Reference group's feedback to course coordinator

Students participating in reference group. Name and study programme:

- **Elling Bjørnstad Vågan, Bachelor in programming**
- **Malin Foss, Bachelor in programming**
- **Arnaud Duhamel, Bachelor in programming**

Short summary of the dialogue between the reference group and course coordinator/teacher and with the other students throughout the course:

The reference group held 3 meetings at the following dates and time:

- On September 21, 2022, from around 14:00 to 14:50
- On October 26, 2022, from around 14:05 to 14:50
- On January 18, 2023, from around 10:00 to 11:00

The professor showed himself very forthcoming, receptive and eager to get input from the reference group. 2 members of the reference group were students retaking the course. And this edition of the course saw significant changes from the last edition, and so the professor was very interested in their input on how the current edition was compared to the previous one.

And not only with the reference group, but also with the whole class. At the beginning of each and every lecture, the professor solicited feedback from the whole class and asked how far they had got in the labs, how difficult it was, if there were struggles, what they were, and then providing advice and adjusting his lectures in consequence on that feedback. In that sense, the professor displayed a huge amount of flexibility and adaptability to the class's needs. And this shows how well he knew what he was teaching. This was on a week-by-week basis.

Something in common from the feedback of both the reference group and the class was guidance. Guidance in all its forms: more lab time with individual help, something that the reference group considers very much needed, as will be detailed in the next section,

more explanations in the lab descriptions, more tutorials or more reference to tutorials, more examples, and more source code.

The reference group also solicited the feedback of the whole class by making available an anonymous survey through which they could share anything they wanted about the course to the reference group. One common thread came out of that consultation: more guidance.

The question asked and the answers received from that survey are attached to this report in appendix 1.

It is worth mentioning that this issue has been addressed between the last edition of the course and this one.

In the previous edition of the course, labs were not related to the assignment, as such that working on the labs was not a direct preparation for the assignment. In this edition of the course, an engine was created throughout the labs of the semester and the applications that were made during those labs were the building blocks on which the assignment was built. And so the assignment was a direct continuation of the labs. It was also the case for the exam. Although the exam was more complex than the assignment, it used the same building blocks that were created in the labs; the exam was also a direct continuation of the labs. This, in and of itself, was more guidance.

This was a significant change from the previous editions, and it involved coming up with a whole new assignment and changes to the exam. This change was a big reform of the course and significantly improved it compared to the previous edition.

Also, in the previous edition of the course, no direct guidance was provided on creating an engine, meaning encapsulating raw OpenGL code in more user-friendly classes.

In this edition of the course, the labs offered extensive descriptions, helpful pieces of source code and many references to external resources and relevant video tutorials on how to create the engine. In short, the labs were significantly more detailed and guiding then in the previous edition of the course. This change was also a big reform of the course and significantly improved it.

And lastly, in this edition of the course, all of the source code related to CMake, a tool to create cross-platform applications with the C++ language, was provided to the class,

something that was not done in the previous edition. This also made the course significantly easier.

Those 3 significant reforms of the course directly addressed this need of guidance and showed how much the professor was not only receptive, but dedicated to this issue long before it was expressed in this edition of the course. The professor can and should be proud of that; this made the course better.

In addition to that, throughout the whole semester, the professor offered additional guidance to the class according to what was expressed at the beginning of every lecture.

Adding all those cases of additional guidance together, the professor gave a significant amount of additional guidance to the students throughout the course, on top of the increase of guidance that was planned for this edition of the course.

So, in short, the professor has been absolutely receptive to the core need expressed by students: guidance. Definitely something the reference group is proud of.

The reference group's assessment of the learning environment in the course

By 'learning environment', we mean: "the totality of physical conditions, plans, curriculum, organisation of teaching, working methods, social relations and attitudes to learning - i.e., all factors that can affect students' learning."[1]

Regarding this aspect, the reference group would like to refer to this table presenting the failing rate in every course that the 2021 cohort of the bachelor in programming had so far, in decreasing order:

[1] Skaalvik, E.M. & Skaalvik S. (2005) Skolen som læringsarena. Selvoppfatning, motivasjon og læring. Oslo, Universitetsforlaget. https://www.universell.no/lmu/lmu-haandbok/hva-er-et-laeringsmiljoe/.

Course Name	fail_rate
Graphics Programming	52 %
Web Technologies	33 %
Object-oriented Programming	32 %
Mathematics for Computer Science	31 %
Operating Systems	28 %
Algorithmic Methods	27 %
Mathematics for Programming	26 %
Fundamental Programming	19 %
Introduction to User-Centered Design	7 %
Examen philosophicum for Science and Technology	6 %
Cyber security and computer networks	4 %
Software Development	0 %

Another version of this table with more information is provided in appendix 3.

All the data taken from Studentweb to create this table is also provided in appendix 4.

By a huge margin, this course is the most failed course so far for the 2021 cohort.

The reference group would also like to refer to the following table presenting the amount of teaching time per week against the failing rate, for this course and a sample of the other courses presented above. The table is sorted by the weekly teaching time in decreasing order. The same table with the breakdown between the lecture and lab time per week is also provided in appendix 5. The data taken from Timeplan to create this table is provided in appendix 6.

Course	teach time (lab & lectures/week)	fail_rate
Mathematics for Computer Science	8	31 %
Software Development	6	0 %
Cyber security and computer networks	6	4 %
Graphics Programming	3	52 %

The graphics course is, by a huge margin, the most failed course of the 2021 cohort so far and it is, by a huge margin, the course in which the least teaching time is invested.

This, in the eyes of the reference group, is the most important aspect to address: to increase both the lecture time to at least 3 hours a week, and to increase the lab time to at least 5 hours a week.

During the labs for this edition of the course, the professor was solicited from the very beginning of the lab to the very end: there is clearly a need of more brain power during the labs to provide individual help to students.

In that sense and given the difficulty to recruit teaching assistants, the reference group recommends inserting in the course a PhD student in graphics programming that is doing his PhD on a 4 years contract with a teaching clause, on top of having student teaching assistants.

The PhD student could be tasked with coming up with examples, exercises, assignments and exams every year, and to work on the labs and create video tutorials. Exams and exercises could be the same every year.

The reference group has been told that the course would be moved back to the fifth semester of the bachelor in programming.

In light of that, the reference group would like to warn against thinking that because students will be in their fifth semester, it will be easier, and so the current format of the course could be kept to 1 hour of lecture and 2 hours of lab every week with only the professor giving help to students during the lab.

The world of OpenGL is a whole different world that abides by its own principles and rules. And it’s nothing like anything learned before in the bachelor. It’s completely new, it is complex, it is highly prone to error because it requires a lot of coding before actually being able to test code, and it is hard to find errors, even with error detecting functions in the code.

Just learning where to look for errors, learning to debug and to use Renderdoc is something that students should be thoroughly thought for many hours. In this edition of the course, one hour of lecture was allocated to debugging. Shockingly not enough.

And on top of that, CMake is added, something completely new. Software design is also brought in with the creation of the engine. Up to this point, the biggest application created by students is the project in the object-oriented programming course, where students must use Git, many files, make many classes and use inheritance. And there is a pretty wide gap between this and the engine to be created in OpenGL. And, new Git workflows have to be learned. And that is creating branches and merging. And lastly, for the graphics programming itself, 3D geometry has to be used, which is not a huge aspect, but still something that students must spend a significant amount of time understanding and learning. But on that aspect the BMA1020 – Mathematics for Programming, provides a good preparation.

Other than a more intense use of Git, courses in the fourth semester do not appear to specifically prepare students for what is mentioned above.

And, in light of all of the above, the reference group highly recommends to bring the course to at least 3 hours of lectures every week and at least 5 hours of lab every week and that a PhD student in graphics be brought in the course, in addition to teaching assistants, even if the course is moved back to the fifth semester of the bachelor in programming.

The reference group's feedback on the teaching and learning activities, and the form(s) of assessment

By learning outcome, we mean the knowledge and skills the students have acquired by having completed and passed the course.

In order to properly address this aspect, a review of the course description has been made. The official course description at the time of this report is provided in appendix 7.

When it comes to the theoretical knowledge part, it is listed as follows:

-Listing and describing the components of the OpenGL graphics pipeline.

-Understanding the mathematical foundations of geometric transformations in computer graphics.

-Handling textures to display image-based content.

-Applying illumination in 2D/3D scenes.

-OpenGL API programming

-GLSL Shaders programming.

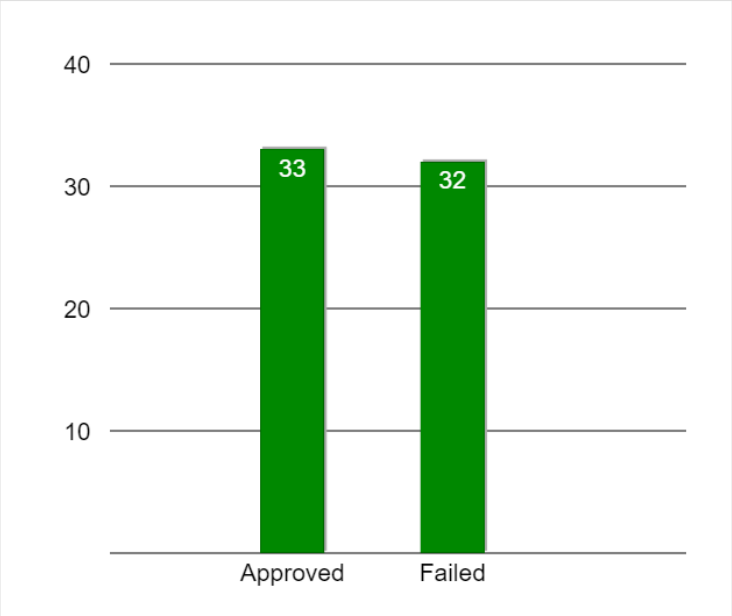
All of those points listed above were thought during the lectures, for as much as it can be thought in 1 hour of lecture per week, which is too little.

