain specific models → code transformed from requirements

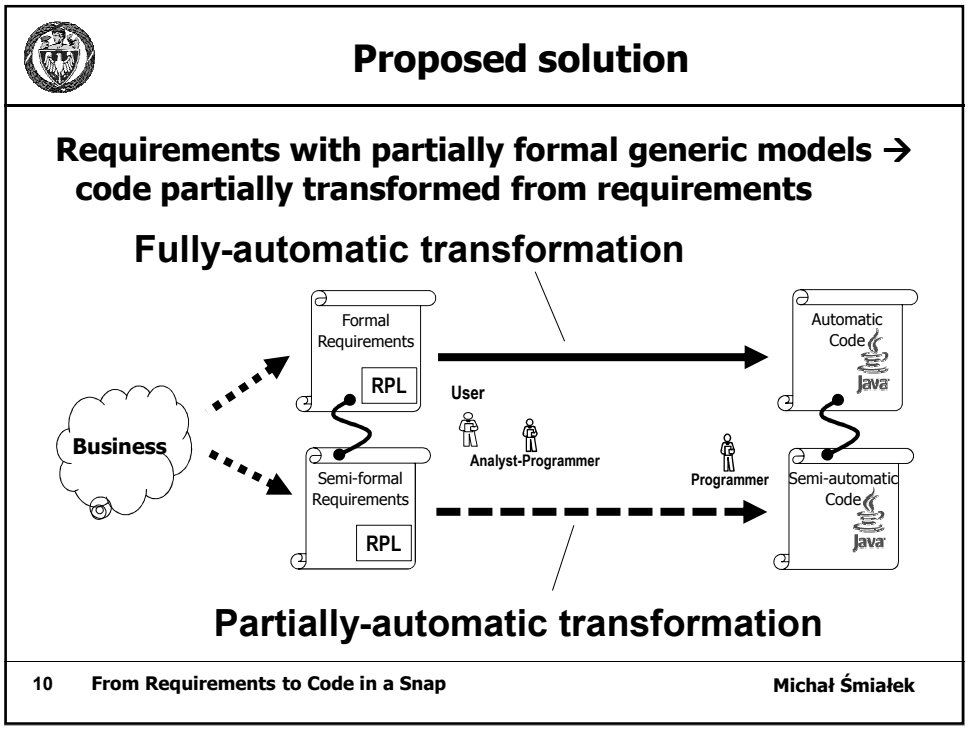
Fully-automatic transformation

Significant productivity gains!



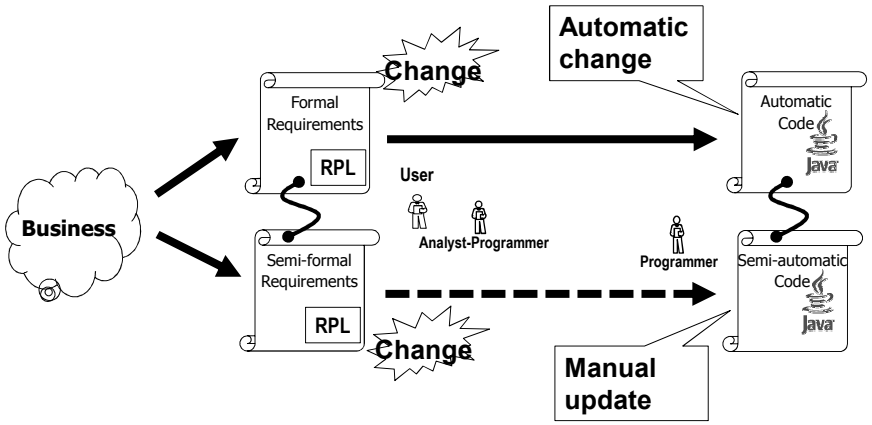
8 From Requirements to Code in a Snap Michał Śmiałek


Requirements-level programming?
Requirements spec. using formal generic models → code transformed from requirements
Fully-automatic transformation
Idea: partial code generation
SNAP!
Business
Requirements Specification
RPL
Code
Java
User
Analyst-Programmer
Unified language
No techno-logies yet
9 From Requirements to Code in a Snap
Michał Śmiałek


Proposed solution
Requirements with partially formal generic models → code partially transformed from requirements
Fully-automatic transformation
Formal Requirements
RPL
Automatic Code
Java
User
Business
Analyst-Programmer
Programmer
Semi-formal Requirements
RPL
Semi-automatic Code
Java
Partially-automatic transformation
10 From Requirements to Code in a Snap
Michał Śmiałek

Evolution schema

Change in requirements → automatic change in code
→ pointers to code for manual change



11 From Requirements to Code in a Snap Michał Śmiałek

What do we need?

Precise but understandable requirements specification language

Operational semantics for the language – "how to transform automatically requirements into code"

Means to trace from changing requirements to changing code

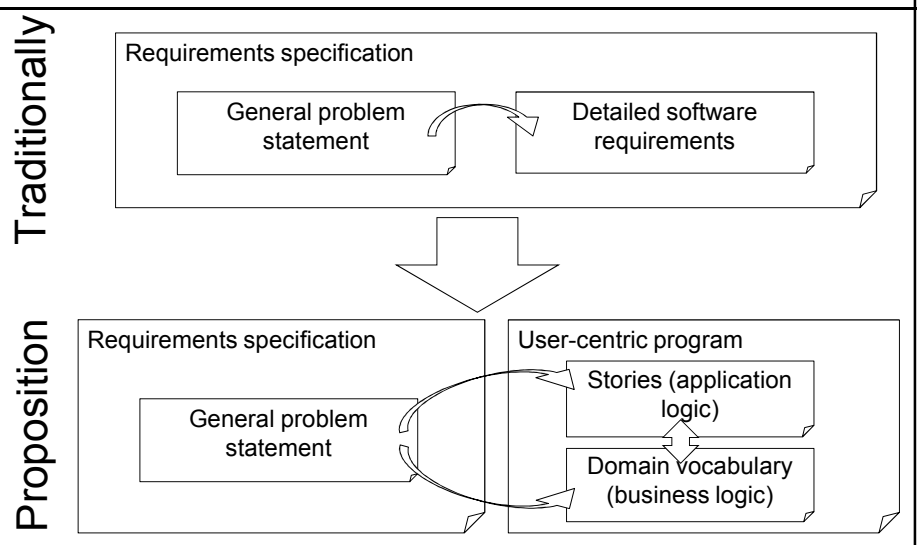
Result:

- Very agile process allowing for quick realization of user requirements in code

12 From Requirements to Code in a Snap Michał Śmiałek

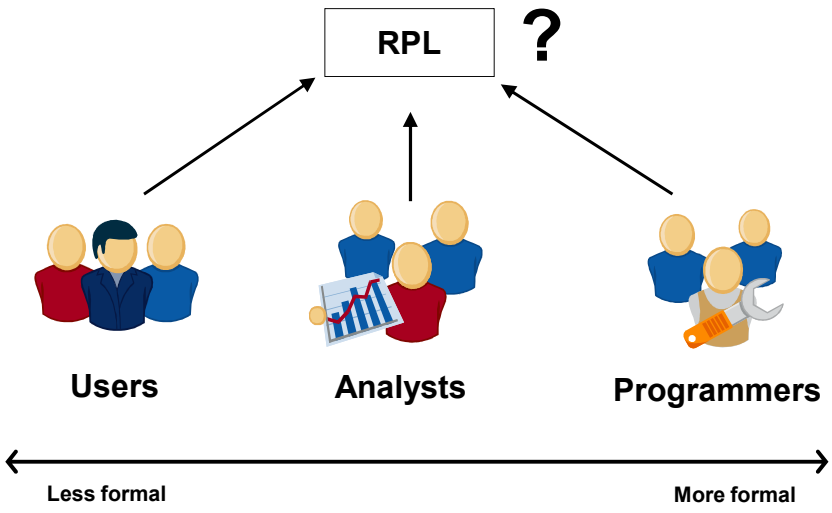
:: PROGRAMMING AT THE REQUIREMENTS LEVEL ::

Programming at a higher level?



14 From Requirements to Code in a Snap Michał Śmiałek

Combining natural language with precision



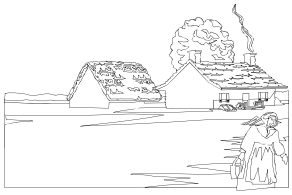
15 From Requirements to Code in a Snap Michał Śmiałek

The story and the environment

4. Tom rides to the old hut.

2. Lord Mark goes to the old hut.

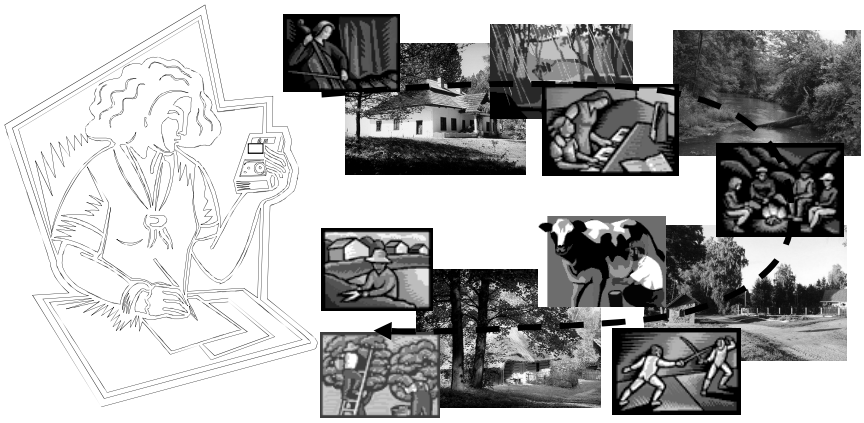
3. Aunt Martha goes to Tom “the wise”.



