Rony G. Flatscher and Günter Müller

"Business Programming" – Critical Factors from Zero to Portable GUI Programming in Four Hours

Conference or Workshop Item (Published)
(Refereed)

"BUSINESS PROGRAMMING" – CRITICAL FACTORS FROM ZERO TO PORTABLE GUI PROGRAMMING IN FOUR HOURS

Rony G. Flatscher[1] and Günter Müller[2]

Abstract:
Teaching programming from scratch in a single semester such that in the end BA students become able to create GUI programs that run on Windows, MacOS, and Linux on their own appears to be impossible at first. However, over the course of twenty years such an endeavor has become successful at the WU, with 25,000 students one of the largest business administration Universities in Europe. The teaching load for the students is 200 hours (8 ECTS, European Credit Transfer System) of which 60 hours (4 hours per week, 15 weeks a semester) have to be attended in class. One critical success factor thereby is the programming language, another the goals that get defined for each installment of the syllabus, and finally, the pedagogical principles that have to be applied. This submission will give a bird eye's view of these critical success factors such that they can be reasoned and discussed.

Keywords: Software Engineering Education, End-user Software Engineering, Teaching goals, Teaching load, Pedagogical principles, Critical success factors, ooRexx, dynamic programming language, object-oriented, message based, open-source, multi-platform, Windows COM/OLE, MS Internet Explorer programming, MS Excel programming, Java bridge, camouflaging Java as ooRexx, GUI programming, socket programming, OpenOffice programming, LibreOffice programming, XML parsing SAX, XML parsing DOM, JavaFX.

INTRODUCTION

Teaching fundamentals of programming at a Business and Economics university is aimed at empowering the students to become able to create programs that interact with end-user business tools like office packages. This way they acquire unique skills that enable them to perform in professional life.
In 2021 the main operating system to address is MS Windows due to its market share in business organizations. However, for various reasons it would be beneficial, if skills would be taught that not only could be applied to Windows, but also to MacOS and Linux, thereby establishing the needed knowledge to create operating system independent programs.
This article will present work that has been ongoing for more than 20 years to come up with two lectures that are successfully used to teach the fundamental programming skills to business administration students without any prior programming knowledge in a single semester.
The article first introduces the formal structure of these two courses, the number of ECTS ("European Credit Transfer System") and how they are deployed in a semester. The format will be discussed and some implications for the students and the lecturer. Subsequently the perceived success factors, programming language and teaching principles, get briefly presented

FORMAL COURSE STRUCTURE

Each of the two courses, entitled "Business Programming 1" and "Business Programming 2", will have a teaching load of four "European Credit Transfer System (ECTS)" points. In Austria one ECTS translates to 25 net hours for the student. So for each of the two courses there is a teaching load of 100 hours available of which approximately 25 to 30 hours are "contact hours" in class. A four ECTS course therefore translates to a weekly two hours course and most of the work has to be conducted in form of homework assignments. In order to ease enrolling both courses for the students within one semester, each course is blocked for half a semester which means that the weekly contact hours are four hours instead of two hours. The second course will immediately start after the first one ends, using the same weekday and time as the first course, which makes sure that all students are able to continue with the second course as it does not interfere or overlap with other courses the students might be enrolled to.
The students get homework assignments and have to complete them right before the next installment of the course, concluding it with a little programming project assignment to be presented in the final installment.
It is very difficult for students to keep up their concentration during the four hours of weekly lectures, as they passively sit in the class room. The first break occurs after one and a half hour and lasts for 30 minutes ("time for two coffees"), followed by up to two additional short breaks of five minutes in the last two hours of an installment. This makes sure that the students can recuperate and that their circular flow reactivates them[3].

[1] University of Vienna, Austria
[2] University of Freiburg, Germany
[3] By contrast the lecturer usually moves constantly such that his concentration level and circular flow is kept at a high level at all times.

The weekly four contact hours of lecturing is meant as teaching as many new concepts as possible, explaining and demonstrating each of them in form of short nutshell examples and their output in detail, such that the students are able to fully understand them and relate to them to their homework assignments.
Nevertheless, the wealth of information per installment is extremely high, such that it is to be expected that the students will forget some of the concepts, so it is very important that the slides are as self-explanatory as possible and that the students are forced to go through the slides on their own thereby repeating the new taught concepts. Sometimes it may be the case that students feel overwhelmed and overburdened and lose their conviction that they would be able to master the concepts well enough to pass the course. To avoid drop outs a simple trick is applied: no student will be on her or his own, rather they have to conduct their homework assignments in groups of two students. This was drawn as an experiment from the military rule, that soldiers who have to keep watch in the dark must not be left alone, because otherwise single soldiers become frightened and quite often would leave their posts unauthorized, which does not occur if keeping the watch is done in groups of at least two soldiers. The same seems to apply to these courses, students practically never drop out if organized in groups of two.