It may also be worth considering putting in place a small theoritecal exam in the course, especially in the wake of ChatGPT. Nothing big. It should be kept to a simple multiple-choice exam. Because the idea here is that it should not possible to pass the course without having a basic understanding of OpenGL mechanics.

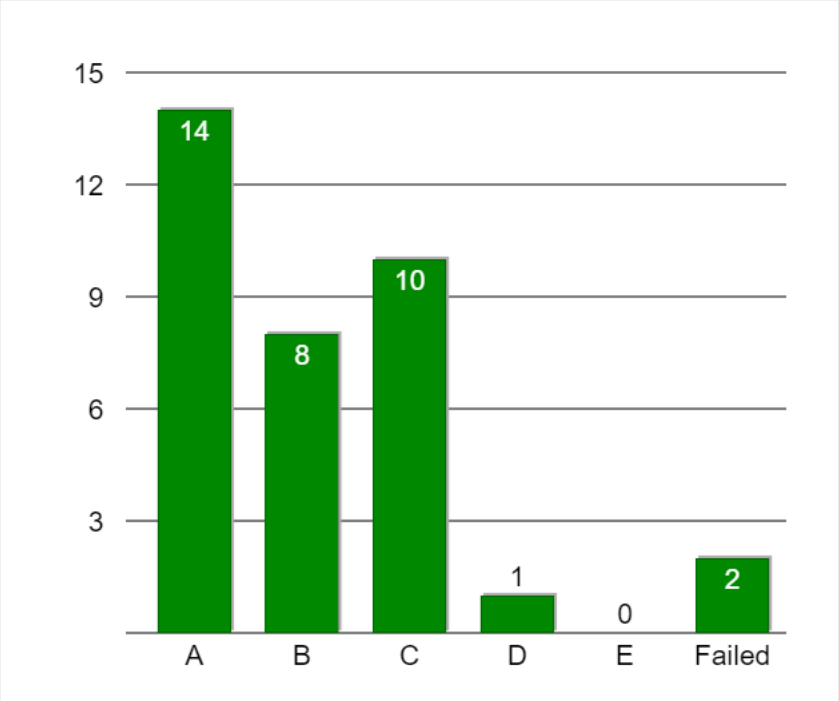
But it really matters to the utmost that this potential exam remains secondary to learning how to program OpenGL. The exam should not be worth more than 30% of the final grade and it should be kept to a simple and straight-forward multiple-choice exam (multiple choice exams can be made to be pretty difficult). Carefully going through the labs and taking the time to understand what is going on in the code when going through the lab should be all that students have to do to have a good grade in this exam. What we want to avoid here is that students go through the course by blindly copying and pasting code or blindly using code given by previous students.

Before getting into the skills and competence that students are expected to acquire throughout the course, it is worth examining in a bit greater detail the failing rate of both the assignment and the exam, as well as its distribution:

Statistics for PROG2002, 2022-12, Mandatory programming assignments



Statistics for PROG2002, 2022 AUTUMN, Home exam



Most of the failing took place at the project. Many students failed the project.

However, during the last reference group meeting, it was mentioned that students that failed the project either handed in absolutely nothing, or very little.

Those students most likely gave up or fell off.

And, it is possible to see that the vast majority of the students that passed the assignment got good grades at the exam.

In the last reference group meeting, it was mentioned that the grade distribution was the same as for last, before the major reforms of the course. But it is important to mention that the reforms significantly improved the course nonetheless. This is the right direction to take.

This, with the fact that the students that failed the project were mostly students that either handed in nothing or very little indicates that students that failed the project are students that gave up or fell off while making the engine.

The most difficult part of creating the engine was to make a chessboard, or what is also called a grid, which is a basic 3D graphics structure. The task was provided in lab 3, in the third week, and no guidance at all was provided on how to create it.

The elements above appear to indicate that the students that failed the course are students that gave up or fell behind very early in the semester.

They also appear to indicate that the most difficult part of the course is creating the engine.

Is this aligned with the learning outcomes of the course? They are listed below for convenience:

Skills:

- Creating procedural animations.
- Utilizing 3D models (loading, onscreen display).
- Creating and manipulating lighting in a 3D scene.
- Using OpenGL for rendering 3D environments.

General Competences:

- Finding solutions to a defined problem orally, and answer question about the solution.
- Using academic material from various online sources in an independent way.
- Improved software development ability.
- Reinforce version control, programming and code analysis.

Those elements appear to confirm that, indeed, one of the core skills that students are supposed to learn in the course is how to use OpenGL, how to program with it. And programming with OpenGL is done with an engine. But then this also implies that the course should actively be teaching students how to program OpenGL and how to create then engine. Which is why the professor is definitely taking the course in the right direction.

The course should step even further in that direction: with lectures, examples and exercises on how to program OpenGL and to create the engine. This should in fact become the main focus of the course.

In the last reference group meeting, the professor mentioned his desire to provide more examples and small OpenGL applications in the next edition of the course. The reference group definitely supports this measure.

The reference understands that the professor can only do so much in the time allocated and can only produce so much material, which why it is recommending sharply increasing the lecture and lab time and to bring in a PhD student, in addition to teaching assistants.

What it boils down to is for NTNU to invest the time and resources necessary so that what is a hard subject can be made as easy as possible to learn. 1 hour of lectures, 2 hours of labs per week and only one professor to produce examples, exercises, labs, assignments and exams is by all means vastly insufficient. The fact that the same 3D Pacman project had been reused for years in the course appears to attest to that.

The professor raised the prospect of having mandatory hand ins in the next edition of the course. This is something that the reference group supports. Because If teaching

how to program OpenGL is to become the main focus of the course, building the engine should also become something that is evaluated. But the reference group supports this measure only on the basis and the assumption that students will be actively thought how to code OpenGL and thought how to create the engine, with lectures, code examples and exercises. Those assignments could be handed in through GitLab, and so students could be initiated early to using branches and merges.

Recommendations of the reference group for the next editions of the course

Based on all of the above, the reference group makes the following recommendations for the next editions of the course, wether or not the course is moved to the 5th semester of the bachelor in programming, in order of importance:

1) Increase the lab time of the course to at least 5 hours per week.

2) Increase the lecture time of the course to at least 3 hours per week.

3) Bring in the course a PhD student in graphics programming that could be tasked with helping students during the labs, creating the examples, the exercises, the assignment, the project and the exam.

4) Make teaching how to code OpenGL, especially the creation of the engine, a complete teaching subject matter, dedicating to it lectures, lab time, code examples and exercises.

5) Putting in place mandatory hand ins related to the creation of the engine, only and only if the recommendation above is implemented. And that students be given a possibility to resubmit within a week if the hand in is failed.

6) Put in place a simple, straight-forward multiple choice theoretical exam that would not be worth more than 30% of the final grade.

Appendix 1

Survey from the reference group to the class

&

Answers

Tilbakemelding PROG2002 H22

Sign in to Google to save your progress. Learn more

Gjerne si alt du ønsker om PROG2002 - Grafikk programmering / here you can share anything you would like about the course PROG2002 - Graphics programming

Your answer

Submit Clear form

Hello world.

Synes det er for lite med info gitt om hvordan man skal løse laboppgavene. Skulle gjerne hatt fler eksempler på hvordan koden skal se ut

Vanskelig å forstå innhold i lab'ene og det oppstår ofte problemer som er svært vanskelig å løse uten veiledning

please share src code so that we as students can study what you want us to do. While i am able to do some things i believe that being able to see the "correct" code would be beneficial.

føles som jeg står helt stille i faget når han aldri viser oss kodesnutter/ gir oss løsningsforslag

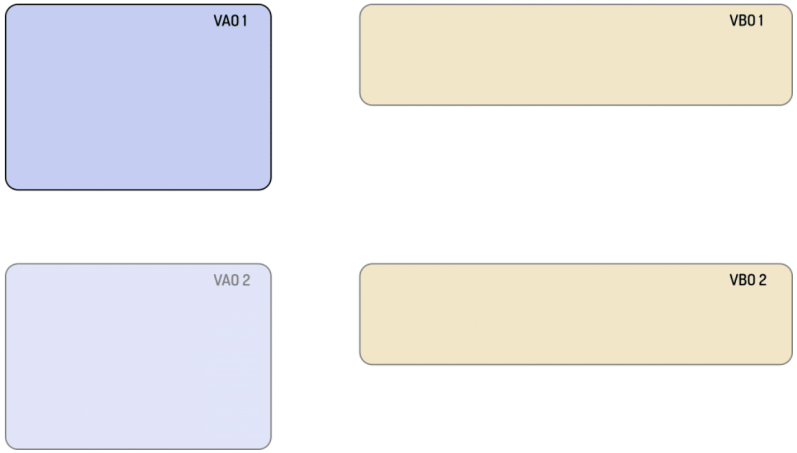
I think the jumps in complexity from week to week are large. For me, the abstraction part is a big problem to overcome. I understand that we can't just get the solution, but some more guidance in the start would be helpful. Yes, we have larned about abstraction, but intuitively understanding how to abstract a graphics engine in a sensible way I feel is a bit much. I think it has been a bit like "here are some tools, now go design and build a house". Right now I am stressed because I am falling behind, but I think the course will be fun once I understand how to structure the abstraction.

