SIGCSE 2020
Portland, Oregon, USA

A Modular, Practical Test for a Programming Course

Jan Stolarek
University of Edinburgh, UK
Lodz University of Technology, Poland

Przemysław Nowak
Lodz University of Technology, Poland

https://tinyurl.com/sto-now

Assessing students' programming skills

We want to teach students how to program

https://tinyurl.com/sto-now

Assessing students' programming skills

We want to teach students how to program

...but how do we they learned the skill?

https://tinyurl.com/sto-now

A typical approach

Typical approach:

- individual test in front of a computer
- task: implement a program

https://tinyurl.com/sto-now

Problems with assessment methods

Sounds like the Right Thing, but in practice often implemented in a way that hinders accurate assessment:

https://tinyurl.com/sto-now

Problems with assessment methods

Sounds like the Right Thing, but in practice often implemented in a way that hinders accurate assessment:

- subtasks may not be independent

https://tinyurl.com/sto-now

Problems with assessment methods

Sounds like the Right Thing, but in practice often implemented in a way that hinders accurate assessment:

- subtasks may not be independent
- or the opposite: subtasks correspond to discrete grades, but are independent

https://tinyurl.com/sto-now

Problems with assessment methods

Sounds like the Right Thing, but in practice often implemented in a way that hinders accurate assessment:

- subtasks may not be independent
- or the opposite: subtasks correspond to discrete grades, but are independent
- monolithic all-or-nothing tests with no clear grading structure

https://tinyurl.com/sto-now

Our solution

A modular, practical programming test:

https://tinyurl.com/sto-now

Our solution

A modular, practical programming test:

1. based on learning outcomes

https://tinyurl.com/sto-now

Our solution

A modular, practical programming test:

1. based on learning outcomes
2. enforces minimal requirements

https://tinyurl.com/sto-now

Our solution

A modular, practical programming test:

1. based on learning outcomes
2. enforces minimal requirements
3. skills assessed independently from one another

https://tinyurl.com/sto-now

Our solution

A modular, practical programming test:

1. based on learning outcomes
2. enforces minimal requirements
3. skills assessed independently from one another

Result: accurate, comprehensive, and fair assessment of students' skills

https://tinyurl.com/sto-now

Disclaimer

This is an experience report

https://tinyurl.com/sto-now

Context and scope

Context and scope:

https://tinyurl.com/sto-now

Context and scope

Context and scope:

- OOP course, 3rd semester undergraduate

https://tinyurl.com/sto-now

Context and scope

Context and scope:

- OOP course, 3rd semester undergraduate
- students (assumed to) know basics of C++ from previous courses

Context and scope:

- OOP course, 3rd semester undergraduate
- students (assumed to) know basics of C++ from previous courses
- course duration: 15 weeks

Context and scope:

- OOP course, 3rd semester undergraduate
- students (assumed to) know basics of C++ from previous courses
- course duration: 15 weeks
- each week: a 90-minute lecture + a 90-minute lab session

Context and scope:

- OOP course, 3rd semester undergraduate
- students (assumed to) know basics of C++ from previous courses
- course duration: 15 weeks
- each week: a 90-minute lecture + a 90-minute lab session
- this talk only about lab sessions

Context and scope:

- OOP course, 3rd semester undergraduate
- students (assumed to) know basics of C++ from previous courses
- course duration: 15 weeks
- each week: a 90-minute lecture + a 90-minute lab session
- this talk only about lab sessions
- each lab session in groups of 12-20 students per TA

Learning outcomes

Learning outcomes to be verified by the test:

https://tinyurl.com/sto-now

Learning outcomes

Learning outcomes to be verified by the test:

- Create object-oriented programs in C++ language based on a provided design.

https://tinyurl.com/sto-now

Learning outcomes

Learning outcomes to be verified by the test:

- Create object-oriented programs in C++ language based on a provided design.
- Use object-oriented elements of the C++ Standard Library.

