International Journal of TESOL Studies (2021)
Vol. 3 (2) 105-113 https://doi.org/10.46451/ijts.2021.06.08

Interview

Developing Practical, Useful Educational Technology Applications for Course Designers, Teachers and Language Learners: An Interview with Laurence Anthony

Stephen Jeaco
Xi'an Jiaotong-Liverpool University, China

Abstract
Professor Laurence Anthony is probably best known for his corpus tools – including the freeware concordancer called *AntConc*. He is also known for his contributions to fields of English for Specific Purposes (ESP), Natural Language Processing (NLP) and Educational Technology. In this interview, Laurence Anthony talks about the early beginnings of his interests in science, programming and language learning; some of the background to *AntConc*; his views on needs analysis and other aspects of English for Specific Purposes; and his perspectives on machine feedback and machine evaluation.

Keywords
Educational technology, corpus tools, needs analysis, machine feedback, machine evaluation

Laurence Anthony, PhD is Professor of Applied Linguistics at Waseda University in Japan where he is the Director of the Center for English Language Education in Science and Engineering. He is the creator of many corpus tools including *AntConc* (Anthony, 2019) – a freeware concordancer. His computer software tools cover monolingual and parallel concordancing, text processing, tagging and corpus creation. His first degree was in Mathematical Physics (UMIST, Manchester, UK), and then he went on to complete his MA in TEFL/TESL and Ph.D. in Applied Linguistics at the University of Birmingham, UK. His main areas of research include corpus linguistics, English for Specific Purposes (ESP), natural language processing (NLP) and educational technology. As a great believer in the power of corpus tools and having worked in ESP and TESOL myself, it was a great pleasure to "meet" Laurence online in September 2020 and discuss at length his background motivations and current trends in language and technology. As the previous few months had seen an unprecedented intensification of online learning and teaching, a focus on technology was particularly poignant and it was interesting to hear how Laurence's many years of experience give rise to his current perspectives and his vision for the future. In this interview he talks about how he got started with programming, some background to *AntConc*, and his views on the present state and the future of English for Specific Purposes, machine feedback and machine evaluation.

Address: Department of Applied Linguistics, HS431, Xi'an Jiaotong-Liverpool University, 111 Ren'Ai Lu, Suzhou Industrial Park, Suzhou, P. R. China
Email: steve.jeaco@xjtlu.edu.cn

1 Corpus Tool Development

How did you first get started with computer programming?

So it's difficult to compress this down into a short story because it's quite a long journey in some sense. But I started programming basically when I was about 11 years old. I've always been interested in science and engineering – probably through my dad's influence. When I was very young, my dad bought me an electronic kit – if you know what those are; you can wire things up together and make little circuits and stuff. And I did that from when I was very young. And in 1981, which is when I was 11 years old, the company Sinclair built the first under 50 pound computer for home use, called the ZX81 and my dad bought the family one of those. At the same time, my secondary school in Huddersfield bought one as well. So that's when I got into programming. With the ZX81, you basically had to program right from the beginning. You would type 10 PRINT "Hello" 20 GOTO 10 and you would see Hello, Hello, Hello scrolling down the screen and stuff. So I started there at 11 with just basic programming. I went on to use the Spectrum computer and then the Acorn Electron computer, which was a budget version of the BBC micro that started to be introduced into schools over the following years.

I remember when I was 14, in our Physics class we were learning about lenses and how you can use them to correct long and shortsightedness. I wrote a little computer program that would simulate replacing lenses, so you could see how the beam of light would be focused. When I showed it to the Physics teacher, he brought the whole class into the computer room, and they sat there and watched me demo this little program. In some sense, I think that was probably the real start for me in terms of where I am now. It was the idea that you could get computers to do something really cool like a physics simulation. I think it also gave me a huge boost of confidence, being a little kid in a class knowing more about programming than the teacher and wowing thirty other kids in the room with computer technology. That really was a massive experience at the time. I also got a boost of confidence from taking an O-level in programming one year early when I was 15, studying with adults using the mainframe computer at a local technical college. After studying mathematics, physics, and chemistry at college, I went to university, where I did mathematical physics. There, I did coding in Fortran as part of the degree. Then, I came to Japan and had a couple of years where I didn't code. But during my masters and PhD studies, I started doing a lot of programming again to create programs with graphical interfaces, writing mostly in Perl. And basically I have not stopped coding since then. So, I started as a kid with Basic; I went on to Fortran; then went to Perl, and now I mainly code in Python, JavaScript and PHP.