Appendix 2

Sample of lecture slides containing source code

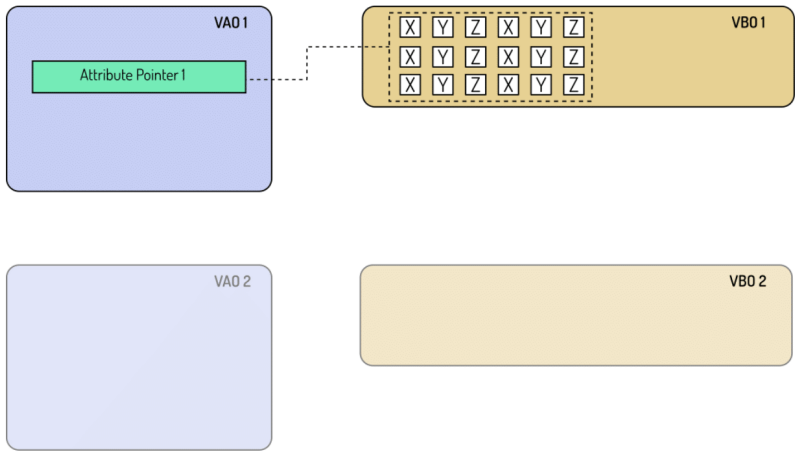
Vertex Array Objects (VAO)

```
GLfloat triangles[18] = {


    -0.5f, -0.5f, 0.0f, 0.5f, 0.5f, 1.0f,
     0.5f, -0.5f, 0.0f, 0.5f, 0.5f, 1.0f,
     0.0f,  0.5f, 0.0f  0.0f, 0.5f, 1.0f
};

// generate Vertex Arrays
GLuint vaos[2];
glGenVertexArrays(2, vaos);

// generate Vertex Buffers
GLuint vbos[2];
glGenBuffers(2, vbos);

// bind first vao
glBindVertexArray(vao[0]);
```



Vertex Array Objects (VAO)

```
1
2  // make the vertex buffer active
3  glBindBuffer(GL_ARRAY_BUFFER,  // type of buffer
4               vertexbufferid);  // id of the target buffer
5
6  // transfer the vertices data to the gpu
7  glBufferData(GL_ARRAY_BUFFER,  // type of buffer
8               sizeof(triangle), // size of buffer (in bytes)
9               triangle,         // pointer to buffer
10              GL_STATIC_DRAW);  // use of the buffer
11
12 glVertexAttribPointer(0,           // Attribute index
13                        3,           // components
14                        GL_FLOAT,    // data type
15                        GL_FALSE,    // normalize data?
16                        4*3,         // stride
17                        (const void *)0);    // ptr to 1st elem
18
19 glEnableVertexAttribArray(0);   // Enable attribute 0
```



Vertex Shaders

```
#version 460 core

layout (location = 0) in vec4 a_Position;

void main()
    {
        gl_Position = a_Position;
    }
```

- Vertex shaders process individual vertices:
  - Set final position of vertex.
  - Set size of point.
- Executed once per vertex on the GPU.

Fragment Shaders

```
#version 460 core

out vec4 Color;

void main()
    {
        Color = vec4(1.0);
    }
```

- Fragment shaders process individual fragments (“pixels”)
  - Assigns color values to different segments of the primitives to be rendered
- Executed once per fragment on the GPU. This is many more times than vertices
- Important to pay attention to optimization considerations

Appendix 3

Table presenting the failing rate in the courses taken by the 2021 cohort of the bachelor in programming up to the date of this report, in decreasing order

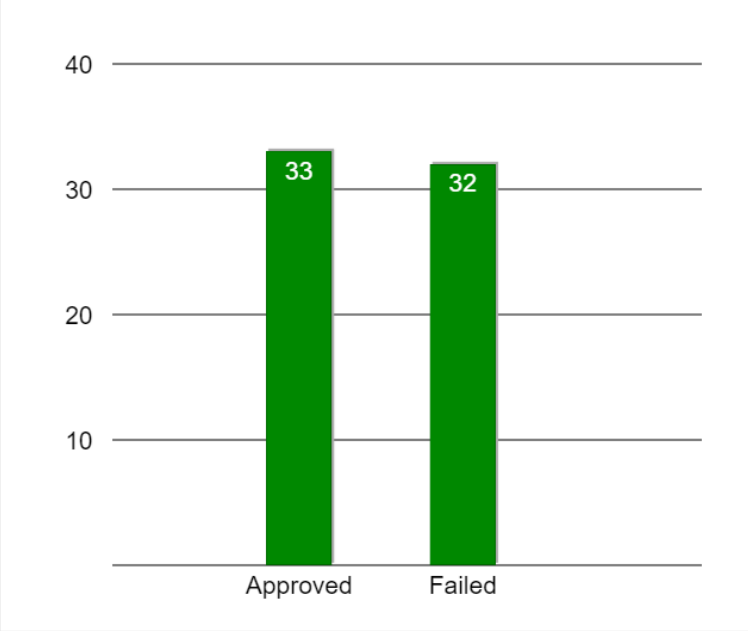
Year	Semester	Course code	Course Name	num_students	num_fail	fail_rate
2022	AUTUMN	PROG2002	Graphics Programming	65	34	52 %
2022	AUTUMN	PROG2053	Web Technologies	146	48	33 %
2022	SPRING	PROG1003	Object-oriented Programming	180	58	32 %
2021	AUTUMN	BMA1010	Mathematics for Computer Science	144	45	31 %
2022	AUTUMN	IDATG2202	Operating Systems	156	44	28 %
2022	AUTUMN	IDATG2102	Algorithmic Methods	124	33	27 %
2022	SPRING	BMA1020	Mathematics for Programming	74	19	26 %
2021	AUTUMN	PROG1001	Fundamental Programming	151	29	19 %
2021	AUTUMN	IDG1362	Introduction to User-Centered Design	206	14	7 %
2022	SPRING	EXPH0300	Examen philosophicum for Science and Technology	987	64	6 %
2021	AUTUMN	IIKG1001	Cyber security and computer network	101	4	4 %
2022	SPRING	PROG1004	Software Development	134	0	0 %

Appendix 4

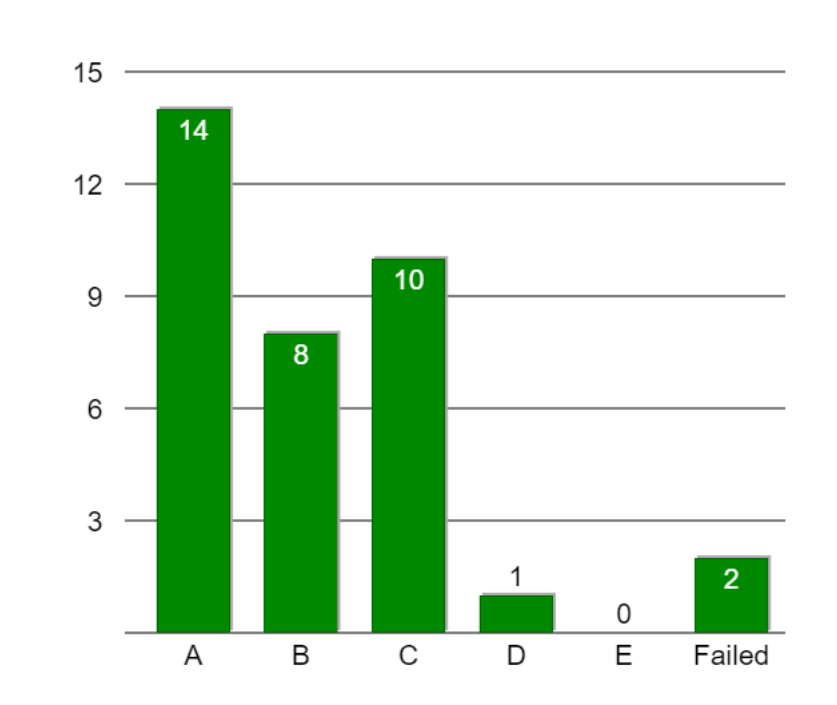
Data from Studentweb used to produce the table in appendix 3