CRITICAL SUCCESS FACTORS

This section stresses some factors that have been perceived to be critical to the success of these two lectures in the course of constantly improving them from semester to semester over the past 20 years.

Use an Easy to Learn Programming Language

The most important success factor (# 1) is the choice of the programming language being used for an introductory programming courses. Such a programming language needs to be easy to learn and have therefore a simple syntax, yet implement the most important programming concepts, such that the students can learn to apply them quickly by experiment. Compiled languages usually have an exhaustive set of syntax rules (including case sensitiveness) that the compiler needs to check at compilation time to make sure that as many possible programming errors of humans can be caught. By contrast, interpreted, dynamically typed programming languages that are caseless have

```
   -- a loop
do i=1 to 99               -- loop 99 times
   if i=1 then iterate     -- next loop, skip remaining body
   if i=4 then leave       -- leave loop prematurely
   say "hello #" i         -- says "hello # 2" and "hello # 3"
end

   -- demo arithmetics
say "i="i "1/i="1/i        -- says "i=4 1/i=0.25"

   -- demo sending a message to a string
say "hi, there!"~reverse   -- says "!ereht ,ih"

   -- demo use of a class
say ".demo:" .demo         -- says ".demo: The DEMO class"
o=.demo~new                -- create an instance
say "o:" o                 -- says "o: a DEMO"
o~name="BEE 2021"          -- assign value to attribute
say "o~name:"    o~name    -- says "o~name: BEE 2021"
say "o~reverse:" o~reverse -- says "o~reverse: 1202 EEB"

   -- demo defining a class
::class demo               -- defines the class DEMO
::attribute name           -- defines an attribute NAME
::method reverse           -- defines a  method REVERSE
  return self~name~reverse -- returns the name reversed
```

Figure 5: An ooRexx program

usually a reduced set of syntax rules and as a result can be learned much quicker.

In the course of more than 30 years a couple of different languages were used, among them VB (Visual Basic) Script which has astonishingly complex syntax rules, causing a lot of time to be consumed while teaching. Using in one semester the IBM language REXX (Cowlishaw 1990) which was explicitly designed to be “human

centered" and got ported from the mainframes to the PC world turned out to improve the success rates for the Business administration students to learn and apply programming considerably faster compared to COBOL, Pascal and VB Script. In addition, IBM had created an object-oriented successor "Object REXX" which later got open sourced by the non-profit special interest group "Rexx Language Association" in 2005 under the name "ooRexx", "open object Rexx" (Flatscher 2013a).
The REXX design principles did not change for ooRexx, such that it remained a "human-centered" language, which also has been kept small in the number of language constructs, built-in functions and built-in classes. One interesting aspect is its "message based" architecture drawn from Smalltalk: conceptually one codes in ooRexx by sending messages to objects which in turn resolves a method by the same name and invokes it, supplying any message arguments, and returns the result of the method to the sender. The notion thereby is that objects are like living things with which one communicates, which can be easily understood by the students. ooRexx employs an explicit message operator (the tilde) which is placed right next to the receiver object, followed by the message name and optionally a parenthesized argument list.
Figure 1 depicts an example ooRexx program that demonstrates that its keyword instructions and message-based architecture can be easily understood, as they almost read like English pseudo code.
The Windows version of ooRexx supports the building block technology COM ("Component Object Model") and OLE ("Object Linking and Embedding") built on it. The Windows COM/OLE classes can be addressed via the ooRexx proxy class OLEObject. Instances of OLEObject are proxy Windows objects to which programmers just send messages by the name of the Windows methods that need to be invoked on the Windows side, very much like the students already got used to with learning ooRexx.[4]
Flatscher (2001, 2003, 2006a, 2009, 2010, 2011, 2012, 2014, 2015, 2018, 2019) developed a Java bridge for ooRexx ("BSF4ooRexx") such that the message-based property of ooRexx can be applied in interacting with Java which effectively turns Java into a message-based, dynamically typed language from the perspective of ooRexx programmers! This makes it possible to teach students without any prior Java knowledge to exploit Java classes and interact with Java objects because they can treat Java as if it was the ooRexx message-based and dynamically typed language! They already know of classes and methods, and relating Java fields with ooRexx attributes is quite simple as well for them.