How did you first get interested in language analysis / language teaching?

I always wanted to do something which had a really big, important impact on the world. When I was a little kid of around 7, I wanted to be an astrophysicist. Funnily, I went to university and did it. I studied mathematical physics which has astrophysics as an important component. After doing that, I kind of lost real touch with reality in some sense; mathematical physics is very mathematical and very theoretical. So, I took a year off after finishing university and came to Japan. And as you do when you come to Japan for a year off, I started teaching conversation English in schools. I also gave a couple of lectures to the community of Okayama where I was living. At that time, I started getting to know people in the area and started talking to scientists and engineers who worked at universities. And then I realized that there was a lot of great science and engineering coming out of Japan, but a lot of the people who were doing it couldn't communicate it well. They really struggled with writing their research papers, presenting their research, and discussing it. This language barrier was so huge and it really struck me. So, I kind of found a new mission. I had done a lot of maths and physics when I was at university, and I could use this knowledge to help them address the problems they had. I thought that's my future. My mission became one of helping scientists and engineers communicate. So, I went on to do a Masters in TEFL specializing

in technical writing, and followed that with a PhD in applied linguistics, and that's been my core aim ever since. So my interest in language analysis and language teaching started with this desire to help scientists and engineers communicate.

And how did these interests first come together for you?

It's interesting because when I did my masters, I found that a lot of TEFL and a lot of humanities research really didn't click with me at all. It was this kind of wishy-washy, qualitative, impressionistic stuff, which I'm not really a fan of at all. I did enjoy the masters; it was a fantastic program to give you the basics of EFL teaching: grammar, syntax, pragmatics, methodology, speech analysis; all the important things you need to know to be a professional in our field. But one course really struck me. It was a course in corpus linguistics with John Sinclair. And that was the first time in the whole programme where I saw a scientific approach applied to language. I liked the fact that properties of language were being counted properly, and from this data we could formulate hypotheses and make predictions, and so on. That really clicked with me – this idea that you could treat language in a scientific way. And I also did a course on discourse analysis. At that time the discourse analysis course wasn't very corpus-y at all. It was a very impressionistic kind of analysis; looking at the move structure of language and things like that. But, that course really connected with me as well because I could easily see how the concepts could help people communicate better. I could use those concepts to explain how you write a research paper, because you start off with this move and then you shift to this move and so on. So corpus linguistics was really interesting as a scientific approach and discourse analysis was interesting as a way of looking at language that could fulfil my goal.

You mentioned just now that the course you took wasn't very corpus-y, and you make an interesting point about different ways of thinking about and doing research. Can you explain your views on this a bit more?

In mathematical physics, terminology is very important, and it is always carefully defined. But, when it comes to linguistic research, I often find scholars don't define exactly what they mean, or they add a very tiny modification to an existing concept and coin a new name for it. This makes it very difficult to build on existing research as it's not always clear what the common knowledge is. So, in a sense, I would hope that people that study language as part of their research would use language more carefully themselves. I always found that quite intriguing. One other difference that I struggled with at the beginning was the focus on word-based descriptions of phenomenon and results in linguistic studies, as well as the heavy use of citations and quotes. I came from a background where descriptions and results were often presented in mathematical form or through figures and tables.

But, you might ask what about now? In many ways, linguistic studies have become 'more scientific' in the sense that they rely more heavily now on controlled experiments and statistics to prove points. But, I sometimes wonder if the researchers conducting these studies really understand the subtleties of the methods and statistics that they are using. We also still see a huge number of studies that are largely descriptive in nature and have such a narrow focus that I wonder how much knowledge they can contribute to the field as a whole. As someone who does a lot of journal reviews, I would say that the main reasons for rejecting papers are (1) the study has very limited value (it's not addressing a real-world problem), and (2) the methods do not allow for a solution to be found.