1.a. Lord Mark runs to aunt Marta’s house.
b. He talks with aunt Martha.

16 From Requirements to Code in a Snap Michał Śmiałek

Writing good stories vs. writing good requirements

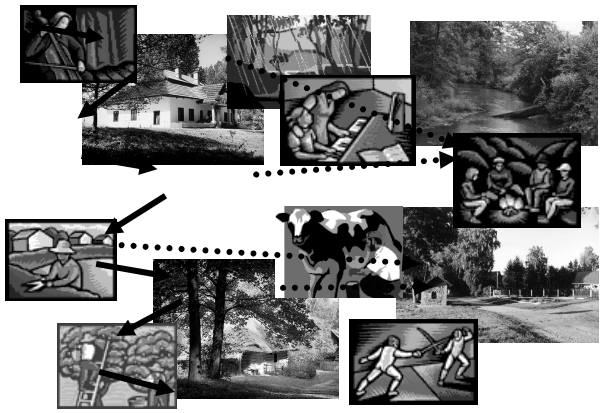


A combination of story (sequence of events) and descriptions of the environment (people, interiors, landscapes…).

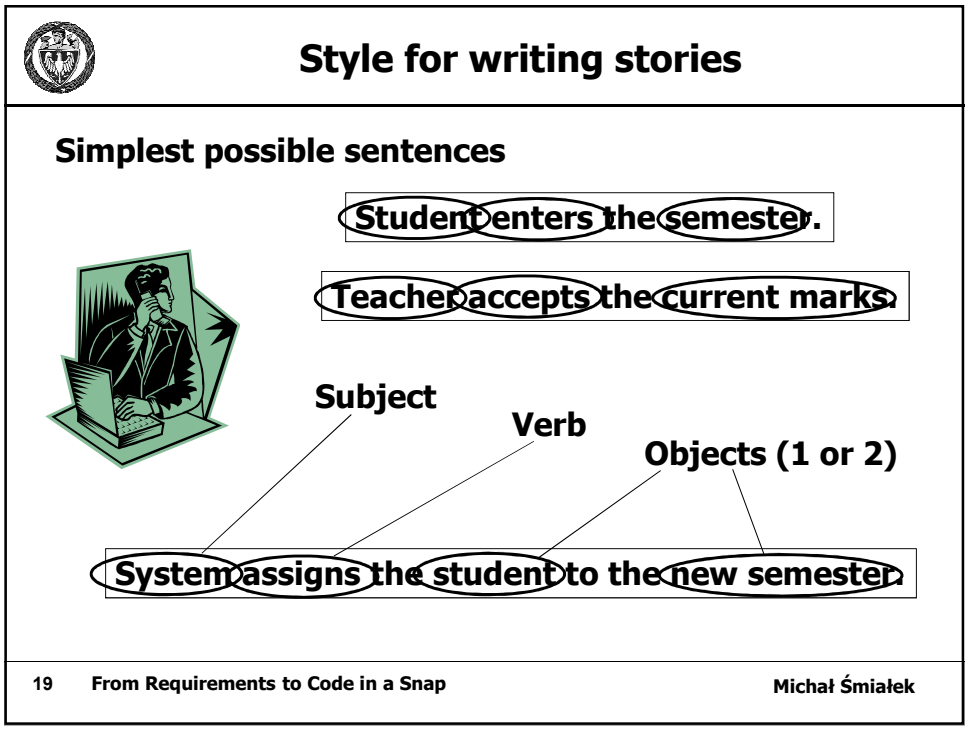
17 From Requirements to Code in a Snap Michał Śmiałek

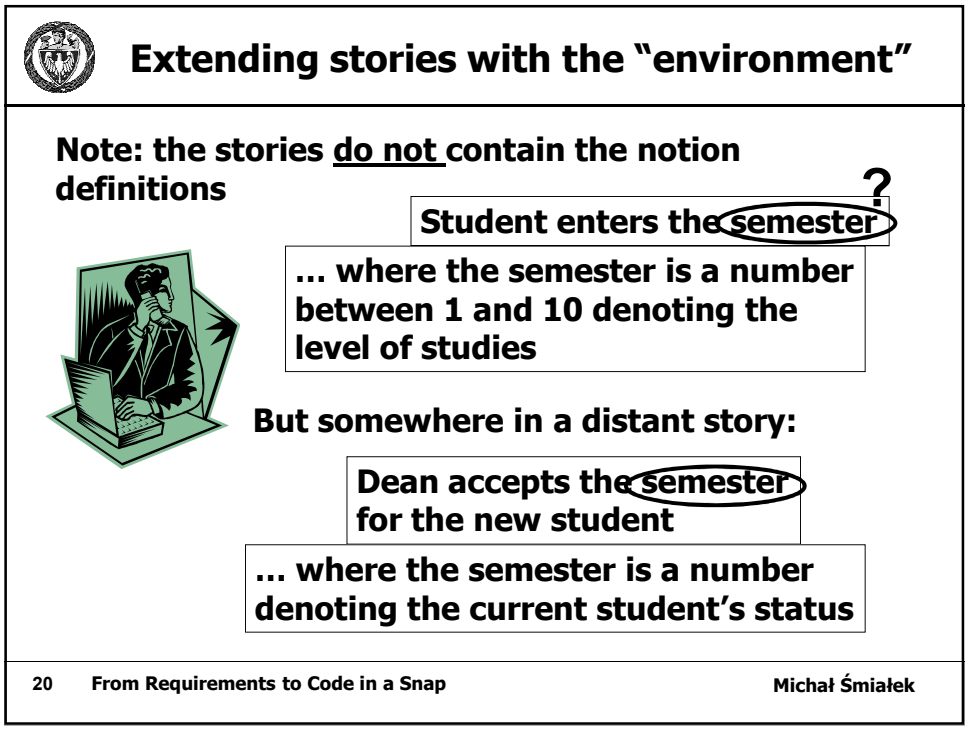
Separating the story and the environment

Events vs. scenes



18 From Requirements to Code in a Snap Michał Śmiałek


Style for writing stories
Simplest possible sentences
Student enters the semester.
Teacher accepts the current marks.
Subject
Verb
Objects (1 or 2)
System assigns the student to the new semester.
19 From Requirements to Code in a Snap Michał Śmiałek


Extending stories with the "environment"
Note: the stories do not contain the notion definitions
Student enters the semester
?
... where the semester is a number between 1 and 10 denoting the level of studies
But somewhere in a distant story:
Dean accepts the semester for the new student
... where the semester is a number denoting the current student's status
20 From Requirements to Code in a Snap Michał Śmiałek

Separating notions from the story

Separate notion vocabulary but consistent with the stories

Student enters the semester

… where the semester is a number between 1 and 10 denoting the level of studies

Semester – number between 1 and 10 denoting the level of studies and the current student's status.

Writing scenarios

Scenario is a sequence of SVO sentences

Dean adds new lecture to a course

1. **Dean** *selects to add* new lecture to course
2. **System** *asks* for semester
3. **Dean** *enters* the semester
4. **System** *asks for data* of the lecture
5. **Dean** *enters the data* of the lecture
6. **System** *adds the lecture* to the list of lectures

Building a vocabulary

The vocabulary is the user's "map of territory". Better to have it in a graphical form.

1. **Dean** wants to *add new lecture* to course
2. **System** *asks* for semester
3. **Dean** *enters* the semester
4. **System** *asks for data* of the lecture
5. **Dean** *enters the data* of the lecture
6. **System** *adds the lecture* to the list of lectures

5. **System** *assigns* the teacher to the lecture

Lecture | Student | Course | Teacher | List of lectures | Semester

23 From Requirements to Code in a Snap Michał Śmiałek

Playing-out stories

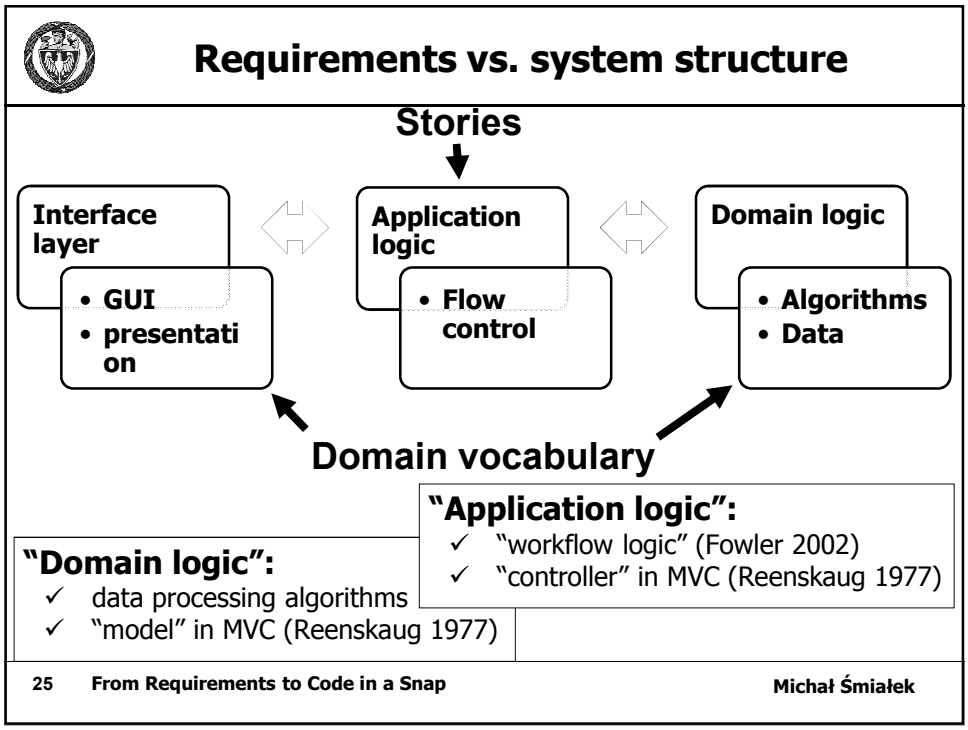
Coherence of the stories can be checked by playing the stories.