PROG2002 - Graphics Programming

Statistics for PROG2002, 2022-12, Mandatory programming assignments

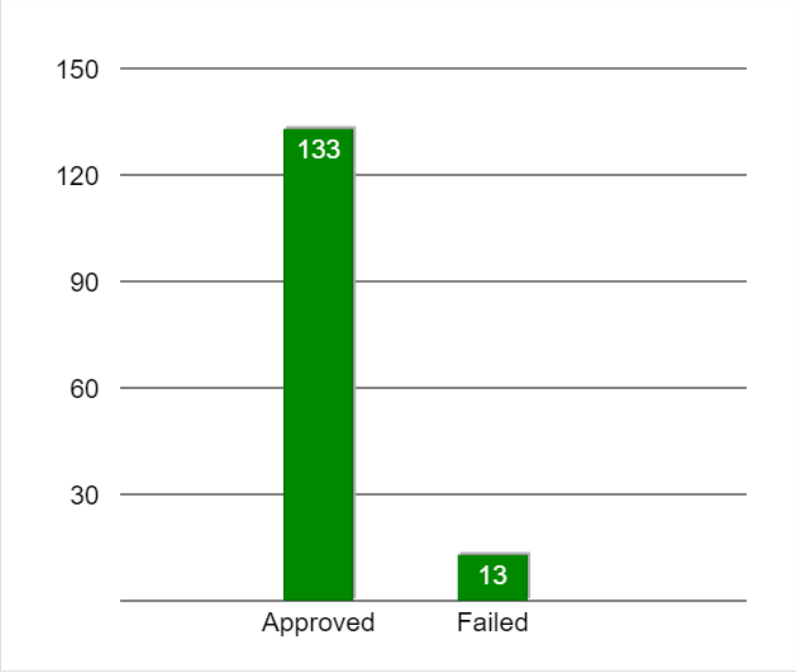


Statistics for PROG2002, 2022 AUTUMN, Home exam

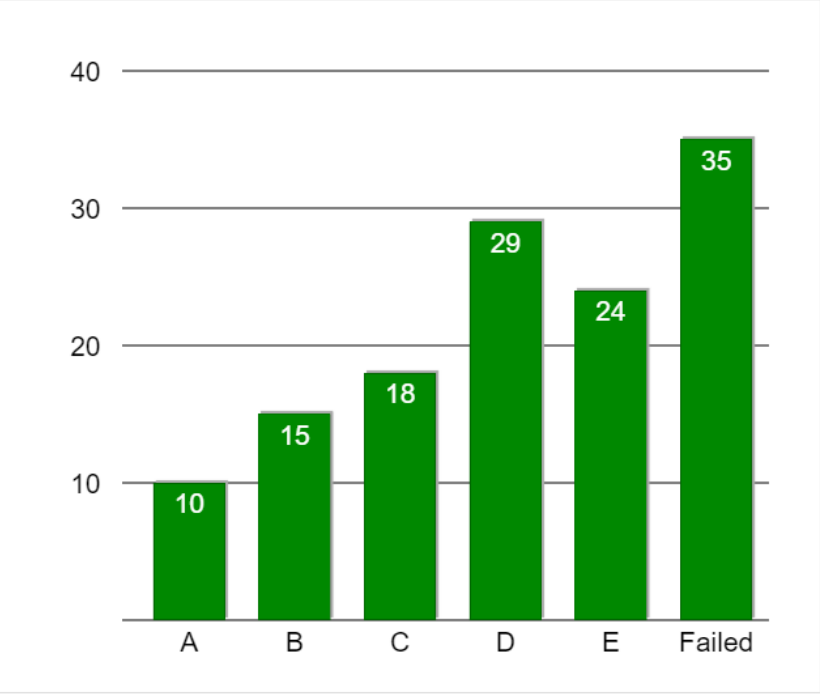


PROG2053 - Web Technologies

Statistics for PROG2053, 2022-12, Mandatory assignments

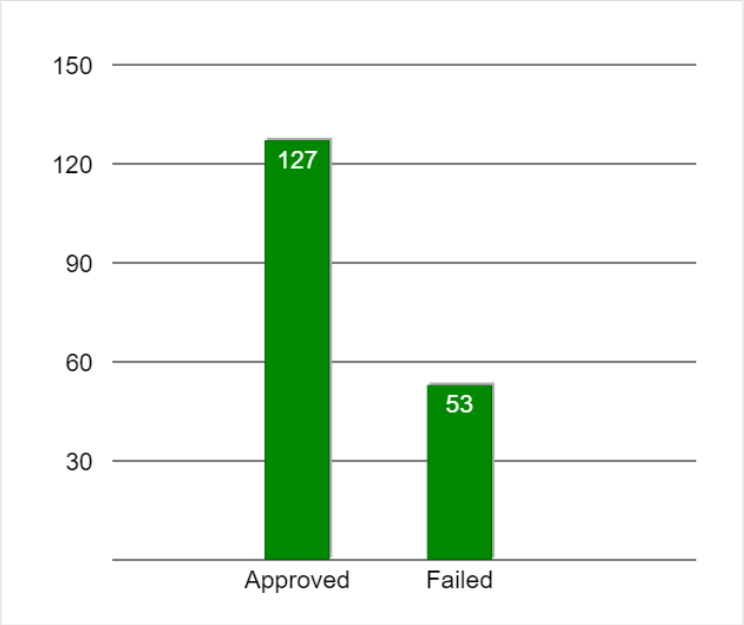


Statistics for PROG2053, 2022 AUTUMN, School exam

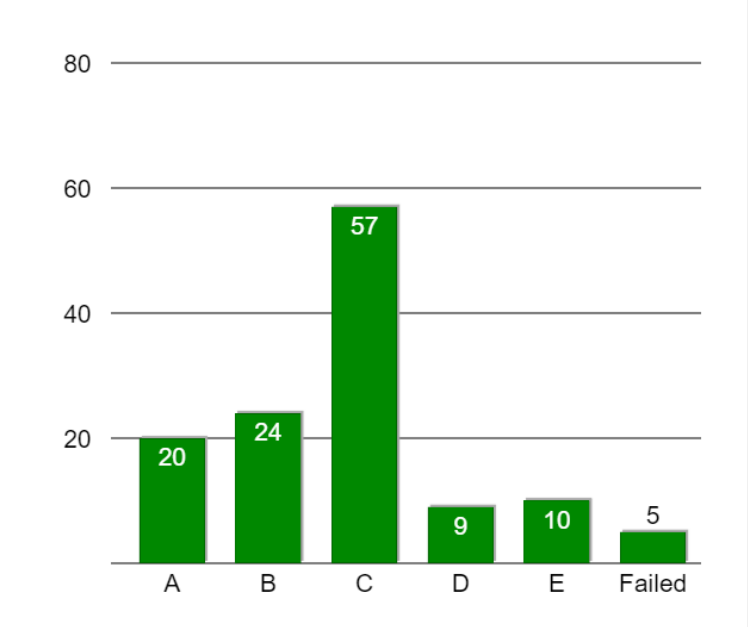


PROG1003 - Object-oriented Programming

Statistics for PROG1003, 2022-06, Prosjektoppgave og obligatoriske arbeidskrav

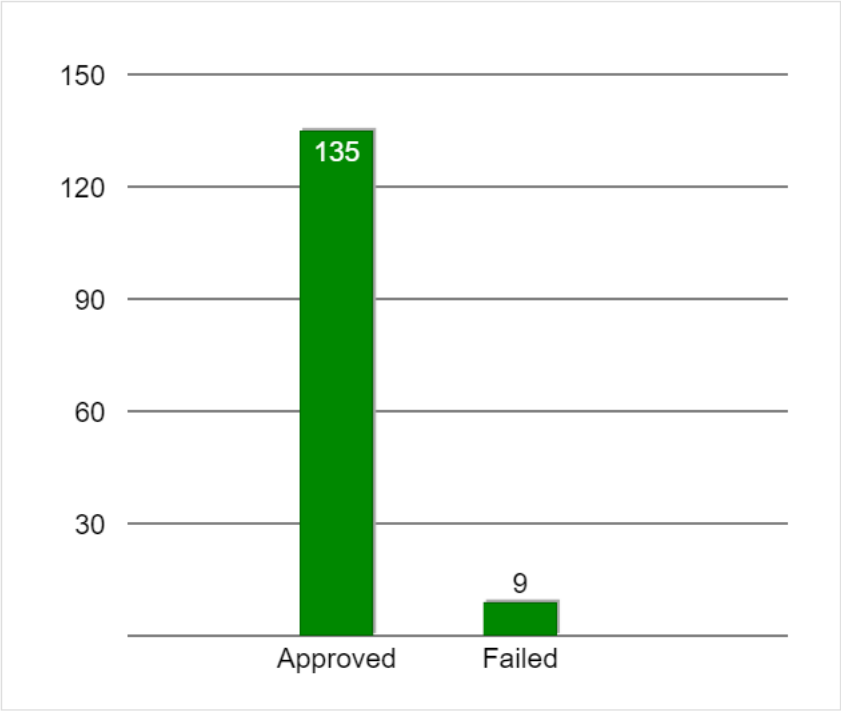


Statistics for PROG1003, 2022 SPRING, School exam

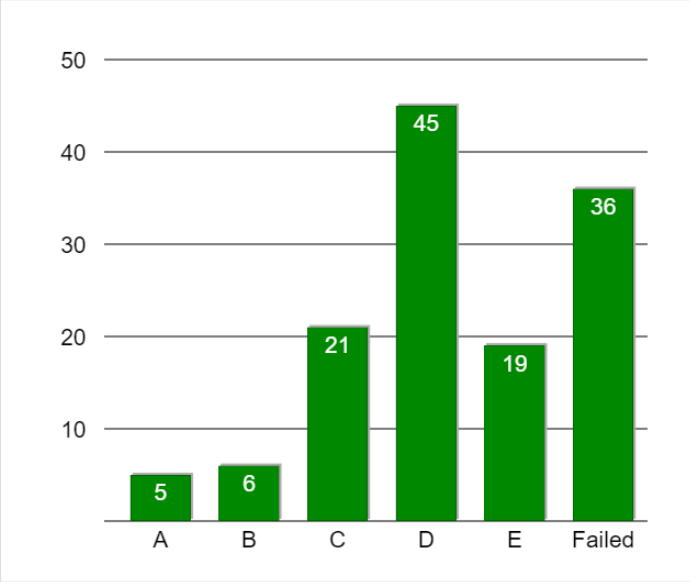


BMA1010 - Mathematics for Computer Science