But getting back to my Master's, my thesis was a Swales (1990) inspired move analysis of computer science research abstracts – like a classic analysis; everybody does that kind of thing. When I went on to the PhD, then it really did come together, because I thought: How can we use corpus linguistics and

apply it to discourse analysis? How do I bring in my own view? I want to do a scientific analysis; I want do it in the best way possible. And that would involve using computer science, artificial intelligence and natural language processing. I hadn't done any NLP at university; it was all mathematics and physics. But it's still the same kind of idea: coming up with a model to represent a real-world phenomenon, and then building that model often with computer software. So that was what my PhD was – an automated discourse analysis tool using AI and NLP methods. That was the first time that I combined programming and language analysis, applying it to science communication. It's what I still do today.

Your software tools are well respected in terms of functionality and ease-of-use. It is great that most of your tools are also completely free. What are your main reasons for making your software available as freeware?

I'm primarily an academic, right? So I get paid by the university to be an academic. What does that mean? Well, we have a duty to make discoveries to make the world a better place. I think that's what research is ultimately trying to do. And we should also be trying to pass on our knowledge to the next generation – to instruct students to become better people than yourself; to give them the tools and skills to know more and be better than you are. In terms of the business of the university, we should also be promoting the institution, that is, we represent the institution. And in those senses, having some well-used, respected software tools serves all of those purposes: it allows students to be able to use the tools to do their own research, it promotes the university, and it gives something back to the community. So that's why I make them free. If I charged people for them, the number of users would be dramatically fewer – I'm almost sure about that. I've gotten emails from many people that said they weren't in a position to buy them. So it's nice to have just a very open, clear setup. It also in some sense happened by accident, I would say. When I've seen the need for a tool, I've often just built it and released it without really thinking too carefully about the financial gain for these things. They're sometimes just little tools and sometimes even just wrappers around other tools that allow people to do things more easily. I've got into the habit now of releasing everything I make. So, I'm not really thinking too far forward about profits or anything like that. I just hope they are useful tools.

And there's a funny story about *AntConc*, because I actually developed it by accident. I was learning to code in Perl for my PhD, so that I could develop this automated discourse analysis tool. As a little practice exercise for developing an interface, I thought I would build a KWIC concordancer. And then a friend of mine who was working at a different university needed a KWIC concordancer and I happened to have just finished it. So I said you can have this. And when they used it they gave me feedback. And I built some more features onto it. And that's just been continuing ever since. So it's really, by accident.

One interesting question that you didn't ask, though, is why it is not open source, because a lot of people get confused about the difference between freeware and open source tools, and mine are not open source. Some people get the confused idea that it's because I want to keep it to myself or that I don't like open source or something. But it's really nothing like that. It's quite a simple answer. My target audience are not generally programmers, so they're not likely to contribute new code to the project. If more people expressed an interest in contributing code, I could certainly see my approach changing. A lot of open-source projects are more like that – projects where lots of people are building on the tools and using them as they build them. Up till now, though, the most common requests to make the project open source have been from profit-making companies, who I suppose might want to avoid building such a tool themselves.

I should also mention just one more thing about this. I do create completely commercial tools as well. So I've got my own company called AntLab Solutions. And I do see that as a completely different aspect of my work. So as an academic, I create tools, release freeware software, collaborate with people and so

on. And as a software developer at AntLab Solutions, I am a commercial software developer. If people need me to build a software tool for them, they will give me a list of specifications, I will give them an estimate of the cost, we'll sign license agreements and so on, and then I'll build the tool for their purpose. And that's been successful too. But you might never hear about it because when I build a commercial tool, I don't announce it. The tool belongs to the client, not me.

The logos for your apps have a black ant on them. Do the logos for your apps have any special meaning or symbolism for you?