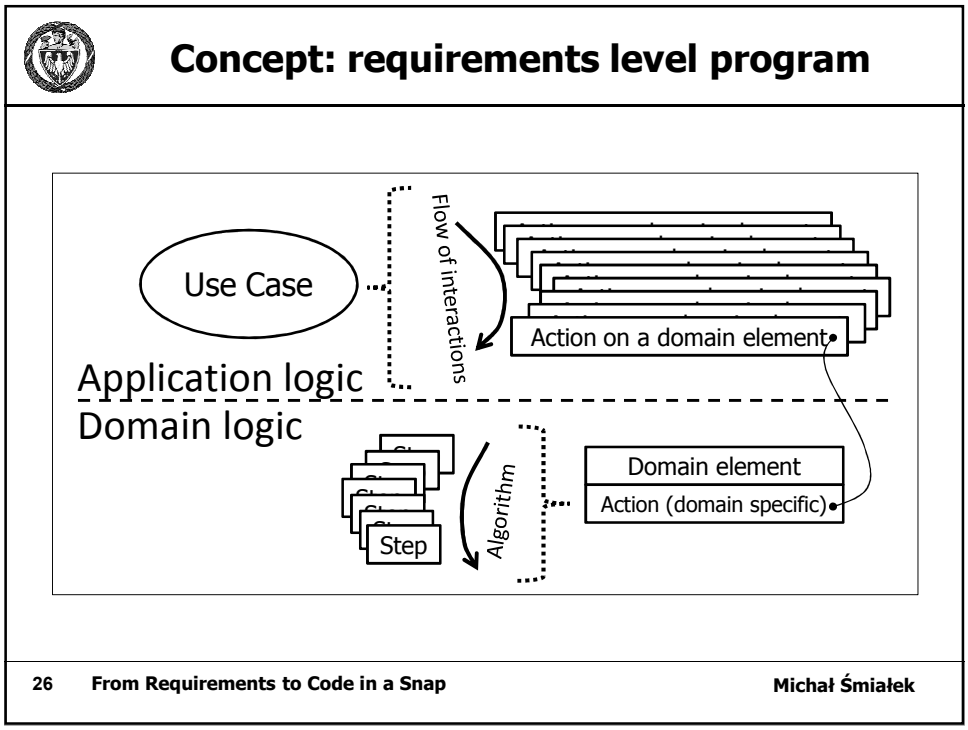
1. **Dean** wants to *add new lecture* to course
2. **System** *asks* for semester
3. **Dean** *enters* the semester
4. **System** *asks for data* of the lecture
5. **Dean** *enters the data* of the lecture
6. **System** *adds the lecture* to the list of lectures

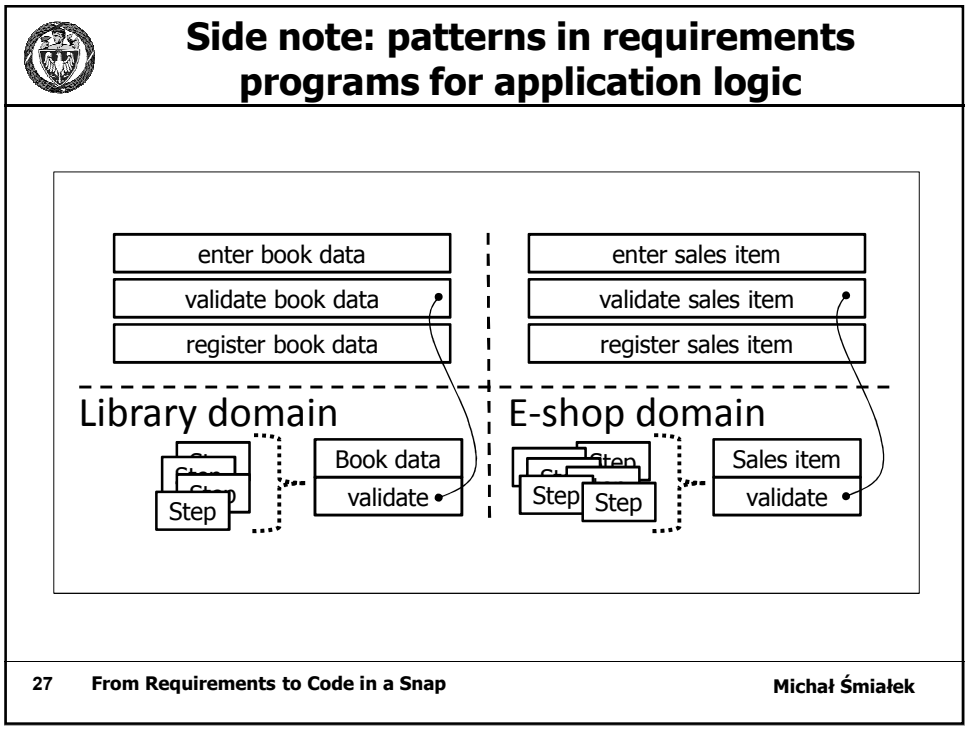
add new lecture *enter* *ask for*

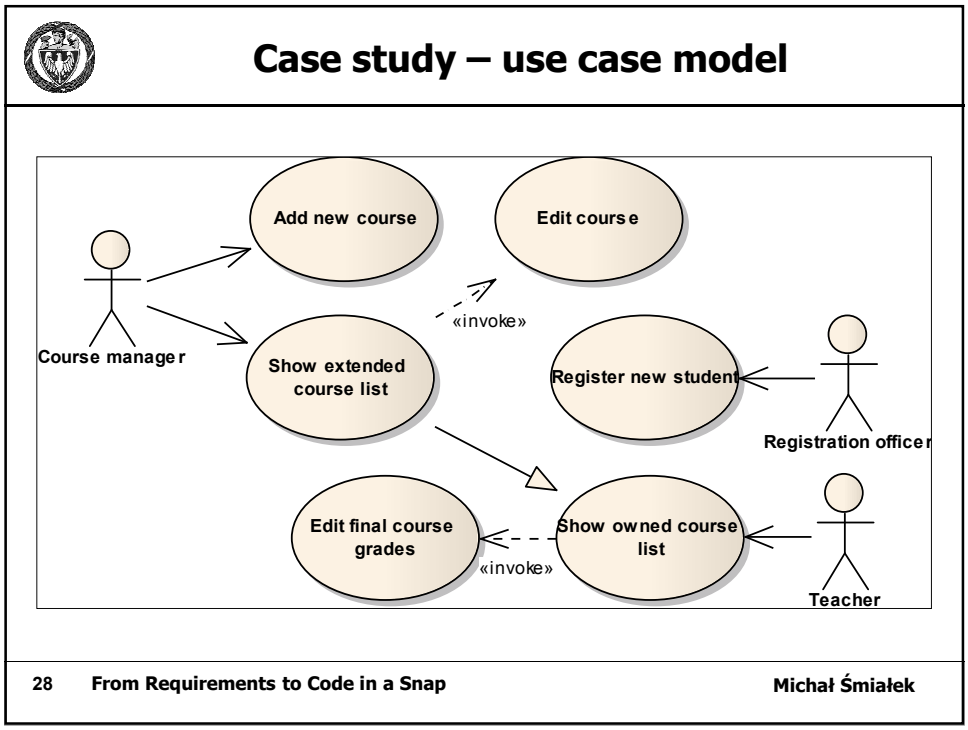
Lecture | Student | Course | Teacher | List of lectures | Semester

24 From Requirements to Code in a Snap Michał Śmiałek


Requirements vs. system structure
Stories
Interface layer
• GUI
• presentation
Application logic
• Flow control
Domain logic
• Algorithms
• Data
Domain vocabulary
"Application logic":
✓ "workflow logic" (Fowler 2002)
✓ "controller" in MVC (Reenskaug 1977)
"Domain logic":
✓ data processing algorithms
✓ "model" in MVC (Reenskaug 1977)
25 From Requirements to Code in a Snap Michał Śmiałek


Concept: requirements level program
Use Case
Flow of interactions
Action on a domain element
Application logic
Domain logic
Step
Algorithm
Domain element
Action (domain specific)
26 From Requirements to Code in a Snap Michał Śmiałek