Statistics for BMA1010, 2021-12, Obligatorisk arbeidskrav

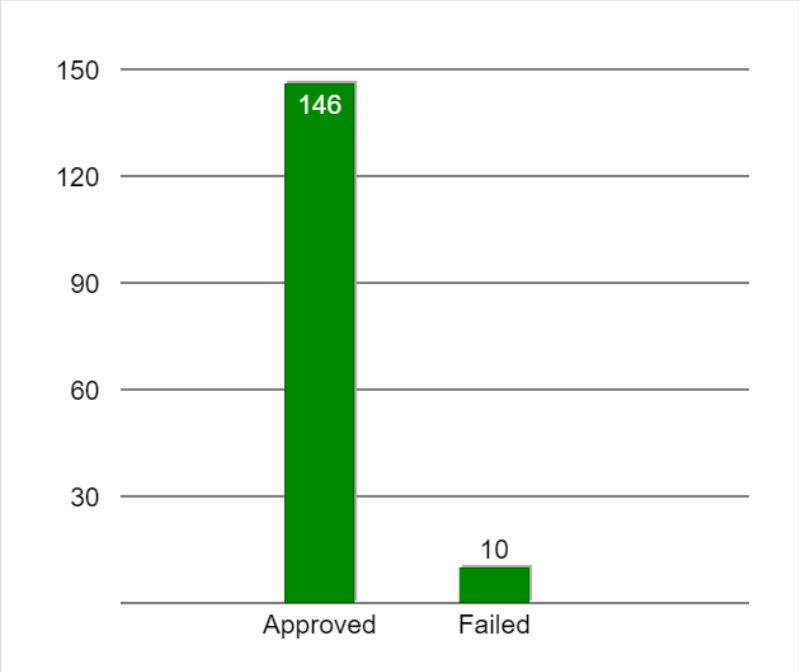


Statistics for BMA1010, 2021 AUTUMN, Written exam and portfolio assessment

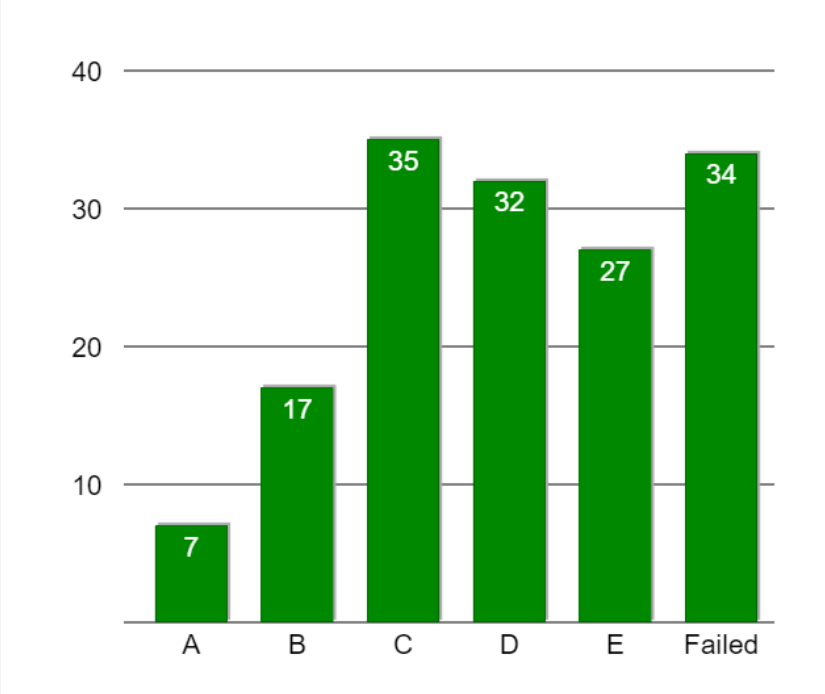


IDATG2202 - Operating Systems

Statistics for IDATG2202, 2022-12, Approved exercises

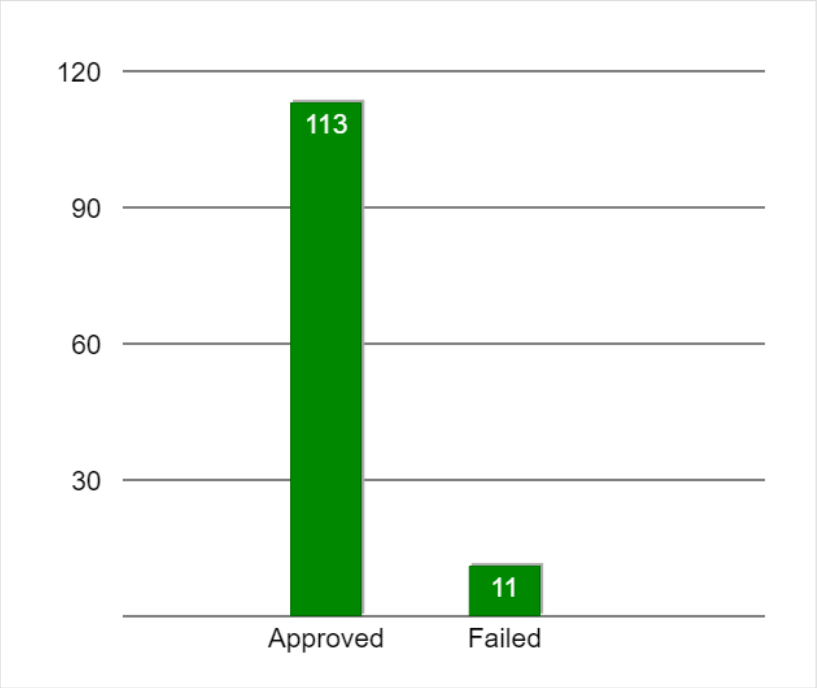


Statistics for IDATG2202, 2022 AUTUMN, School exam

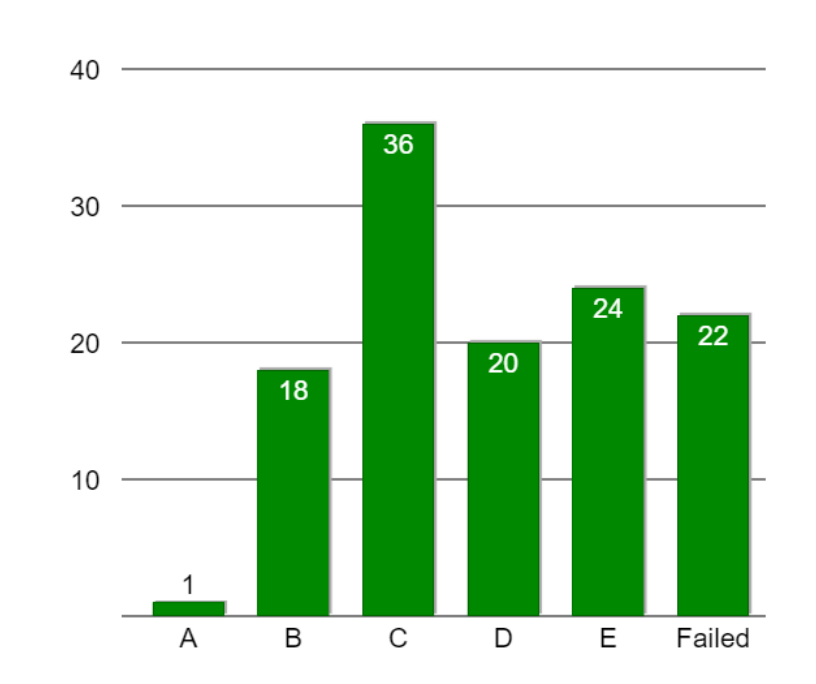


IDATG2102 - Algorithmic Methods

Statistics for IDATG2102, 2022-12, Mandatory works

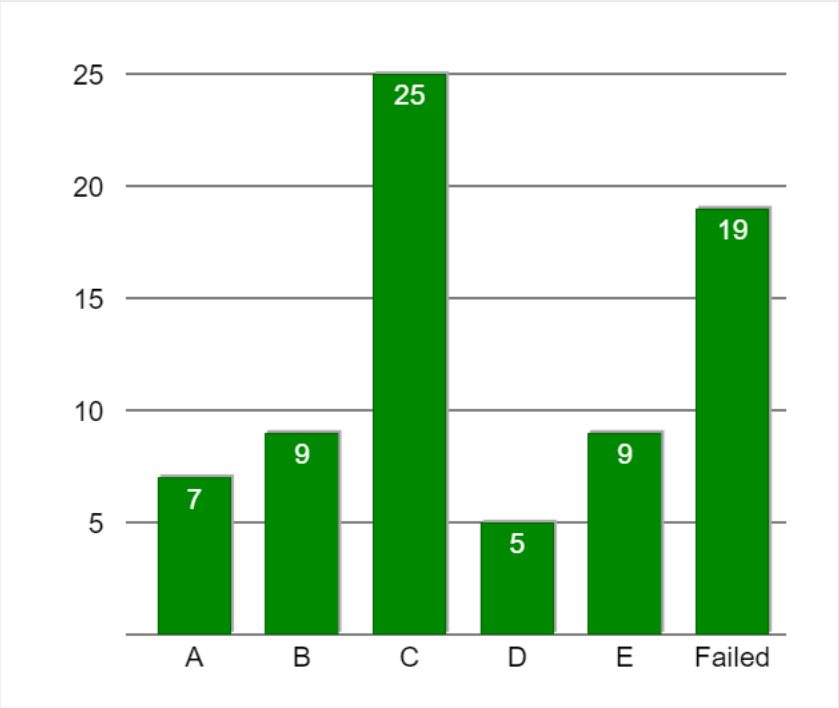


Statistics for IDATG2102, 2022 AUTUMN, School exam



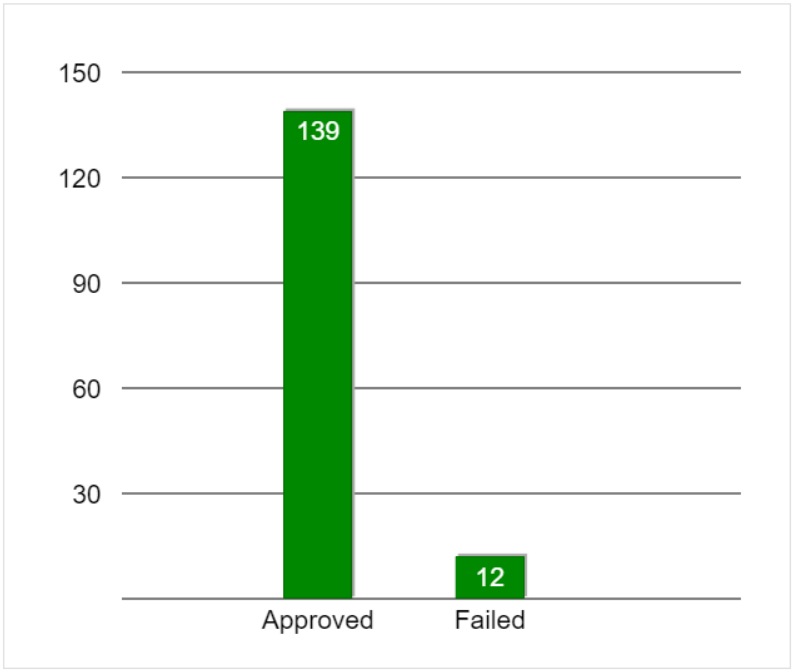
BMA1020 - Mathematics for Programming

Statistics for BMA1020, 2022 SPRING, Portfolio assessment