Well, originally, I didn't have any logo on my apps – if you look at the early versions of *AntConc*, for example. But my name's Laurence Anthony. And Anthony is pronounced without the h so it's "an" + "tony". And when I made the very first version of AntConc, I released it to my friend and I put it online as Conc. It was just CONC.EXE, because I didn't think anything further than concordancer is a long name, so let's call it Conc. After I released it, I found just days later that another program called "Conc" already existed on the web. I panicked and thought I'd have to change the name immediately. And the first thing I came up with was taking "Ant" from my surname. So I stuck "Ant" on the front of it, without thinking much more about it. But, I quickly realized that it's useful as a namespace, so that if you put "Ant" on the front of something, it's probably not going to clash with other names – it helps in that respect. And then I kind of started getting to like that ant. I don't know if other people like ants, but they represent to me this kind of meticulous, hardworking, careful kind of creature. When ants build their nests, they collaborate; they work together; they make something that's carefully built and useful for everybody. I kind of like that image. So I've embraced the "Ant" design; now it's actually the logo for my company as well.

When developing a new application (or developing a major update), which parts of the process do you enjoy most? And which are the most frustrating or stressful?

So definitely, the most enjoyable part is talking to people and finding out how they complete their work normally without the help of any software. Often it's a description of a time-consuming, very laborious task. And they often spend days, weeks, or months doing this laborious task. When I listen to what they do, I start thinking about how I could help them do the task better, faster, more efficiently, more accurately, or more rigorously. It's really nice to see the tools that I've worked on being used. That's probably very egotistic in some way, but it's a proof, in some sense, that my work is providing some kind of value to the world. And that comes back to that story I said earlier about physics. Also, it's like a series of puzzles. It's like a game, and I very much like that.

I would say the most stressful aspect is when something doesn't work, like at a trivial level, I have this code that is supposed to produce an even number of rows and it keeps producing an odd number. And I don't know why. I have many times spent hours figuring out a problem which ended up just being a comma in the wrong place or some other silly syntax error!

What advice would you give to someone getting started with programming for corpus tools or apps for the classroom?

Actually, I've just written a chapter for a book as an introduction to programming (Anthony, 2021), so I would say read that chapter because it talks about the background to real programming: what programming is, what a programming language does, how to think about coding, what are the different

coding strategies, and so on. And I would say to program well, you need to have a reason to code. It's like any language. I would also say that you should pick a popular general-purpose language such as Python or JavaScript, because you'll have a bigger community of programmers around you that can answer your questions. And basically always be prepared to learn new things. Start with small, tiny apps that do useful things, for example something to rename all the files in a folder, or to search-replace text. Use online tutorials that take you through not only the syntax but also the logic of programming – how to think in a programmatic way.

2 Language Learning and Teaching

In the book Introducing English for Specific Purposes (Anthony, 2018) you discuss the four pillars of ESP. Can you describe these briefly? Which of these do you think is the least understood or the area in most need for in-service teacher training and research?

My book was heavily influenced by Hutchinson & Waters (1987) and Nation & Macalister (2010). There are a lot of influences from them in the model and throughout the book, and I do want to recognize them. The four pillars are: needs analysis, learning objectives, materials & methods and evaluation. And I would say these are common sense. In the book, I give an example of what you would probably do if you were hired to teach a call centre staff training class in English. The first thing would be a needs analysis. So that would be: you go to the company and you find out who the stakeholders are, and then find out what the company representatives want, and what the call staff currently have to do in the jobs, and what they can't do right now. And, if you're forward thinking, you would also ask the call center management what they want to achieve from the program. You would find out whether the staff want to do the course or whether they were pushed into it. So, needs involves finding out about necessities, lacks, and wants of all the stakeholders.