Side note: patterns in requirements programs for application logic
enter book data
validate book data
register book data
enter sales item
validate sales item
register sales item
Library domain
E-shop domain
Book data
validate
Sales item
validate
Step
Step
Step
27 From Requirements to Code in a Snap Michał Śmiałek


Case study – use case model
Add new course
Edit course
«invoke»
Course manager
Show extended course list
Register new student
Registration officer
Edit final course grades
«invoke»
Show owned course list
Teacher
28 From Requirements to Code in a Snap Michał Śmiałek

Case study – initial scenarios

1. **Teacher** *selects to see* **course list** — system
2. **System** *builds* **course list** — system
3. **System** *shows* **course list** — teacher

=>invoke/REQUEST — Edit final course grades

final: success

1. **Course manager** *selects to add* **course** — system
2. **System** *shows* **course entry form** — course manager
3. **Course manager** *enters* **course** — system
4. **System** *verifies* **course** — system

=>cond: course correct

5. **System** *registers* **course** — system

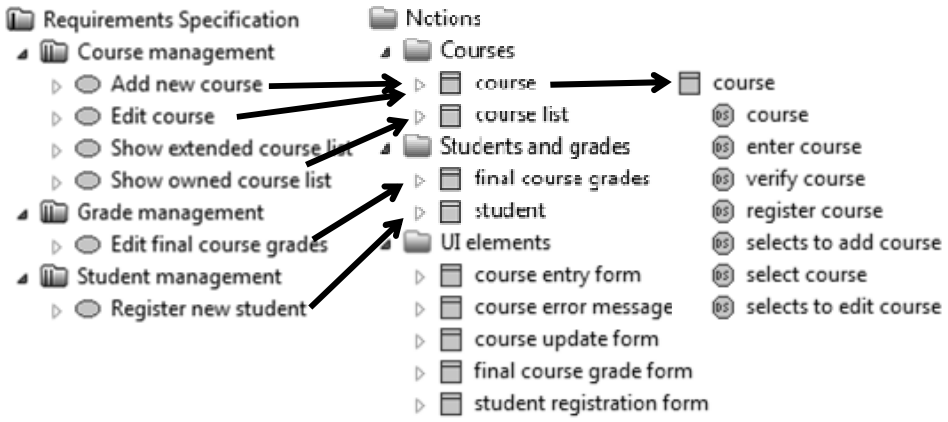
final: success

=>cond: course incorrect

4.1.1 **System** *shows* **course error message** — course manager

4.1.2 **System** *shows again* **course entry form** — course manager

Case study – initial requirements model

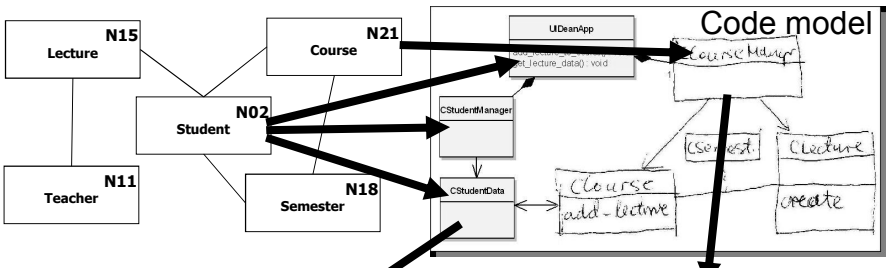


:: FROM REQUIREMENTS TO CODE ::

31 From Requirements to Code in a Snap Michał Śmiałek

Obtaining the code structure?

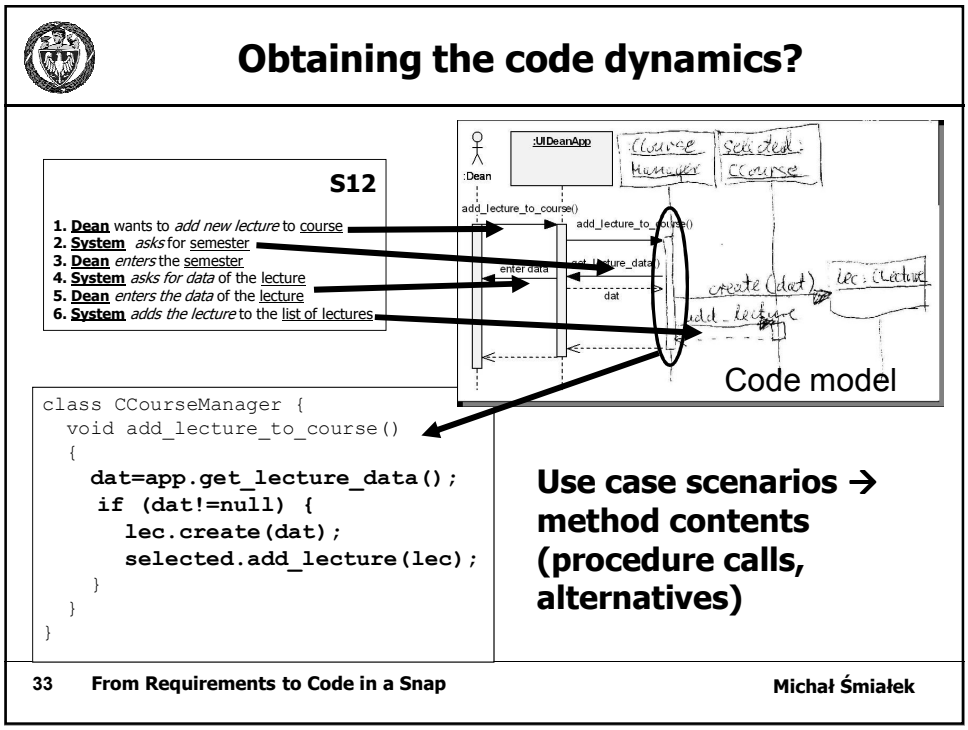
Vocabulary notions → Java (etc.) classes




```
class CStudentData {
  void update_data(Dat d)
  {
    \\ no code yet
  }
}
```

```
class CCourseManager {
  void add_lecture_to_course()
  {
    \\ no code yet
  }
}
```

32 From Requirements to Code in a Snap Michał Śmiałek



Transformation rules

Source model written in Requirements Programming Language (RPL)

- Use cases written in SVO grammar
- Separate vocabulary

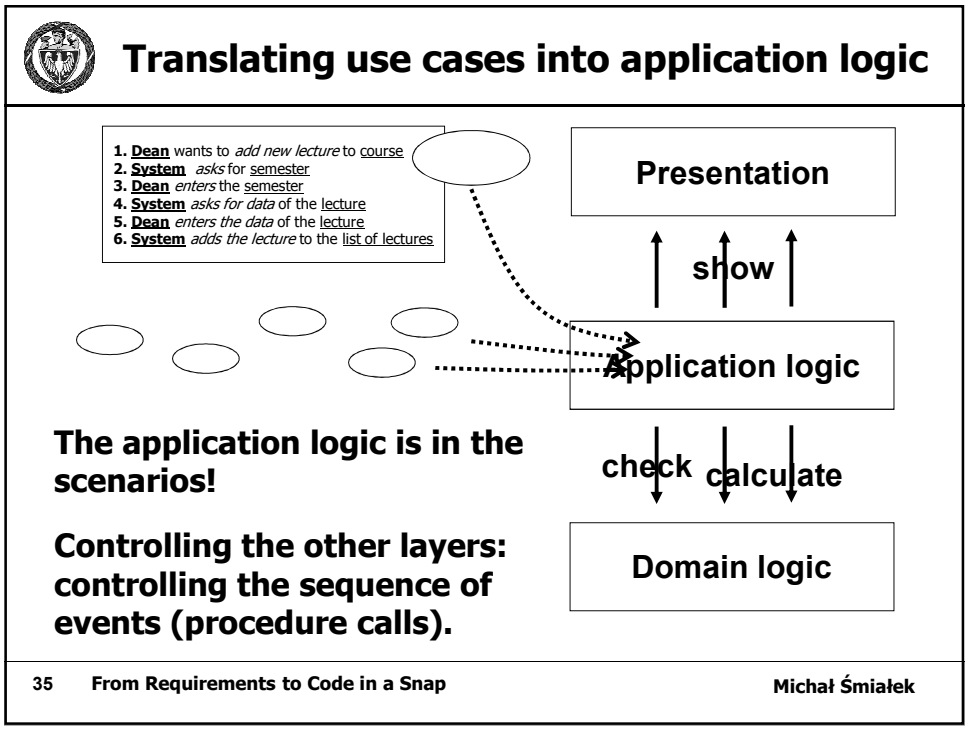
Target code has the following architecture