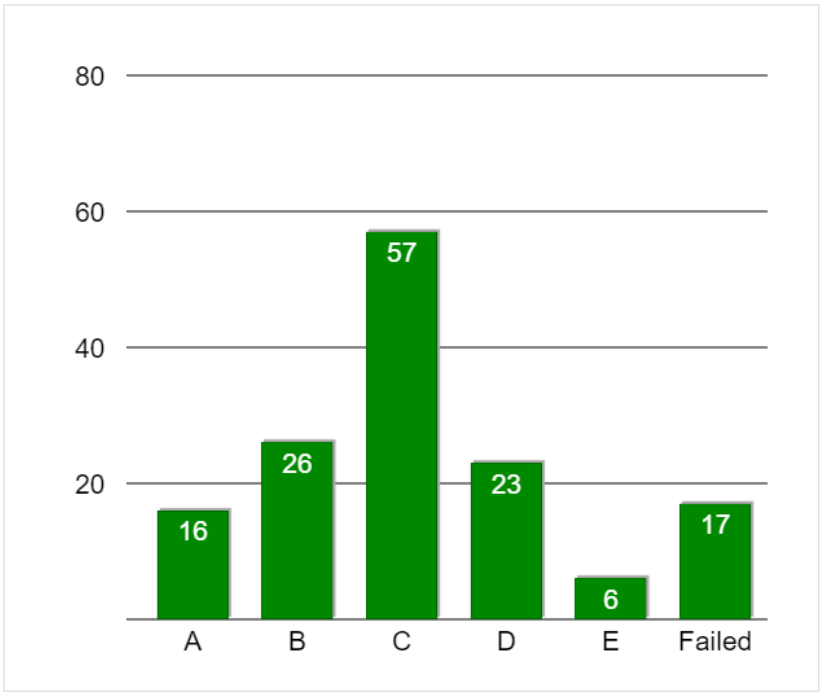


PROG1001 - Fundamental Programming

Statistics for PROG1001, 2021-12, Approved exercises

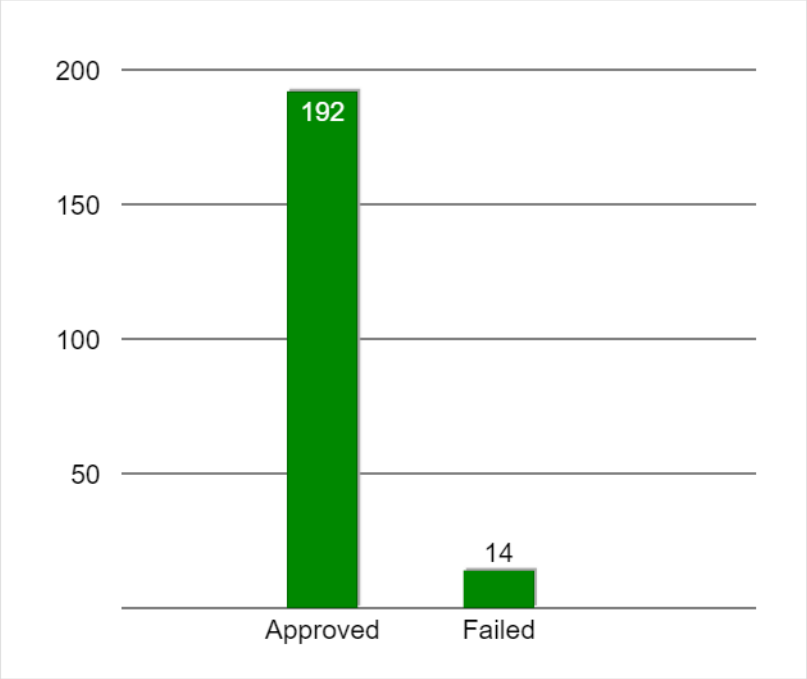


Statistics for PROG1001, 2021 AUTUMN, School exam

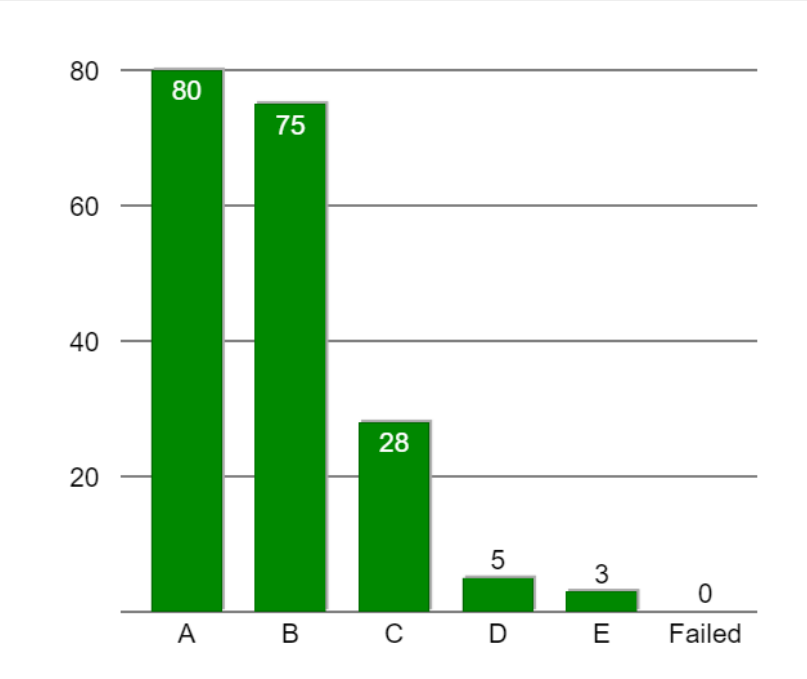


IDG1362 - Introduction to User-Centered Design

Statistics for IDG1362, 2021-12, Obligatoriske arbeidskrav

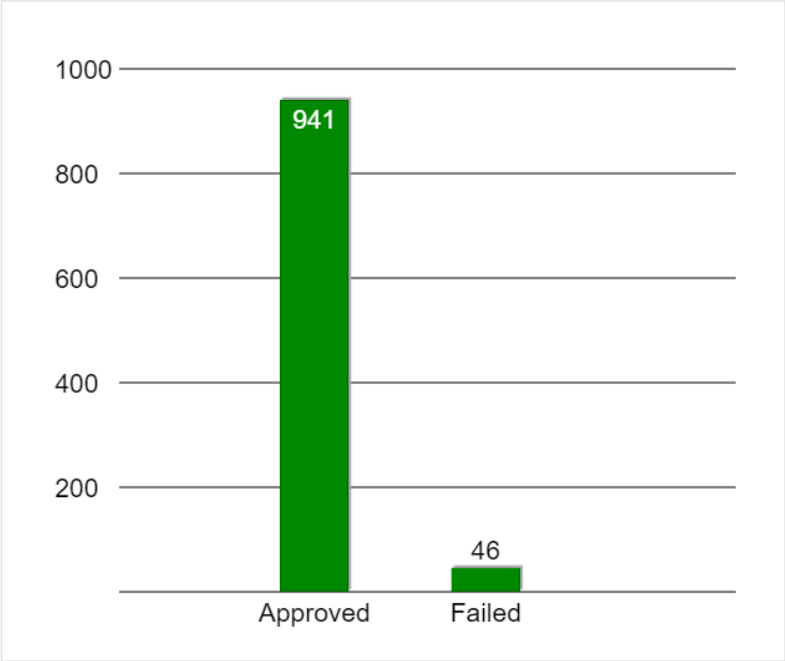


Statistics for IDG1362, 2021 AUTUMN, Home Exam

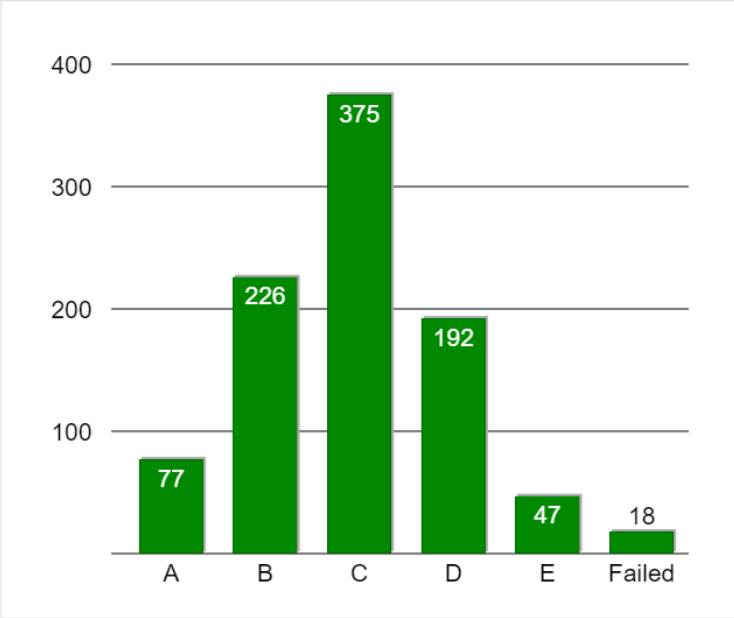


EXPH0300 - Examen philosophicum for Science and Technology

Statistics for EXPH0300, 2022-06, Obligatory activity

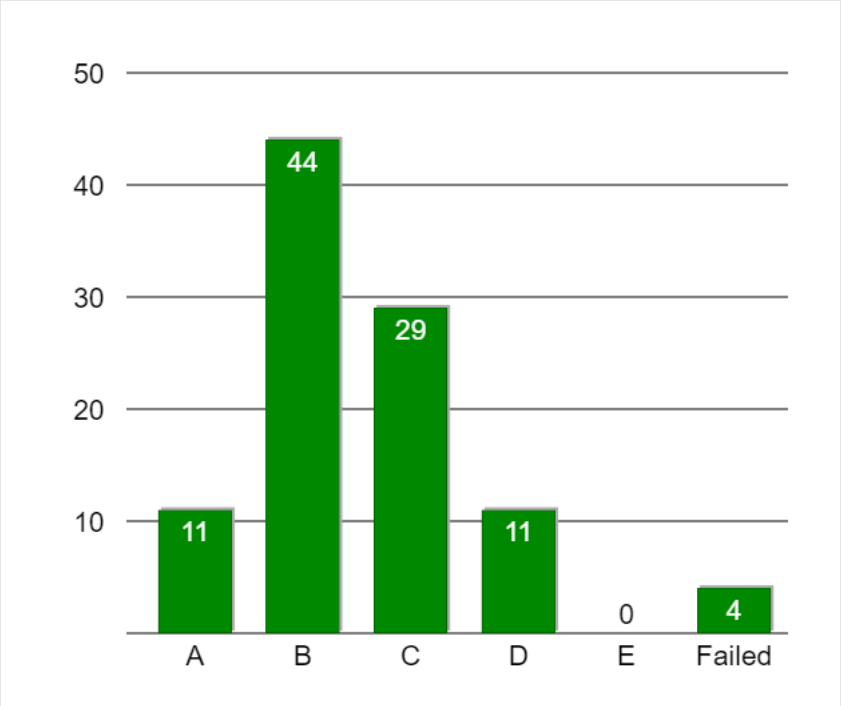


Statistics for EXPH0300, 2022 SPRING, Off Campus Examination / (Off Campus-) Multiple choice exam