https://tinyurl.com/sto-now

Elaboration of learning outcomes: basic requirements

First, we elaborate the details of learning outcomes. Basic requirements:

Elaboration of learning outcomes: basic requirements

First, we elaborate the details of learning outcomes. Basic requirements:

- create classes and inheritance structure according to an UML class diagram;

Elaboration of learning outcomes: basic requirements

First, we elaborate the details of learning outcomes. Basic requirements:

- create classes and inheritance structure according to an UML class diagram;
- properly initialize objects using constructors;

https://tinyurl.com/sto-now

Elaboration of learning outcomes: basic requirements

First, we elaborate the details of learning outcomes. Basic requirements:

(i) create classes and inheritance structure according to an UML class diagram;

(ii) properly initialize objects using constructors;

(iii) implement methods to read and modify fields of an object or perform other simple operations on it;

Elaboration of learning outcomes: basic requirements

First, we elaborate the details of learning outcomes. Basic requirements:

(i) create classes and inheritance structure according to an UML class diagram;

(ii) properly initialize objects using constructors;

(iii) implement methods to read and modify fields of an object or perform other simple operations on it;

(iv) call methods of an object, both directly and via pointers – this includes calling virtual methods in derived classes via pointer to a base class;

Elaboration of learning outcomes: basic requirements

First, we elaborate the details of learning outcomes. Basic requirements:

(i) create classes and inheritance structure according to an UML class diagram;

(ii) properly initialize objects using constructors;

(iii) implement methods to read and modify fields of an object or perform other simple operations on it;

(iv) call methods of an object, both directly and via pointers – this includes calling virtual methods in derived classes via pointer to a base class;

(v) operate (create/add/remove/iterate) on basic Standard Template Library (STL) collections, like vectors;

Elaboration of learning outcomes: basic requirements

First, we elaborate the details of learning outcomes. Basic requirements:

(i) create classes and inheritance structure according to an UML class diagram;

(ii) properly initialize objects using constructors;

(iii) implement methods to read and modify fields of an object or perform other simple operations on it;

(iv) call methods of an object, both directly and via pointers – this includes calling virtual methods in derived classes via pointer to a base class;

(v) operate (create/add/remove/iterate) on basic Standard Template Library (STL) collections, like vectors;

(vi) incorporate C++ knowledge from the previous semesters (e.g., handling of streams, random number generation, use of unions, enumerations, etc.) into object-oriented programs.

https://tinyurl.com/sto-now

Elaboration of learning outcomes: additional requirements

In addition we wish that students:

https://tinyurl.com/sto-now

Elaboration of learning outcomes: additional requirements

In addition we wish that students:

- use static class components;

https://tinyurl.com/sto-now

Elaboration of learning outcomes: additional requirements

In addition we wish that students:

- use static class components;
- properly allocate and de-allocate memory via pointers (including virtual destructors if necessary);

https://tinyurl.com/sto-now

Elaboration of learning outcomes: additional requirements

In addition we wish that students:

- use static class components;
- properly allocate and de-allocate memory via pointers (including virtual destructors if necessary);
- extend the program with new methods not shown in an UML class diagram, based only on written description of their behavior

https://tinyurl.com/sto-now

Environment

Test environment:

- Windows + GCC + CodeBlocks
- no tools that translate UML to code

https://tinyurl.com/sto-now

Solutions

Solution for a typical test:

https://tinyurl.com/sto-now

Solutions

Solution for a typical test:

1. 80-100 LOC + 50-80 LOC in header files

https://tinyurl.com/sto-now

Solutions

Solution for a typical test:

1. 80-100 LOC + 50-80 LOC in header files
2. 45-75 minutes

https://tinyurl.com/sto-now

Solutions

Solution for a typical test:

1. 80-100 LOC + 50-80 LOC in header files
2. 45-75 minutes

Solution must compile to be accepted!