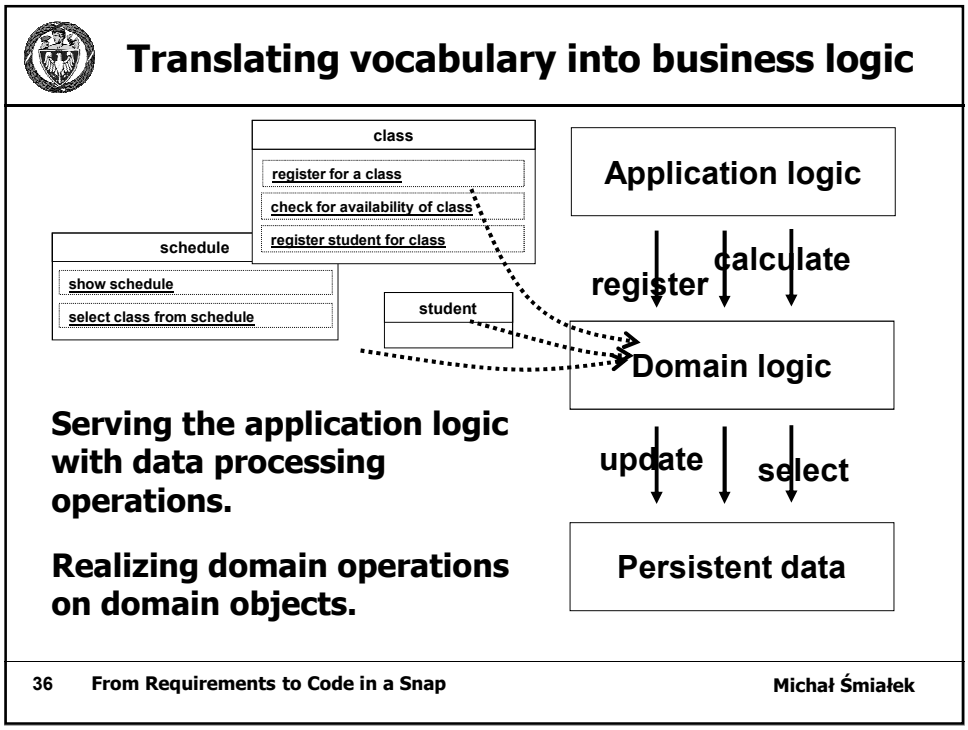
- Three layers: presentation, application logic, domain logic
- Structured into components (packages)
- Domain logic controls the application (controller)

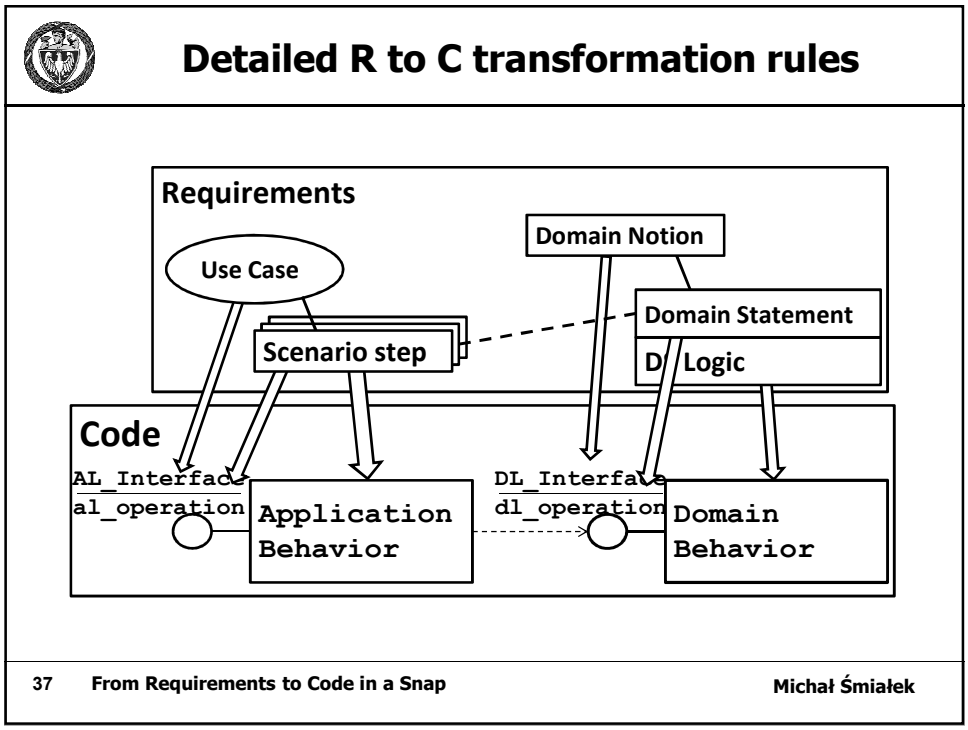
General transformation rules

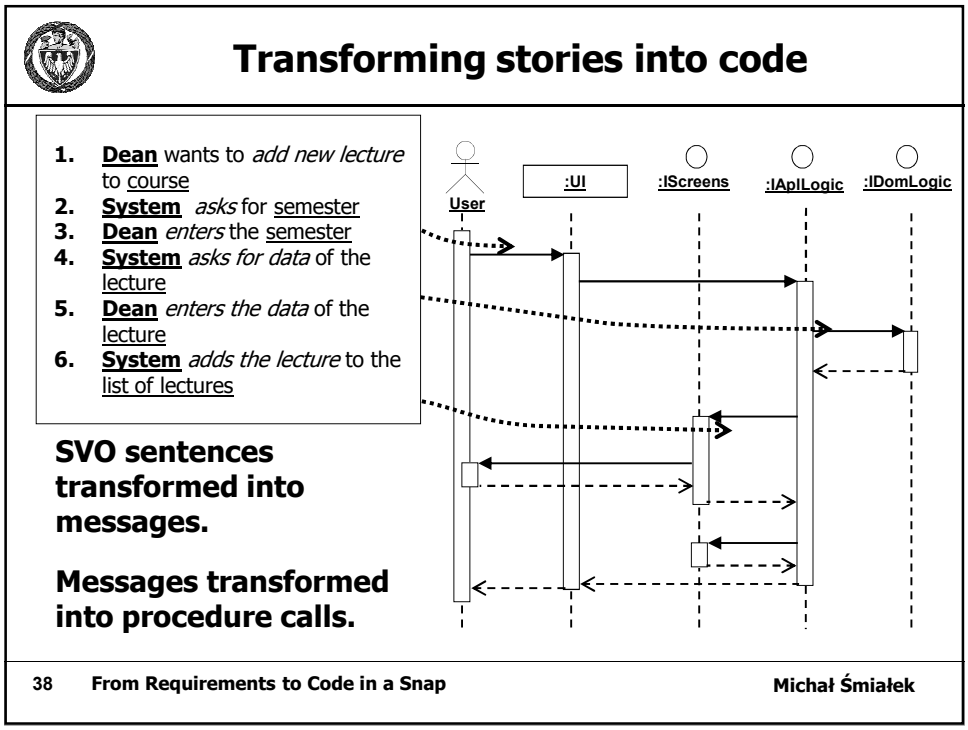
- Use cases transformed into application logic
- Domain vocabulary transformed into domain logic

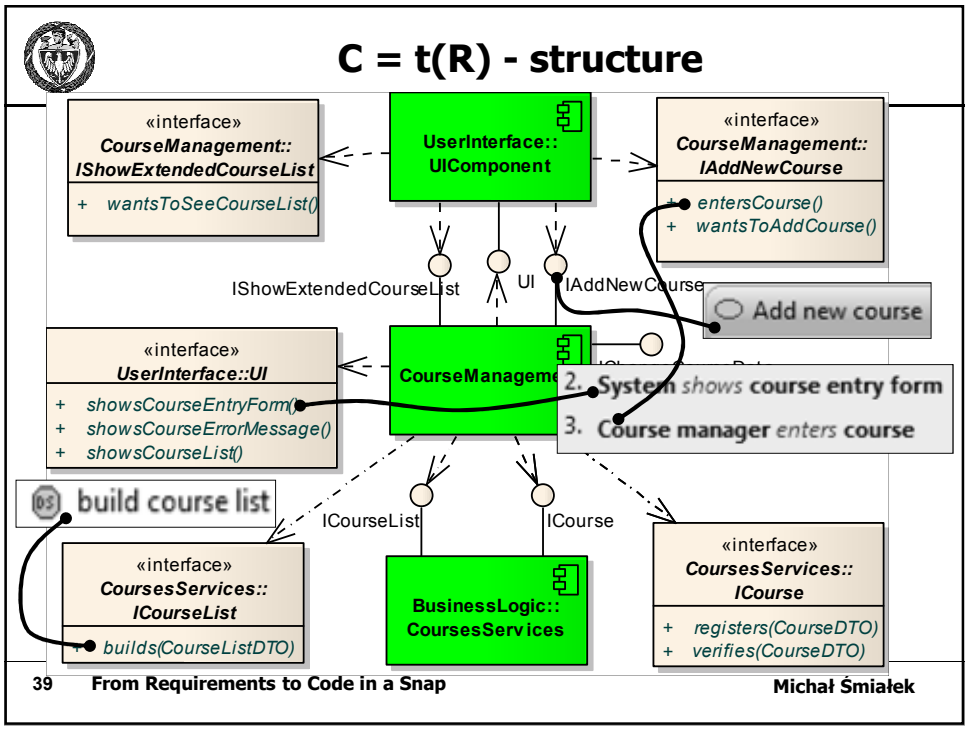
34 From Requirements to Code in a Snap Michał Śmiałek