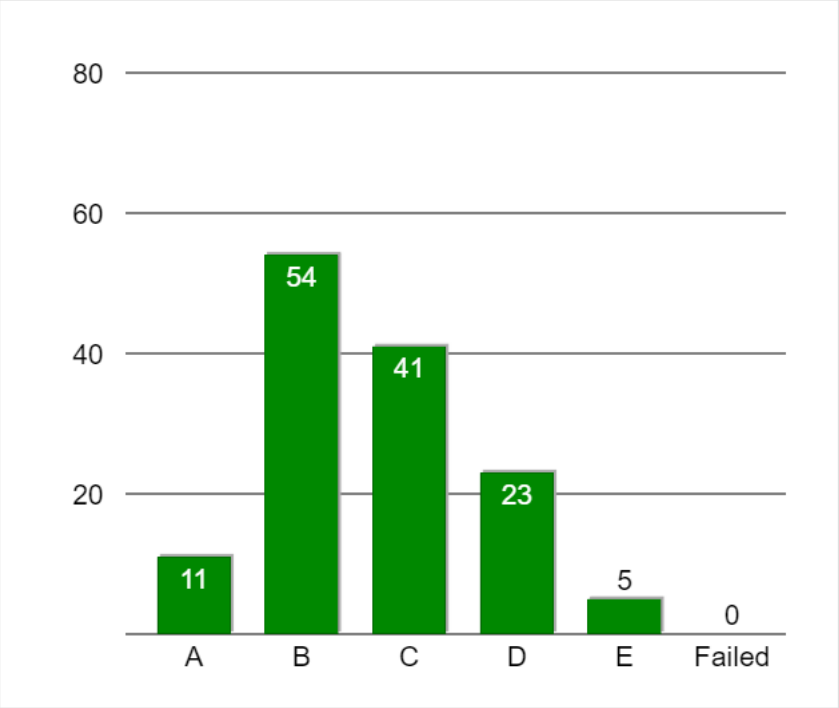
IIKG1001 - Cyber security and computer networks

Statistics for IIKG1001, 2021 AUTUMN, Aggregate score



PROG1004 - Software Development

Statistics for PROG1004, 2022 SPRING, Group based Project



Appendix 5

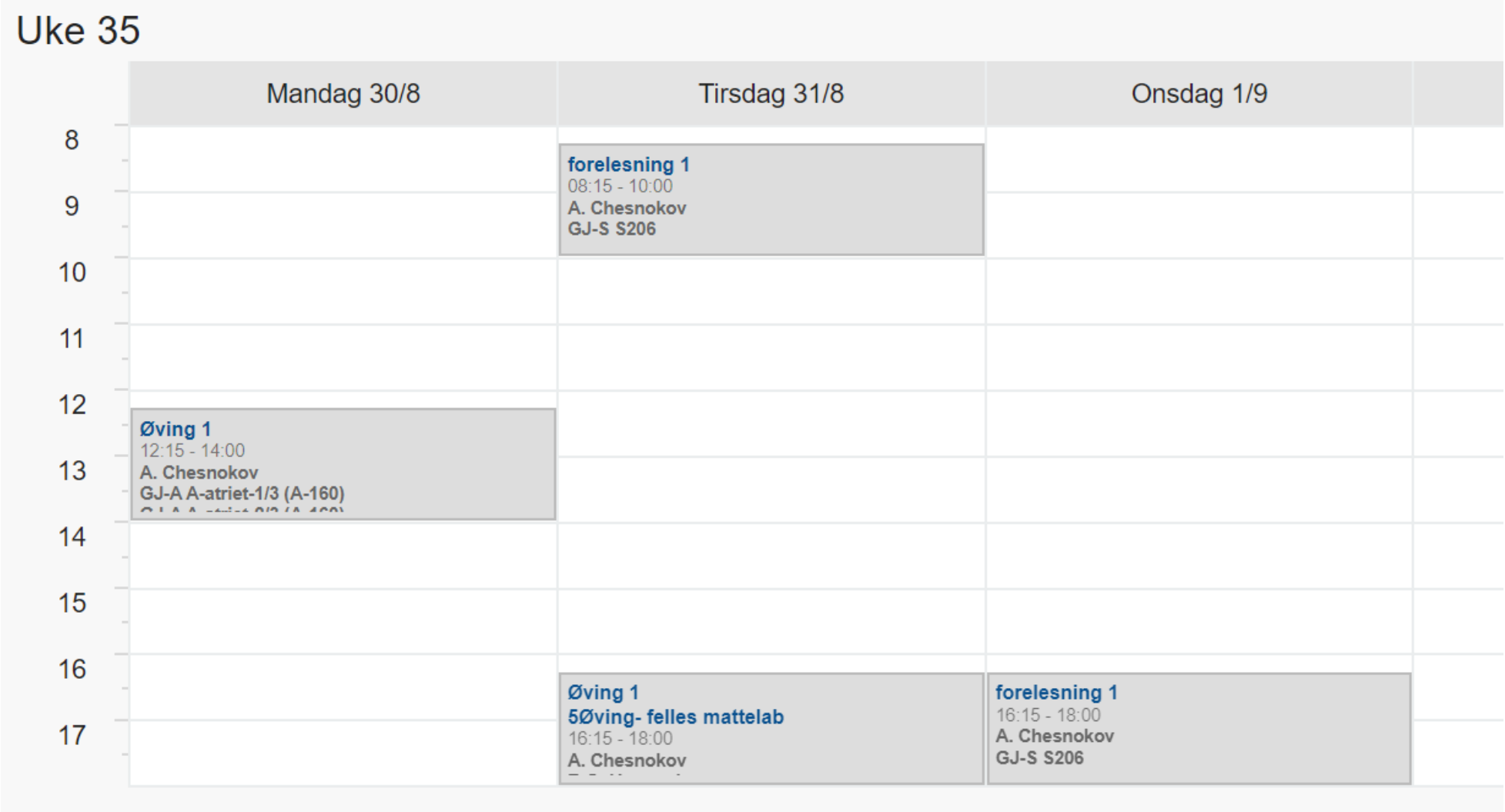
Table presenting the amount of lab and lecture time per week against the failing rate for a sample of courses

Course	lecture_time	lab_time	total	fail_rate
Mathematics for Computer Science	4	4	8	31 %
Software Development	4	2	6	0 %
Cyber security and computer networks	2	4	6	4 %
Graphics Programming	1	2	3	52 %