Pareto Principle

Teaching students without prior programming skills object-oriented programming and empowering them in the end to create applications in a platform independent manner in one semester using two courses seems to be impossible. However, if one analyzes what concepts are of fundamental importance and must be taught it follows that there are many interesting concepts that may not be as important and could be left out, thereby saving a lot of teaching time. The Pareto principle in this context (critical success factor # 2) would then allow to teach students 80% of the most important concepts in 20% of the time compared to teaching 100% (the remaining 20% would impose an effort of 80% which may be justified in the case where a certain programming language needs to be taught in all details).

Humboldt's Ideal

Teaching is sometimes an art where picking the concepts and creating pedagogical paths that ease digesting and understanding them. One can never be sure whether a lecture unit is already perfect or could still get improved. In addition new scientific developments may cause the adding of new concepts to existing lectures, which pose the same basic question: are the pedagogical concepts and materials good enough to be taught to the students in a manner, that allows them to quickly and efficiently learn them? Applying constantly the Humboldt's ideal (critical success factor # 3), observing how the students manage new concepts, observing what problems they get and why, and if necessary, create new paths to ease the understanding of those new concepts[5] and retest them in the next installment of a course, will over time allow to constantly improve the course contents.

No Student Is Left Alone

Another critical success factor is creating groups of two students (critical success factor # 4) who together have to carry out their homework assignments and the final project together. In the case that there are students with prior programming skills, then mix them such that each gets a student as his or her team member who cannot program

[4] The implementation of the OLE support takes advantage of the *unknown* mechanism which allows a Rexx class to intercept unknown messages (messages for which no method could be found in the ooRexx class hierarchy) and forward them appropriately marshalled to Windows instead.
[5] The Java bridge BSF4ooRexx has been developed applying the Humboldt's ideal over a course of almost 20 years, which in the end has created the surprising effect that students became able to interact with Java without any need of learning the Java programming language itself.

at all. In this case the student with the prior programming knowledge will take on the role of a "buddy tutor" at least initially as long as known concepts get taught. One psychological effect of building groups of two is the desire of group members not to desert the companion and as a result stay enrolled in the class even if a student feels overcharged at times.
Also note that increasing the group beyond two students may increase the phenomenon of "submarine" students who do not participate in the group's work at all, but rather abuse the solidarity of the other students in the group who do not reveal such loafers, if the local culture scorns betrayal.

Searching the Internet

In today's world solving programming problems usually is most efficiently done by searching the Internet first (critical success factor # 5). It can be expected that many coding problems have been posted and solved on the Internet already such that looking for solutions there first most likely saves a lot of time and effort on the programmer's side. In addition, students may take advantage of tutorials in all media forms, from Internet articles to Youtube online videos, becoming able to learn about new concepts on their own.

Nutshell Examples with Output

Making it as easy as possible for students to learn a programming language, nutshell examples (critical success factor # 6) are of paramount importance. "Nutshells" are very short programs that stress a single concept and can be executed as is. Supplying the output of such nutshells allows the students to relate to the effects ("seeing is better than believing") without personally executing them.

Weekly Coding Assignments

The students have to create two short programs[6] that each employ one of the new concepts learned in class (critical success factor # 7). For beginners this can be already quite challenging, causing a lot of effort (timewise, but also intellectually) which at the same time serves as constant training. The assignments have to be carried out in groups together, such that one student becomes able to help the other in case of problems.

Concluding Project Assignment

During the courses the students become able to master the concepts of each installment from date to date, however, they usually do not realize how much knowledge and how many skills they have acquired during the course. To make them realize this, a concluding project assignment takes place (critical success factor # 8): the students have to come up with three project ideas in the penultimate installment, where they need to interact with three Windows programs (or Java class libraries in the second course). One of the three project ideas gets picked and the group has to code that project for the last installment where the chosen project gets presented and the solution gets demonstrated. The result of this last assignment is usually that the students become excited and motivated as they have experienced that their efforts have yielded problem solution skills that were unforeseen for them at the beginning of the course.

BRIEF DESCRIPTIONS OF BUSINESS PROGRAMMING 1 AND BUSINESS PROGRAMMING 2

This section describes very briefly the concepts that get taught in each of the two consecutive, blocked courses. This will allow the reader to assess the large scope that can be taught successfully if the critical success factors get applied to each lecture and each installment.