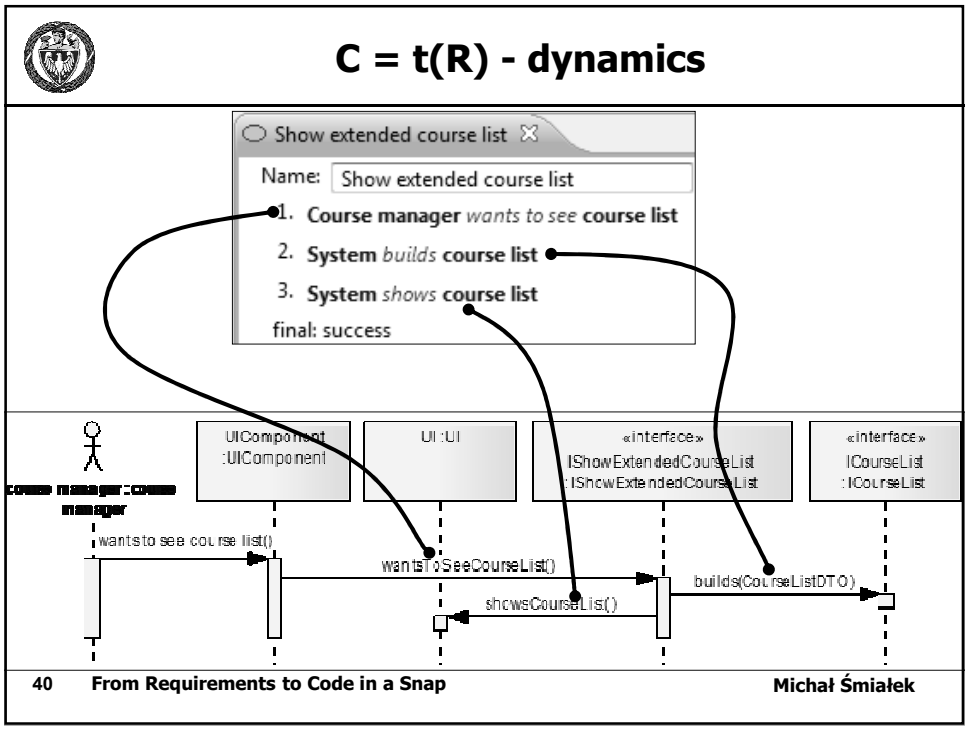

Translating use cases into application logic
1. Dean wants to add new lecture to course
2. System asks for semester
3. Dean enters the semester
4. System asks for data of the lecture
5. Dean enters the data of the lecture
6. System adds the lecture to the list of lectures
Presentation
show
Application logic
check
calculate
Domain logic
The application logic is in the scenarios!
Controlling the other layers: controlling the sequence of events (procedure calls).
35 From Requirements to Code in a Snap
Michał Śmiałek


Translating vocabulary into business logic
class
register for a class
check for availability of class
register student for class
schedule
show schedule
select class from schedule
student
Application logic
register
calculate
Domain logic
update
select
Persistent data
Serving the application logic with data processing operations.
Realizing domain operations on domain objects.
36 From Requirements to Code in a Snap
Michał Śmiałek

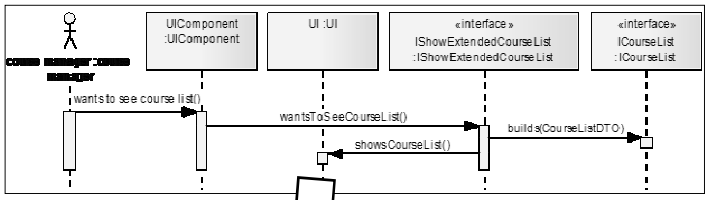

Detailed R to C transformation rules
Requirements
Use Case
Scenario step
Domain Notion
Domain Statement
D Logic
Code
AL_Interfac
al_operation
Application
Behavior
DL_Interfa
dl_operation
Domain
Behavior
37 From Requirements to Code in a Snap
Michał Śmiałek

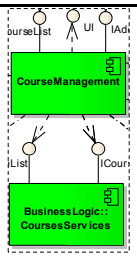

Transforming stories into code
1. Dean wants to add new lecture to course
2. System asks for semester
3. Dean enters the semester
4. System asks for data of the lecture
5. Dean enters the data of the lecture
6. System adds the lecture to the list of lectures
SVO sentences transformed into messages.
Messages transformed into procedure calls.
User
:UI
:IScreens
:IAplLogic
:IDomLogic
38 From Requirements to Code in a Snap
Michał Śmiałek


C = t(R) - structure
«interface»
CourseManagement::
IShowExtendedCourseList
+ wantsToSeeCourseList()
UserInterface::
UIComponent
«interface»
CourseManagement::
IAddNewCourse
+ entersCourse()
+ wantsToAddCourse()
IShowExtendedCourseList
UI
IAddNewCourse
Add new course
«interface»
UserInterface::UI
+ showsCourseEntryForm()
+ showsCourseErrorMessage()
+ showsCourseList()
2. System shows course entry form
3. Course manager enters course
build course list
ICourseList
ICourse
«interface»
CoursesServices::
ICourseList
+ builds(CourseListDTO)
BusinessLogic::
CoursesServices
«interface»
CoursesServices::
ICourse
+ registers(CourseDTO)
+ verifies(CourseDTO)
39
From Requirements to Code in a Snap
Michał Śmiałek


C = t(R) - dynamics
Show extended course list
Name: Show extended course list
1. Course manager wants to see course list
2. System builds course list
3. System shows course list
final: success
UIComponent
:UIComponent
UI :UI
«interface»
IShowExtendedCourseList
«interface»
ICourseList
:ICourseList
wantsToSeeCourseList()
builds(CourseListDTO)
showsCourseList()
40
From Requirements to Code in a Snap
Michał Śmiałek

Application logic code






```
class CShowExtendedCourseList
implements IShowExtendedUserList {
// (...)
  wantsToSeeCourseList(){
     ICourseList.builds(cl);
     UI.showsCourseList(cl);
  }
}
```

IShowExtended
CourseList

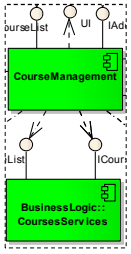
CourseManagement

41 From Requirements to Code in a Snap Michał Śmiałek

Domain logic code

CoursesServices

ICourseList



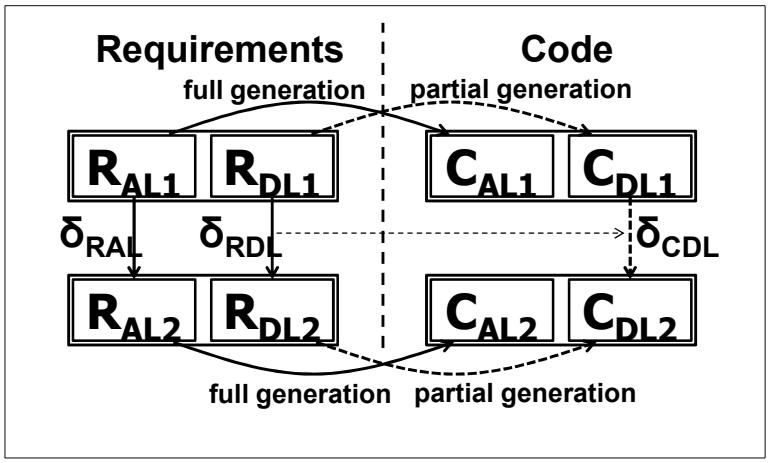

```
class CCourseList implements ICourseList {
// (...)
  void builds(CourseListDTO cld){
    UserListDTO resList;
    for (i=1; i==user_list_length; i++) {
      resList.add(userList[i]);
    }
    cld=resList;
  }
}
```

R1

42 From Requirements to Code in a Snap Michał Śmiałek

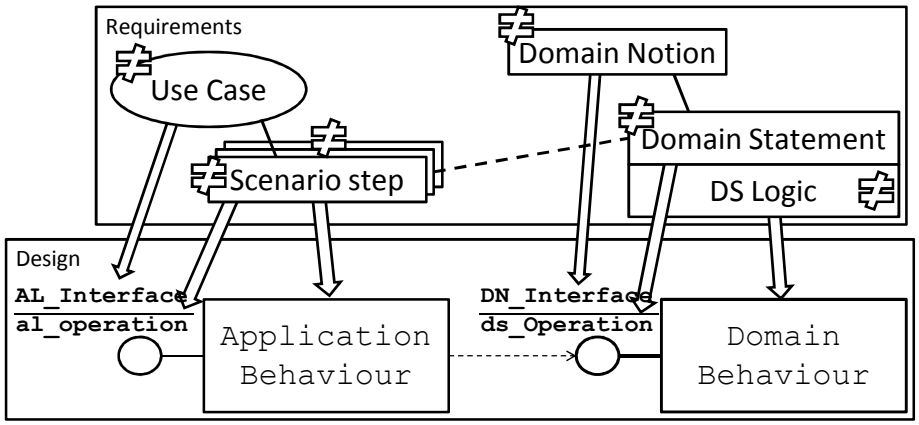
:: SOFTWARE EVOLUTION ::