https://tinyurl.com/sto-now

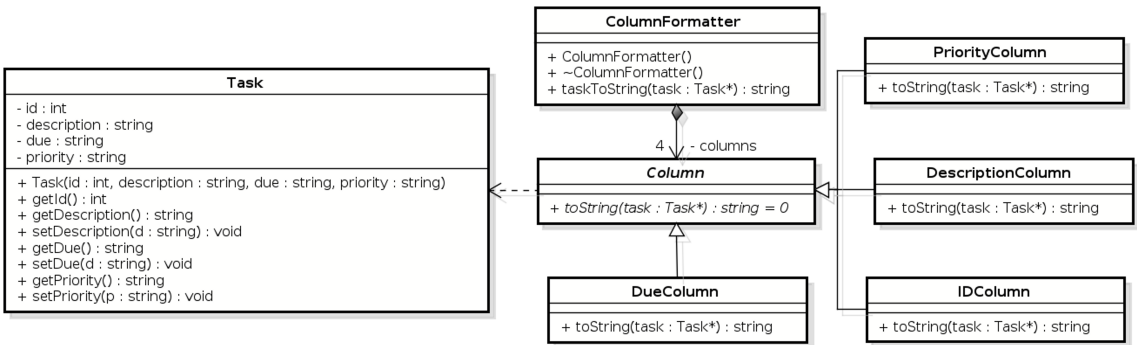
Example test

https://tinyurl.com/sto-now

Example test

Your task is to implement a fragment of a system for managing a TODO list. Your program will store a list of tasks to do and display that list on the screen. See Figure 1 for a class diagram.

https://tinyurl.com/sto-now



https://tinyurl.com/sto-now

Example test

Specification:

https://tinyurl.com/sto-now

Example test

Specification:

A. Create class structure according to the diagram. **(3 points)**

https://tinyurl.com/sto-now

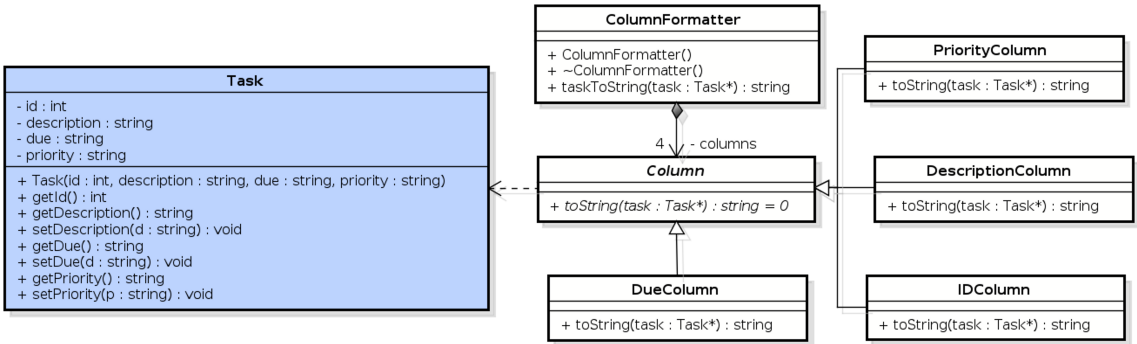
Example test

Specification:

A. Create class structure according to the diagram. **(3 points)**

B1. Implement methods in `Task` class. **(1 point)**

https://tinyurl.com/sto-now



https://tinyurl.com/sto-now

Example test

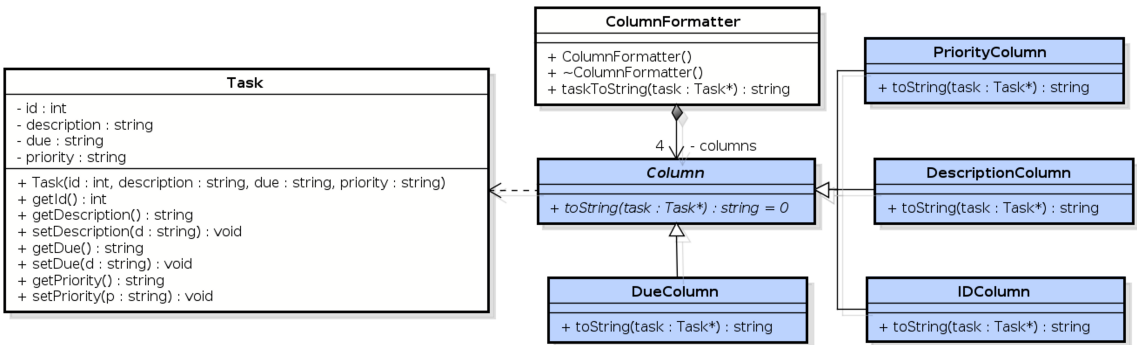
Specification:

A. Create class structure according to the diagram. **(3 points)**

B1. Implement methods in `Task` class. **(1 point)**

C. Implement column formatters via inheritance **(2 points)**

`https://tinyurl.com/sto-now`



https://tinyurl.com/sto-now

Example test

Specification:

A. Create class structure according to the diagram. **(3 points)**

B1. Implement methods in `Task` class. **(1 point)**

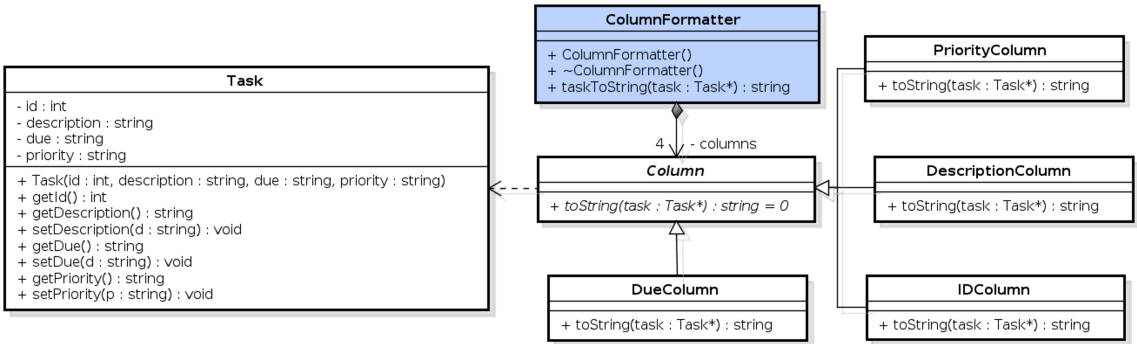
C. Implement column formatters via inheritance **(2 points)**

D1. Implement `ColumnFormatter` class:

(1) create STL collection of column formatters **(1 point)**

(2) iterate over that collection, render columns by calling abstract method **(2 points)**

https://tinyurl.com/sto-now



https://tinyurl.com/sto-now

Example test

Specification:

A. Create class structure according to the diagram. **(3 points)**

B1. Implement methods in `Task` class. **(1 point)**

C. Implement column formatters via inheritance **(2 points)**

D1. Implement `ColumnFormatter` class:

(1) create STL collection of column formatters **(1 point)**

(2) iterate over that collection, render columns by calling abstract method **(2 points)**

E. Create `main` function that constructs example tasks and displays them **(2 points)**

`https://tinyurl.com/sto-now`

Example test

Specification:

A. Create class structure according to the diagram. **(3 points)**

B1. Implement methods in `Task` class. **(1 point)**

C. Implement column formatters via inheritance **(2 points)**

D1. Implement `ColumnFormatter` class:

(1) create STL collection of column formatters **(1 point)**

(2) iterate over that collection, render columns by calling abstract method **(2 points)**

E. Create `main` function that constructs example tasks and displays them **(2 points)**

B2. **(extends B1)** Add a static field to `Task` class. **(2 points)**

`https://tinyurl.com/sto-now`

Example test

Specification:

A. Create class structure according to the diagram. **(3 points)**

B1. Implement methods in `Task` class. **(1 point)**

C. Implement column formatters via inheritance **(2 points)**

D1. Implement `ColumnFormatter` class:

(1) create STL collection of column formatters **(1 point)**

(2) iterate over that collection, render columns by calling abstract method **(2 points)**

E. Create `main` function that constructs example tasks and displays them **(2 points)**

B2. **(extends B1)** Add a static field to `Task` class. **(2 points)**

D2. **(extends D1)** De-allocate memory in `ColumnFormatter` destructor **(2 points)**

`https://tinyurl.com/sto-now`

Example test

Specification:

A. Create class structure according to the diagram. **(3 points)**

B1. Implement methods in `Task` class. **(1 point)**

C. Implement column formatters via inheritance **(2 points)**

D1. Implement `ColumnFormatter` class:

(1) create STL collection of column formatters **(1 point)**

(2) iterate over that collection, render columns by calling abstract method **(2 points)**

E. Create `main` function that constructs example tasks and displays them **(2 points)**

B2. **(extends B1)** Add a static field to `Task` class. **(2 points)**

D2. **(extends D1)** De-allocate memory in `ColumnFormatter` destructor **(2 points)**

D3. **(extends D1)** Extend `Column` class to return column description, use that method to display column headers. **(3 points)** `https://tinyurl.com/sto-now`

Grading scale

Grading scale:

5

4.5

4

3.5

3

2 (fail)

https://tinyurl.com/sto-now

Grading scale

Grading scale:

- 5 - 17-18 points
- 4.5 - 16 points
- 4 - 14-15 points
- 3.5 - 13 points
- 3 - 11-12 points
- 2 (fail) - 0-10 points

https://tinyurl.com/sto-now

Modularity

Req#	**A** (3pts)	**B1** (1pt)	**C** (2pts)	**D1** (1pt+2pts)	**E** (2pts)	**B2** (2pts)	**D2** (2pts)	**D3** (3pts)
(i)	✓							
(ii)		✓		✓				
(iii)		✓	✓					✓
(iv)				✓	✓			✓
(v)				✓	✓		✓	✓
(vi)				✓	✓			✓
(vii)						✓		
(viii)							✓	
(ix)								✓

https://tinyurl.com/sto-now

Modularity

Req#	**A** (3pts)	**B1** (1pt)	**C** (2pts)	**D1** (1pt+2pts)	**E** (2pts)	**B2** (2pts)	**D2** (2pts)	**D3** (3pts)
(i)	✓							
(ii)		✓		✓				
(iii)		✓	✓					✓
(iv)				✓	✓			✓
(v)				✓	✓		✓	✓
(vi)				✓	✓			✓
(vii)						✓		
(viii)							✓	
(ix)								✓

All possible combinations of tasks that allow achieving a passing grade must test all the basic requirements!

https://tinyurl.com/sto-now

Modularity

Req#	**A** (3pts)	**B1** (1pt)	**C** (2pts)	**D1** (1pt+2pts)	**E** (2pts)	**B2** (2pts)	**D2** (2pts)	**D3** (3pts)
(i)	✓							
(ii)		✓		✓				
(iii)		✓	✓					✓
(iv)				✓	✓			✓
(v)				✓	✓		✓	✓
(vi)				✓	✓			✓
(vii)						✓		
(viii)							✓	
(ix)								✓

For example: A + B1 + C + D1 + E = 11 points

https://tinyurl.com/sto-now

Modularity

Req#	**A** (3pts)	**B1** (1pt)	**C** (2pts)	**D1** (1pt+2pts)	**E** (2pts)	**B2** (2pts)	**D2** (2pts)	**D3** (3pts)
(i)	✓							
(ii)		✓		✓				
(iii)		✓	✓					✓
(iv)				✓	✓			✓
(v)				✓	✓		✓	✓
(vi)				✓	✓			✓
(vii)						✓		
(viii)							✓	
(ix)								✓

But also: A + D1 + D2 + D3 = 11 points

https://tinyurl.com/sto-now

Grading process

Grading process:

- not possible to test programs by running them
- inspection of source code, no automated grading tools
- uniform structure of all solutions; a single one takes 3-4 minutes to grade, so about an hour for a lab group

https://tinyurl.com/sto-now

Practical analysis

75 students over course of 3 years, over 120 individual tests

https://tinyurl.com/sto-now

Question 1: *Do students take advantage of test modularity? Do they skip subtasks they do not know how to do and focus on the ones they can do?*

https://tinyurl.com/sto-now

Question 1: *Do students take advantage of test modularity? Do they skip subtasks they do not know how to do and focus on the ones they can do?*

Answer: Yes. Many students with lower grades do exactly that.

https://tinyurl.com/sto-now

Question 1: *Do students take advantage of test modularity? Do they skip subtasks they do not know how to do and focus on the ones they can do?*

Answer: Yes. Many students with lower grades do exactly that.

Caveat: make it clear to the students that this is what they are expected to do.

https://tinyurl.com/sto-now

Question 1: *Do students take advantage of test modularity? Do they skip subtasks they do not know how to do and focus on the ones they can do?*

Answer: Yes. Many students with lower grades do exactly that.

Caveat: make it clear to the students that this is what they are expected to do.

Numbering of subtasks indicates dependencies.

Question 2: *Does the test enforce the course learning outcomes? Do students that pass it demonstrate basic skills (i)-(vi)?*

https://tinyurl.com/sto-now

Question 2: *Does the test enforce the course learning outcomes? Do students that pass it demonstrate basic skills (i)-(vi)?*

Answer Yes. There were no cases where a student would pass a test without fulfilling all the basic requirements.

Question 3: *Do students feel they are being judged fairly?*

https://tinyurl.com/sto-now

Question 3: *Do students feel they are being judged fairly?*

Answer Yes. They more often attribute their performance on the test to their knowledge than to external factors.

Summary

To summarise:

https://tinyurl.com/sto-now

Summary

To summarise:

- a simple idea, but only in hindsight

https://tinyurl.com/sto-now

Summary

To summarise:

- a simple idea, but only in hindsight
- a product of refinement over a course of several years

https://tinyurl.com/sto-now

Summary

To summarise:

- a simple idea, but only in hindsight
- a product of refinement over a course of several years
- this approach is more of a validation method than a recipe; relies on expert knowledge

https://tinyurl.com/sto-now

Summary

Future work:

https://tinyurl.com/sto-now

Summary

Future work:

- generalisation to a broader scope

https://tinyurl.com/sto-now

Summary

Future work:

- generalisation to a broader scope
- making intuitions presented in this work more precise?

https://tinyurl.com/sto-now

Summary

More in the paper:

https://tinyurl.com/sto-now

Summary

More in the paper:

- detailed presentation and discussion of learning outcomes

https://tinyurl.com/sto-now

Summary

More in the paper:

- detailed presentation and discussion of learning outcomes
- structure of lab sessions

https://tinyurl.com/sto-now

Summary

More in the paper:

- detailed presentation and discussion of learning outcomes
- structure of lab sessions
- full text of the test

https://tinyurl.com/sto-now

Summary

More in the paper:

- detailed presentation and discussion of learning outcomes
- structure of lab sessions
- full text of the test
- more details of practical analysis

https://tinyurl.com/sto-now

Summary

More in the paper:

- detailed presentation and discussion of learning outcomes
- structure of lab sessions
- full text of the test
- more details of practical analysis
- validity assessment

https://tinyurl.com/sto-now

Please email questions to
jan.stolarek@ed.ac.uk

https://tinyurl.com/sto-now

SIGCSE 2020
Portland, Oregon, USA

A Modular, Practical Test for a Programming Course

Jan Stolarek
University of Edinburgh, UK
Lodz University of Technology, Poland

Przemysław Nowak
Lodz University of Technology, Poland

https://tinyurl.com/sto-now