Business Programming 1 (BP1)

The purpose is to teach the students the fundamentals of object oriented programming and have them apply their acquired skills to programming the Windows shell and Windows applications using OLE. The syllabus for the spring semester 2021 is given in Flatscher 2021a, the slides going with this class in Flatscher 2021b.[7]

[6] Demanding short programs is important because in the case of students with prior programming knowledge the danger arises that they create large, complex programs that frustrate the newcomers who barely can understand them due to missing out the necessary out-of-class acquired knowledge of such "monster" programs. In addition even competitions who is able to create the most impressive program were observed among groups with prior programming skills over time, frustrating the newcomers even more.
[7] The forerunner syllabus got documented more than 20 years ago (Flatscher 1999).

1) Installment # 1, 4 Hours

The purpose is to relate the students to the programming language ooRexx, its history, its "human-centered" design which eases learning it. The content is taught slowly step by step introducing the programming concepts of statement, block comment, line comment, symbol, variables, block, comparisons (yielding Boolean values), branch, loop.

2) Installment # 2, 4 Hours

The purpose is to make the students understand the need and concept of routines. This incurs the concepts jump label, internal routines, scopes, functions with return values, built-in-functions, external functions (from function libraries) and associative arrays ("stem" variables).

3) Installment # 3, 4 Hours

The purpose is to introduce the students to conditions (exceptions) and condition handlers, routine directive, requires directive, fetching arguments by reference.

4) Installment # 4, 4 Hours

The purpose is to introduce the students to the fundamental object-oriented programming concepts, ADT (abstract data type), class (synonym: type), attribute, method, message, cascading messages, creating instances (synonym: values, objects), multithreading, garbage collector. Note, the multithreading concept is not taught in a manner that the students could apply it at this point in time, rather it is meant to make them aware that such a concept exists, what it means, learning thereby that ooRexx would implement inter-object and intra-object multithreading.

5) Installment # 5, 4 Hours

The purpose is to introduce the students to the concepts class hierarchy, inheritance, multiple inheritance, fundamental classes, collection classes and iterator class. This includes many nutshell examples that demonstrate the different introduced classes such that the students become able to create programs on their own for their homework assignments, thereby experimenting with some of these classes.

6) Installment # 6, 4 Hours

This installment introduces the students first to the history of the fundamental Windows programming infrastructures COM (component object model) and OLE (object linking and embedding), GUID, UUID, the Windows registry, registry hives, CLSID, ProgId, Monikers, such that they can develop a bird eye's view of the OLE programming model. This is followed by introducing the ooRexx OLEObject class which serves as the proxy class for Windows COM classes (Flatscher 2004). The acquired theoretical knowledge is immediately applied to the ooRexx OLE sample nutshell examples coming with the Windows version of ooRexx, which they need to study one by one and then create two small OLEObject programs on their own in their homework assignment. Among them are samples for interacting with MS Access, MS Excel and MS Word from ooRexx via OLE.

7) Installment # 7, 4 Hours

The students learn the fundamental concepts of SGML markup languages (element, attribute, content model) demonstrated with HTML (hypertext markup language), CSS (cascading style sheets), MS DHTML ("dynamic HTML"), MS Internet Explorer and interacting with and controlling it via OLE, navigating to web sites, fetching the documents and parsing them.

To ease transcribing ooRexx OLE programs to VB (Visual Basic) Script and VB Script examples on the Internet to ooRexx the VB Script programming language gets introduced and compared to ooRexx. Nutshell examples show how the different concepts map to the other language, and in detail explain VB Scripts arguments by name as that notation is being used by macro recorders quite extensively.

Business Programming 2 (BP2)

The purpose is to teach the students operating system independent programming by exploiting Java which gets camouflaged as ooRexx with the help of the BSF4ooRexx Java bridge. The students will learn socket programming, GUI programming, OpenOffice/LibreOffice programming, XML parsing and JavaFX GUI programming for the most complex GUI needs in a portable manner. The syllabus for the spring semester 2021 is given in Flatscher 2021c, the slides going with this class in Flatscher 2021d.

1) Installment # 1, 4 Hours

The purpose is to relate the students to the programming language Java, its history, its concepts and equivalence to ooRexx. The concepts taught are types, primitive types, classes, methods, fields, visibility, Javadocs, the BSF4ooRexx Java bridge, need of boxing primitive types, Java arrays, boxing ooRexx objects as Java objects.