General evolution schema



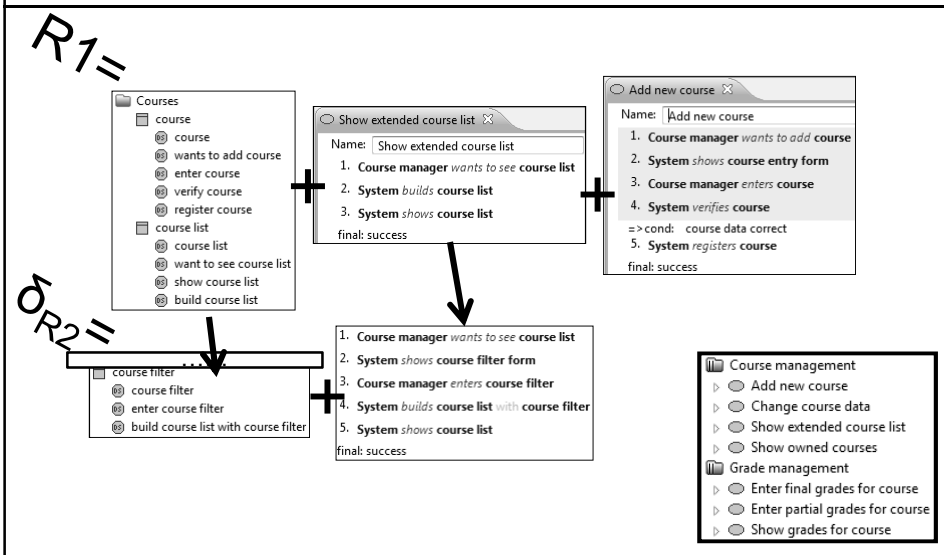
44 From Requirements to Code in a Snap Michał Śmiałek

System changes after requirements changes

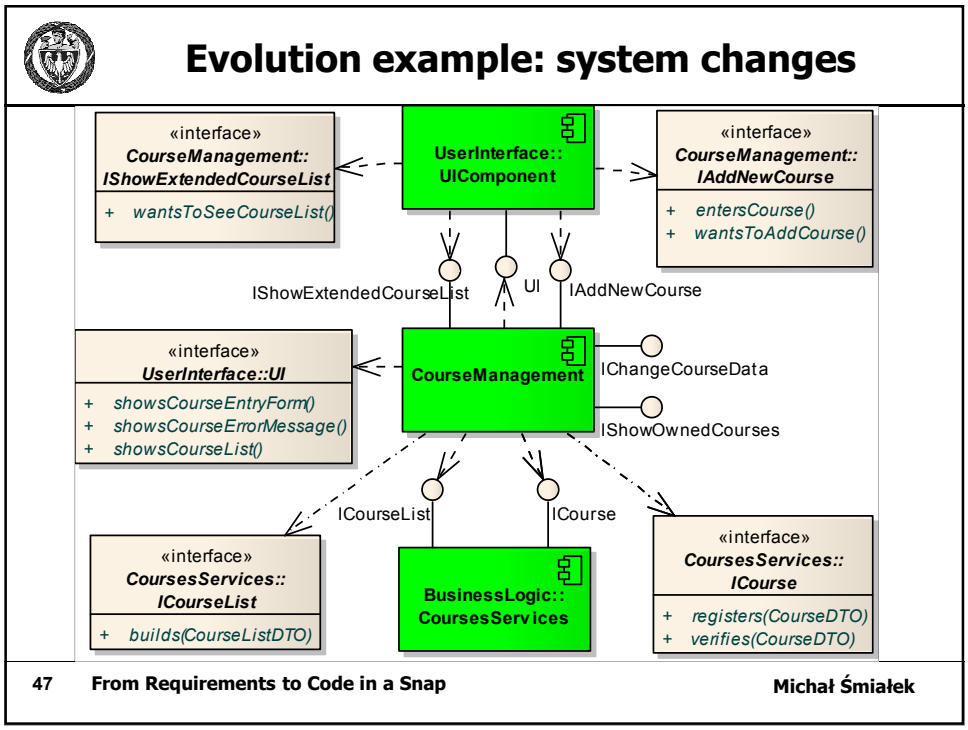


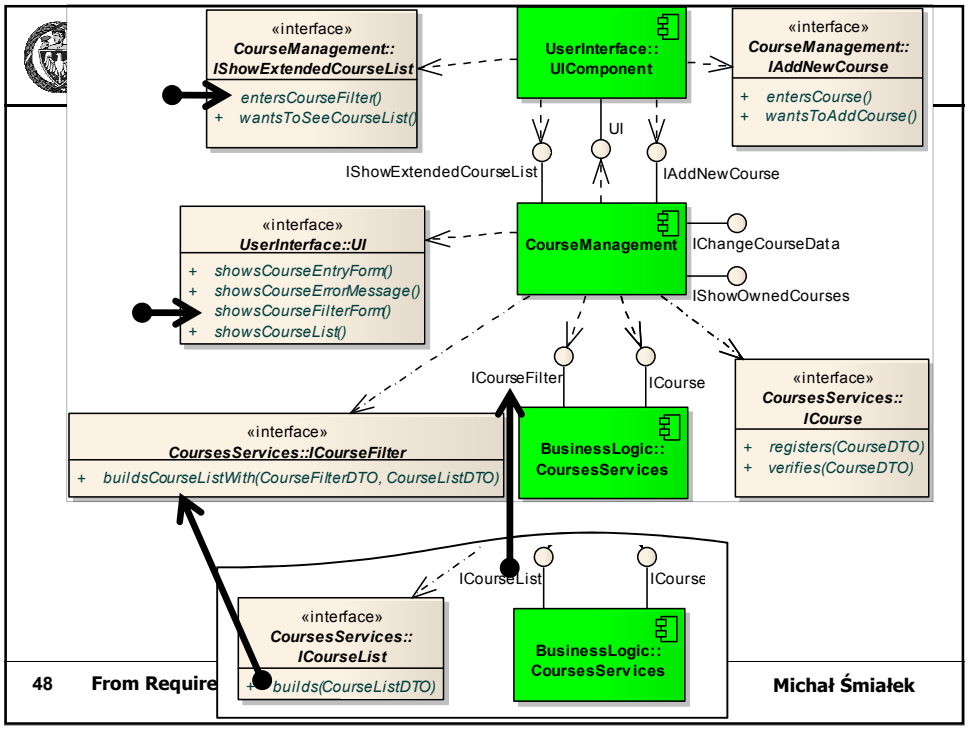
45 From Requirements to Code in a Snap Michał Śmiałek

Evolution example: requirements change

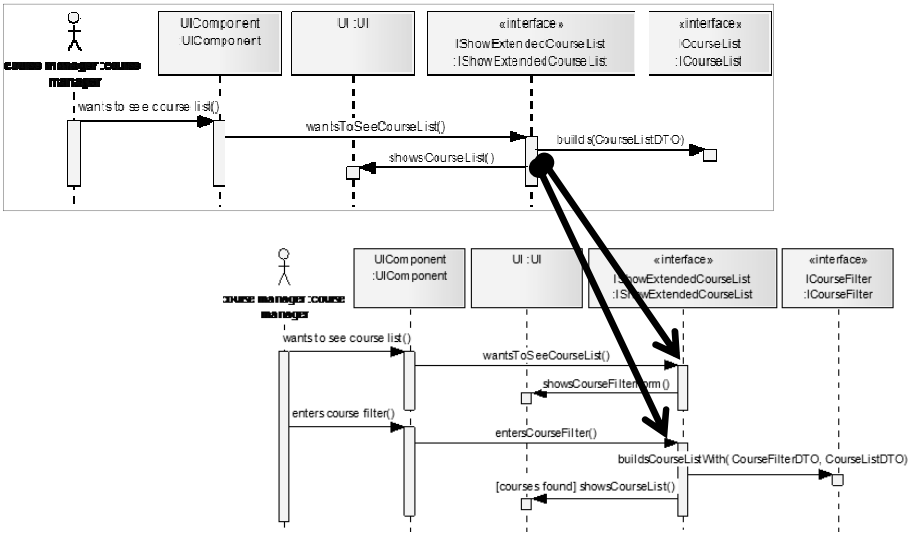


46 From Requirements to Code in a Snap Michał Śmiałek

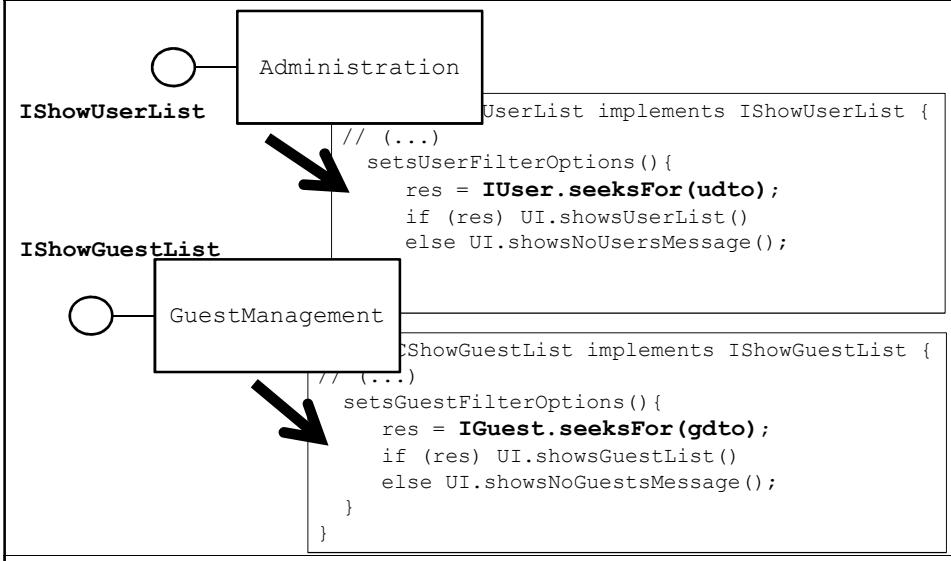

Evolution example: system changes
«interface»
CourseManagement::
IShowExtendedCourseList
+ wantsToSeeCourseList()
UserInterface::
UIComponent
«interface»
CourseManagement::
IAddNewCourse
+ entersCourse()
+ wantsToAddCourse()
IShowExtendedCourseList
UI
IAddNewCourse
«interface»
UserInterface::UI
+ showsCourseEntryForm()
+ showsCourseErrorMessage()
+ showsCourseList()
CourseManagement
IChangeCourseData
IShowOwnedCourses
ICourseList
ICourse
«interface»
CoursesServices::
ICourseList
+ builds(CourseListDTO)
BusinessLogic::
CoursesServices
«interface»
CoursesServices::
ICourse
+ registers(CourseDTO)
+ verifies(CourseDTO)
47
From Requirements to Code in a Snap
Michał Śmiałek