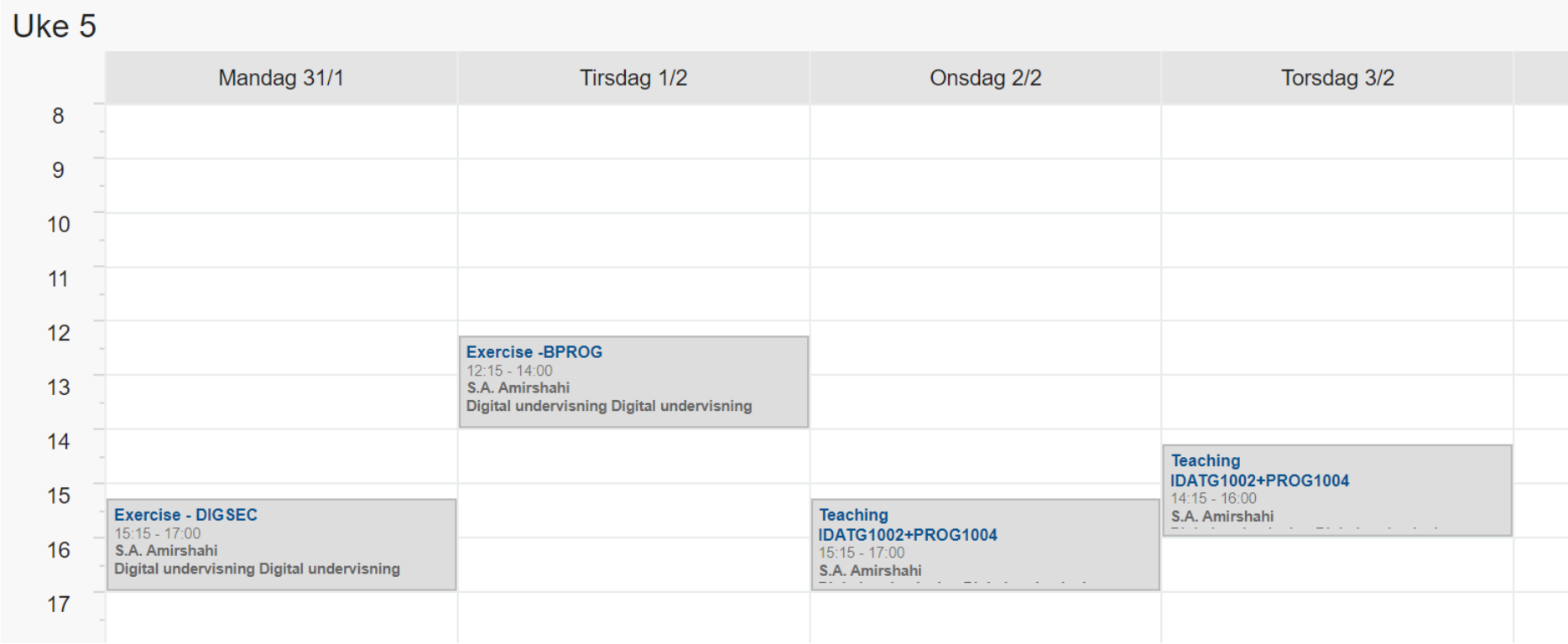
Appendix 6

Data from Timeplan used to produce the table in appendix 5

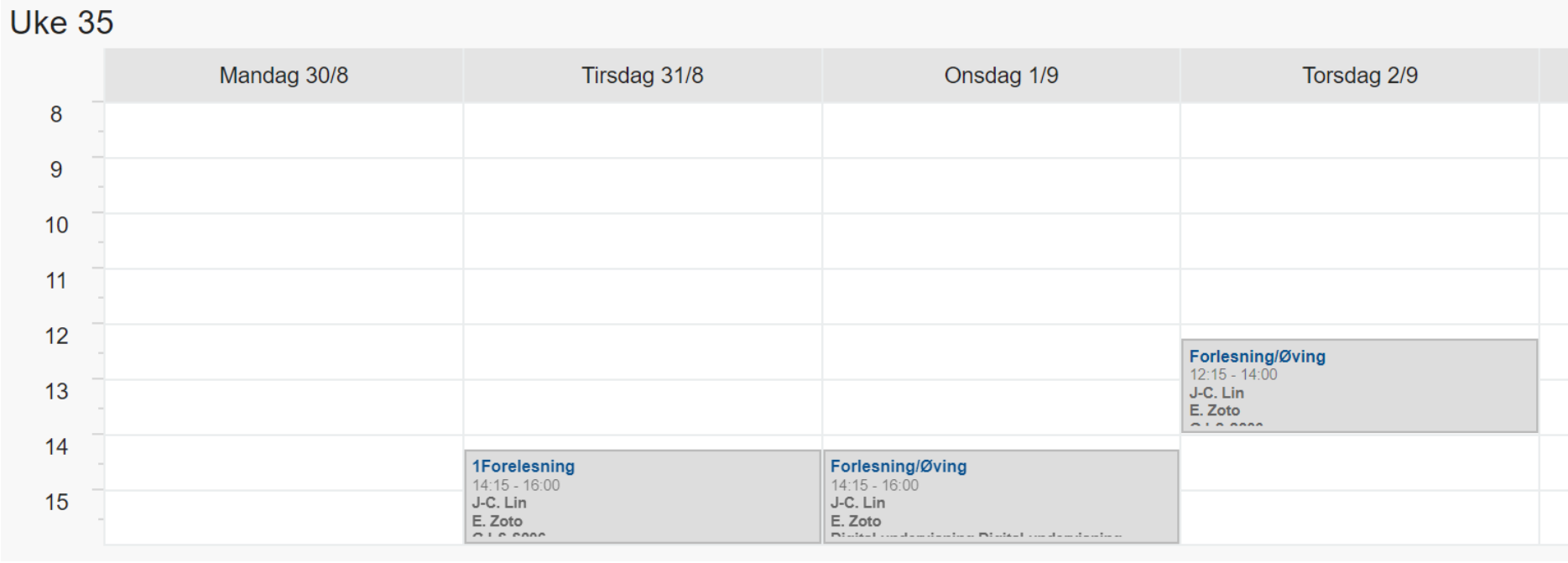
BMA1010 - Mathematics for Computer Science



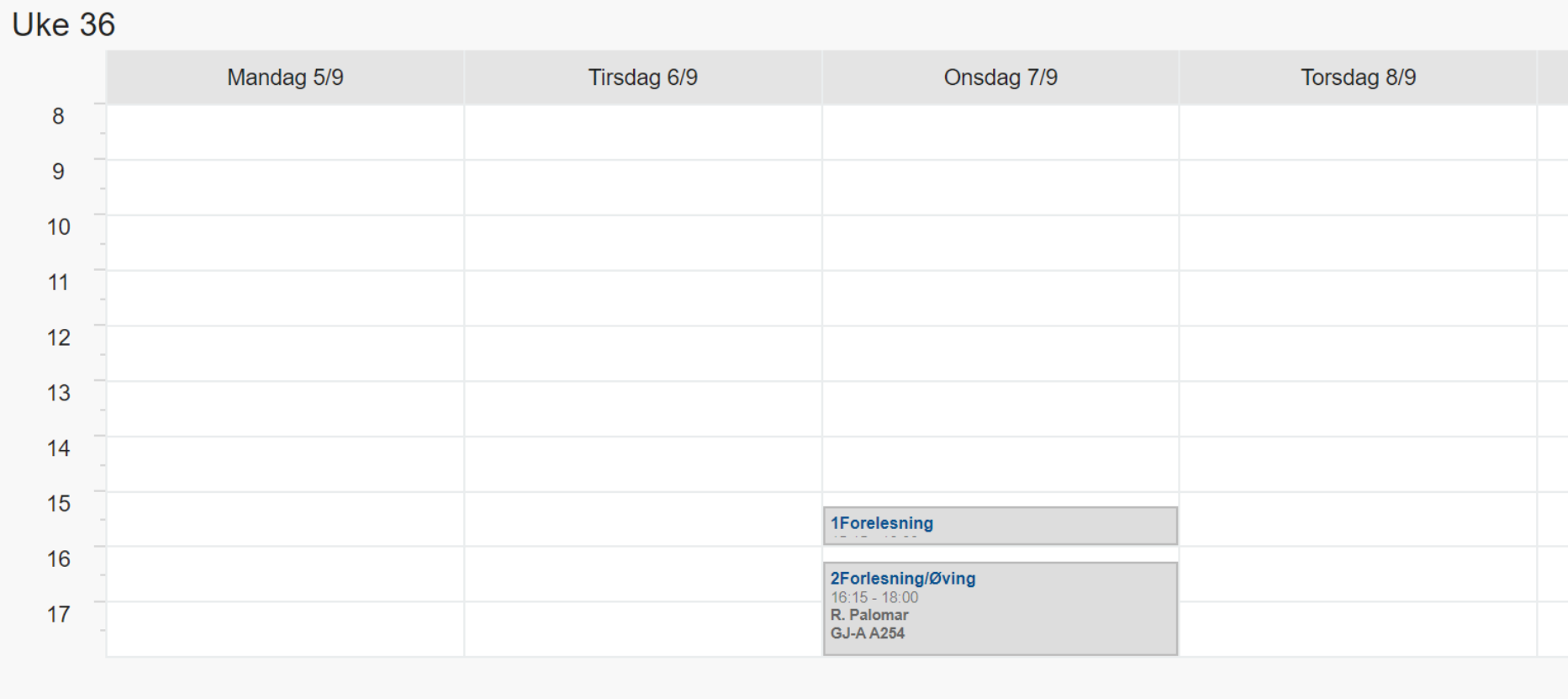
PROG1004 - Software Development



IIKG1001 - Cyber security and computer networks



PROG2002 - Graphics Programming



Appendix 7

Course description of this course at the time of this report

PROG2002 - Graphics Programming

Autumn 2022/ Spring 2023

Examination arrangement

Examination arrangement: Home exam
Grade: Letter grades

Evaluation	Weighting	Duration	Examination aids
Home exam	100/100	72 hours	

Course content

Core Topics:

- The OpenGL Graphics Pipeline (2D/3D).

- Geometric Transformations (2D/3D).

- Colors, Textures, Blending and Sprites.

- Shaders Programming with GLSL.

- Illumination Techniques.

- C++ Programming with CMake and Git.

Optional Topics:

- Advanced surface descriptions.

- Shadowing.

- Animations.

Learning outcome

On completion of this course the students will have the following.

Knowledge:

- Listing and describing the components of the OpenGL graphics pipeline.

- Understanding the mathematical foundations of geometric transformations in computer graphics.

- Handling textures to display image-based content.

- Applying Illumination in 2D/3D scenes.

- OpenGL API programming.

- GLSL Shaders programming.

Skills:

- Creating procedural animations.

- Utilizing 3D models (loading, onscreen display).

- Creating and manipulating lighting in a 3D scene.

- Using OpenGL for rendering 3D environments.

General Competences:

More on the course

No

Facts

Version: 1
Credits: 7.5 SP
Study level: Intermediate course, level II

Coursework

Term no.: 1
Teaching semester: AUTUMN 2022

Language of instruction: English

Location: Gjøvik

Subject area(s)

Computer Science

Contact information

Course coordinator:
Rafael Palomar Avalos

Department with academic responsibility
Department of Computer Science

- Finding solutions to a defined problem orally, and answer question about the solution.

- Using academic material from various online sources in an independent way.

- Improved software development ability.

- Reinforce version control, programming and code analysis.

Learning methods and activities

- Lectures with theoretical content.

- Labs and Assignments where the students will work in a problem-based learning setting using C++, OpenGL, CMake and Git, as well as some other relevant C++ libraries.

Compulsory assignments

Mandatory programming assignments

Further on evaluation

Re-sit examination is possible in agreement with the course responsible.

Assessment:

Home exam (100%)

Mandatory programming assignments that needs to be approved to be able to take the home exam.

Specific conditions

Compulsory activities from previous semester may be approved by the department.

Admission to a programme of study is required:
Computer Science (BIDATA)
Programming (BPROG)

Recommended previous knowledge

- BMA1010 Mathematics for Programming (or equivalent).

Required previous knowledge

- PROG1001 Fundamental Programming (or equivalent).

Course materials

Computing Resources:

- This course requires a computer with a GPU and an operating system capable of running OpenGL 4.3 or superior. **It is the responsibility of the student to set up the operating system and drivers to work properly for the course.**

- The material of the course has been prepared and tested in GNU/Linux systems. MS Windows users should be able to run this material as well. **Mac OS does not support OpenGL > 4.1** and we discourage its use for this course. Students who decide to take the course with Mac OS can do it using OpenGL 4.1, however, **the student will take full responsibility of adapting the material of the course to Mac OS, as well as making sure the deliverables work properly in Windows/Linux with OpenGL 4.3 or superior** ("but it works on my machine" won't be accepted).

Online Resources:

- Anton's OpenGL 4 Tutorials (http://antongerdelan.net/opengl/#ebook).

- Learn OpenGL Tutorials (https://learnopengl.com/).

Recommended Texts:

- OpenGL Programming Guide: The Official Guide to Learning OpenGL, 9th Edition.

- OpenGL SuperBible: Comprehensive Tutorial and Reference, 7th Edition.

- Mathematics for 3D game programming and computer graphics, 3rd Edition.