2) Installment # 2, 4 Hours

This installment consists of two different contents: creating simple GUIs with the classes in the Java package java.awt, teaching the concepts of (event driven) callbacks and implementing Java interface classes with ooRexx classes, nutshells using ooRexx code exclusively (Flatscher 2013b). Acquiring this knowledge is fascinating and "fun" for the students as often can be seen by their homework assignment code.

The second part introduces socket programming starting out with the concepts developed for telephony originally (switchboard, plugs, patch cords) and then turning to the concepts Internet address, socket, server socket. This part concludes with the introduction of SSL (secure socket layer) and a nutshell demonstrating it as well.

3) Installment # 3, 4 Hours

This installment introduces the history of OpenOffice and LibreOffice, its UNO (universal network object) architecture, its type system, its Java based scripting framework that allows the BSF4ooRexx Java bridge to be put to use and even make ooRexx available as a macro language for these open source, platform independent office packages (Flatscher 2005a, 2005b, 2006b). The students learn how to program the word processor module ("swriter"), the spreadsheet module ("scalc") and the presentation module ("simpress") such that their ooRexx programs run unchanged on Windows, MacOS and Linux from within or outside of these office packages.

4) Installment # 4, 4 Hours

In a business world that marks up many vital text data with XML it becomes important to be able to process XML files. The students learn the fundamentals of XML markup languages and the two most important techniques for processing XML marked up data: SAX (simple API for XML) and DOM (document object model). The nutshell examples for SAX (Flatscher 2013c) and DOM (Flatscher 2013d) both achieve the same results, such that the students can easily assess the programming differences among the two techniques.

5) Installment # 5, 4 Hours

The purpose is to teach the students the Java scripting framework architecture (Java package "javax.script") which Java programs may use to host and execute scripts/macros. They get introduced to the BSF4ooRexx RexxScriptEngine implementation, which allows ooRexx to be deployed via the Java scripting framework by merely denoting "rexx" as the desired scripting language.

6) Installment # 6, 4 Hours

The students learn the fundamentals of JavaFX, JavaFX properties, FXML, the JavaFX SceneEditor, and the architecture that one needs to adhere to (Flatscher 2017). As JavaFX supports the Java scripting framework it becomes possible to implement controllers for FXML files in ooRexx, and in the end even create stand-alone ooRexx programs that take advantage of all of what JavaFX offers without a need to write any Java code.

7) Installment # 7, 4 Hours

This installment will teach the setting up of Java environments if external Java class libraries (and starting with Java 9 external Java modules) need to be made available to Java. Optionally, if interest is warranted by students the concepts of .Net get taught as BSF4ooRexx, exploiting a Java bridge to .Net, makes it fully available to the ooRexx programmer very much like OLEObject the OLE interface.

ROUNDUP AND OUTLOOK

This article aims to communicate the critical success factors that play a role for teaching platform independent, object-oriented programming to Business administration students at WU in one semester. The changes with regard to previous teaching methods have been shown and a descriptive evaluation performed during class . This has been done by assessing the teaching load, the knowledge and the skills they acquired. Graduates of these courses may come back for Bachelor theses that deal with creating information systems that help their parental companies or companies they actually work for. In any case the result is that Business administration students appreciate the skills and personal experiences that will empower them to guide software engineering project groups