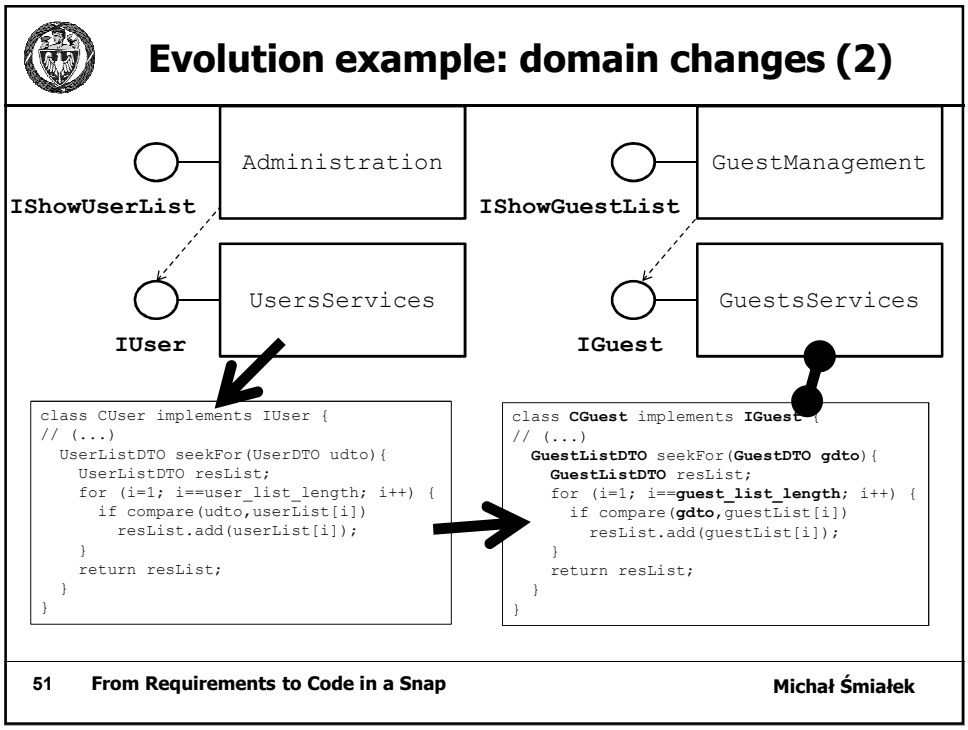

«interface»
CourseManagement::
IShowExtendedCourseList
entersCourseFilter()
+ wantsToSeeCourseList()
UserInterface::
UIComponent
«interface»
CourseManagement::
IAddNewCourse
+ entersCourse()
+ wantsToAddCourse()
UI
IShowExtendedCourseList
IAddNewCourse
«interface»
UserInterface::UI
+ showsCourseEntryForm()
+ showsCourseErrorMessage()
showsCourseFilterForm()
+ showsCourseList()
CourseManagement
IChangeCourseData
IShowOwnedCourses
ICourseFilter
ICourse
«interface»
CoursesServices::
ICourse
+ registers(CourseDTO)
+ verifies(CourseDTO)
«interface»
CoursesServices::ICourseFilter
+ buildsCourseListWith(CourseFilterDTO, CourseListDTO)
BusinessLogic::
CoursesServices
ICourseList
ICourse
«interface»
CoursesServices::
ICourseList
+ builds(CourseListDTO)
BusinessLogic::
CoursesServices
48
From Require
Michał Śmiałek

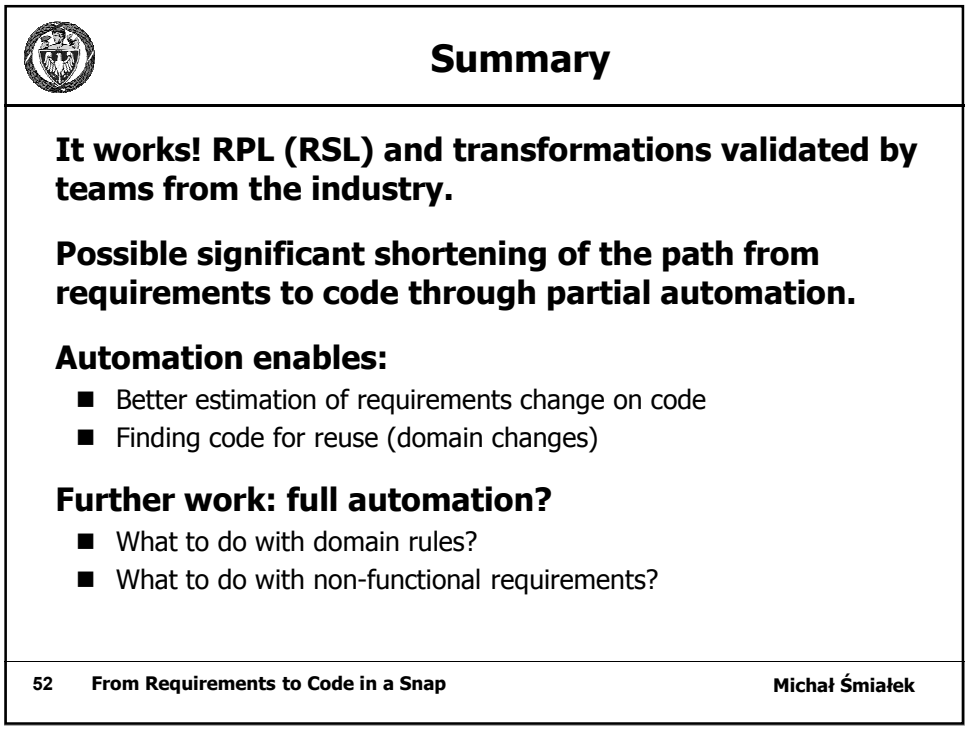
Evolution example: dynamics changes



Evolution example: domain changes (1)




Evolution example: domain changes (2)
Administration
IShowUserList
GuestManagement
IShowGuestList
UsersServices
IUser
GuestsServices
IGuest
class CUser implements IUser {
// (...)
UserListDTO seekFor(UserDTO udto){
UserListDTO resList;
for (i=1; i==user_list_length; i++) {
if compare(udto,userList[i])
resList.add(userList[i]);
}
return resList;
}
}
class CGuest implements IGuest {
// (...)
GuestListDTO seekFor(GuestDTO gdto){
GuestListDTO resList;
for (i=1; i==guest_list_length; i++) {
if compare(gdto,guestList[i])
resList.add(guestList[i]);
}
return resList;
}
}
51 From Requirements to Code in a Snap Michał Śmiałek


Summary
It works! RPL (RSL) and transformations validated by teams from the industry.
Possible significant shortening of the path from requirements to code through partial automation.
Automation enables:
Better estimation of requirements change on code
Finding code for reuse (domain changes)
Further work: full automation?
What to do with domain rules?
What to do with non-functional requirements?
52 From Requirements to Code in a Snap Michał Śmiałek

More info?

Kalnins, A., Barzdins, J., Celms, E.: Basics of model transformation language MOLA. In: Workshop on Model Driven Development (WMDD 2004). (2004)

Smialek, M.: Accommodating informality with necessary precision in use case scenarios. Journal of Object Technology 4(6) (2005) 59-67

Smialek, M., Bojarski, J., Nowakowski, W., Ambroziewicz, A., Straszak, T.: Complementary use case scenario representations based on domain vocabularies. Lecture Notes in Computer Science, MODELS 4735 (2007) 544-558

Smialek, M.: Software Development with Reusable Requirements-Based Cases. Publishing House of the Warsaw University of Technology (2007)

Smialek, M., Kalnins, A., Kalnina, E., et al.: Comprehensive System for Systematic Case-Driven Software Reuse. Lecture Notes in Computer Science, SOFSEM (2010)

Smialek, M.: Software Evolution based on Requirements-Level Programming, Proc. DBI&S Conference, Riga (2010)

ReDSeeDS Project

www.redseeds.eu

Warsaw University of Technology

INFOVIDE-MATRIX

UNIVERSITÄT KOBLENZ · LANDAU

University of Latvia Institute of Mathematics and Computer Science

Cybersoft

TU VIENNA

HERIOT WATT UNIVERSITY

HITeC

algoritmų sistemos

Fraunhofer Institut Experimentelles Software Engineering

PRO DV Software AG

:: THANK YOU FOR YOUR ATTENTION ::

55 **From Requirements to Code in a Snap** **Michał Śmiałek**