REFERENCES

1. Cowlishaw, MF 1990, *The REXX Language*, 2nd edn, Prentice Hall, Englewood Cliffs, New Jersey.
2. Flatscher, RG 1999, 'A syllabus for introducing MBA students to procedural and object-oriented programming (Obejct REXX)' *in proceedings of Americas Conference on Information Systems (AIS AMCIS),* Milwaukee, Wisconsin.
3. Flatscher, RG 2001, 'Java bean scripting with Rexx' in C Davis & R Jansen (eds) *proceedings of the 2001 International Rexx Symposium,* Research Triangle Park, North Carolina.
4. Flatscher, RG 2003, 'The Augsburg version of BSF4Rexx' in C Davis & R Jansen (eds) *proceedings of the 2003 International Rexx Symposium,* Raleigh, North Carolina.
5. Flatscher, RG 2004, 'Object Rexx and Windows automation interfaces' in C Davis & R Jansen (eds) *proceedings of the 2004 International Rexx Symposium,* Böblingen, Germany.
6. Flatscher, RG 2005a, 'Automating OpenOffice.org with ooRexx: Architecture, Gluing to Rexx using BSF4Rexx' in C Davis & R Jansen (eds) *proceedings of the 2005 International Rexx Symposium,* Los Angeles, U.S.A.
7. Flatscher, RG 2005b, 'Automating OpenOffice.org with ooRexx: ooRexx nutshell examples for writer and calc' in C Davis & R Jansen (eds) *proceedings of the 2005 International Rexx Symposium,* Los Angeles, U.S.A.
8. Flatscher, RG 2006a, 'The Vienna version of BSF4Rexx' in C Davis & R Jansen (eds) *proceedings of the 2006 International Rexx Symposium,* Austin, Texas.
9. Flatscher, RG 2006b, 'An (Open) Object Rexx module for universal network objects' in C Davis & R Jansen (eds) *proceedings of the 2006 International Rexx Symposium,* Austin, Texas.
10. Flatscher, RG 2009, 'The 2009 edition of BSF4Rexx' in C Davis & R Jansen (eds) *proceedings of the 2009 International Rexx Symposium,* Chilworth, England.
11. Flatscher, RG 2010, 'The 2010 edition of BSF4ooRexx' in C Davis & R Jansen (eds) *proceedings of the 2010 International Rexx Symposium,* Amsterdam and Almere, The Netherlands.
12. Flatscher, RG 2011, 'The 2011 edition of BSF4ooRexx (for Windows, Linux and MacOSX)' in C Davis & R Jansen (eds) *proceedings of the 2011 International Rexx Symposium,* Aruba, Dutch West Indies.
13. Flatscher, RG 2012, 'Creating ooRexx interpreter instances from Java and NetRexx' in C Davis & R Jansen (eds) *proceedings of the 2012 International Rexx Symposium,* Raleigh, North Carolina.
14. Flatscher, RG 2013a, *Introduction to Rexx and ooRexx,* facultas, Vienna.
15. Flatscher, RG 2013b, 'Creating cross-platform GUIs with BSF4ooRexx' in C Davis & R Jansen (eds) *proceedings of the 2013 International Rexx Symposium,* Raleigh, North Carolina.
16. Flatscher, RG 2013c, 'Processing XML documents with SAX using BSF4ooRexx' in C Davis & R Jansen (eds) *proceedings of the 2013 International Rexx Symposium,* Raleigh, North Carolina.
17. Flatscher, RG 2013d, 'Processing XML documents with DOM using BSF4ooRexx' in C Davis & R Jansen (eds) *proceedings of the 2013 International Rexx Symposium,* Raleigh, North Carolina.
18. Flatscher, RG 2014, 'The 2014 edition of BSF4ooRexx (for Windows, Linux and MacOSX)' in C Davis & R Jansen (eds) *proceedings of the 2014 International Rexx Symposium,* Memphis, Tennesse.
19. Flatscher, RG 2015, 'New features in BSF4ooRexx (camouflaging Java as ooRexx)' in C Davis & R Jansen (eds) *proceedings of the 2015 International Rexx Symposium,* Vienna, Austria.
20. Flatscher, RG 2017, 'JavaFX for ooRexx – Creating powerful portable GUIs for ooRexx' in C Davis & R Jansen (eds) *proceedings of the 2017 International Rexx Symposium,* Amsterdam, The Netherlands.
21. Flatscher, RG 2018, 'The new BSF4ooRexx 6.00' in C Davis & R Jansen (eds) *proceedings of the 2018 International Rexx Symposium,* Aruba, Dutch West Indies.
22. Flatscher, RG 2019, 'The 2019 edition of BSF4ooRexx (for Windows, Linux and MacOSX)' in C Davis & R Jansen (eds) *proceedings of the 2019 International Rexx Symposium,* Hursley, England.
23. Flatscher, RG 2021a, 'Business Programming 1 (Syllabus)', WU Vienna, 1 February, viewed 12 June 2021, <http://wi.wu.ac.at/rgf/wu/lehre/autowin/2021sBP1/BP1-autowin-2021s-uebersicht.pdf>.
24. Flatscher, RG 2021b, 'Business Programming 1 (Slides)', WU Vienna, 1 February, viewed 12 June 2021, <http://wi.wu.ac.at/rgf/wu/lehre/autowin/material/foils/>.
25. Flatscher, RG 2021c, 'Business Programming 2 (Syllabus)', WU Vienna, 1 February, viewed 12 June 2021, <http://wi.wu.ac.at/rgf/wu/lehre/autojava/2021sBP2/BP2-autojava-2021s-uebersicht.pdf>.
26. Flatscher, RG 2021d, 'Business Programming 2 (Slides)', WU Vienna, 1 February, viewed 12 June 2021, <http://wi.wu.ac.at/rgf/wu/lehre/autojava/material/foils/>.