I'd say that a needs analysis is the least understood area of ESP. If you imagine a traditional university setting, it's really easy to fall into the trap of just thinking needs is *wants*. The students are in the room, and as a teacher you want to know what they want to learn.... So, you give them surveys asking about what they want to do in the class and what do they want to talk about, how they want to study, and all of these questions. And then you start doing just that. But, in the book, I choose the example of a call center for a really specific reason. In a workplace setting, it's much easier to imagine that the main stakeholders, the company executives in this case, will not know what language problems the workers have or what they need to learn. Clearly, they will have some basic desires for the course. But, it would be barmy to start asking the company executives, "What do you want me to teach the workers," because they will say, "We don't know what they need; we hired you to tell us that!". So, this company example really helps clarify that needs is not just about wants. It should start with the question, "What's necessary to complete the job?", this is the necessities part. Then, "What can't the people do?" – the lacks part. And finally, "What does everybody want to get out of the course?" – this is the wants part of needs analysis.

Once you've got the needs down, it's pretty straightforward then to establish some learning objectives, which is the second pillar. And then you get the materials and methods, which is the third pillar. And I chose that wording very, very deliberately. Because the third pillar – materials and methods – is a phrase that is common in science and engineering. Actually, I think people often kind of conflate the two. But, it's better to separate them and first think about the language that they need to know and only then think how best to teach it. And then finally, the fourth pillar is evaluation, which is something, again, that is often misunderstood, but not as badly as needs. Teachers in a regular academic classroom think evaluation is simply a measure of how well the students did on the course – so learner success. And, yes, you need to measure that; you need to see how well they improved. But, it's also important to measure how well all the stakeholders achieved the goals that they were trying to get to. So it is not just about the learners' goals but also those of the administration.

Machine feedback and evaluation of language learner writing has been developing over the last two decades. How do you feel about current developments in this area and what do you see as the main limitations or challenges right now?

As you know, I have done some commercial work in this area, especially in Japan, with The Japan Institute for Educational Measurement (JIEM), where I helped build their CASEC-G grammar tutorial system. That was about 10 years ago, but it's still running strong. What I found, even at that time, was that the problem with machine feedback and evaluation was precision *and* recall. So, what you find is that both of them are pretty weak in an automated system. So, a system might flag a problem, but you look at it and it's not a problem at all. So that's the problem of precision. It's giving you feedback that is wrong. Spell checkers, in some sense, are a bit like that. Their precision is pretty good these days. But they will flag, for example, a name as being misspelled, even though the name is totally fine. With spell checkers, you can usually look at the flagged word and you can immediately notice that it's rubbish. And you just ignore it. But when a grammar checker flags a grammatical error, like you should not be using passive voice, it's more difficult to know if you should ignore it or not. Maybe a native speaker can do this, and we often hear teachers tell students to just ignore some of the flagged grammar errors in Microsoft Word. But how do learners know whether or not to ignore one of these things? So precision is a really important aspect. And, what we find is that if the precision is too low, the tool itself becomes unusable. I've found that many automated tools are unusable when it comes to actual in-class teaching; I can't rely on the tool giving the students valuable feedback that is going to help their work, except perhaps spelling and some common article and subject-verb errors. So, that's the worst problem with automated tools. The precision problems are really bad. So, I would recommend anybody who works on these systems to improve the precision; forget about trying to cover many different error types; get precision as good as you can. So, when the system flags something as an error, like "This paragraph has a weak topic sentence." it really needs to be an error, not some *possible* error.

Then, the second problem is recall, the number of actual errors flagged. I think companies often try to improve recall, because they want to have some results show on the screen. They want to have a screen with lots of things flagged. I would say, in some sense, the current systems are pretty good at recall. They do flag lots of things. But the precision is so low that many of the things they're flagging aren't problems at all. And, they are missing lots of other errors, too. One good example is identifying singular countable noun article problems. I'm sure a system could be designed to do this very well, but a commercial software development team won't want to just focus on this because it's too narrowly focused. Maybe this is something only an academic could work on. But isn't that bizarre? Because in the medical industry, that's very much the kind of tool that they focus on. They have extremely precise systems, right? In medical examinations, we start out with a system that has very good recall – like for example breast cancer detection. So every woman goes and has an x-ray. If they have some kind of shadow on the x-ray, they'll get called in for a second test. So, the aim here is to have as high a recall as possible. The doctors don't want to miss anybody. But, when it comes to the second test to confirm if the cancer is there or not, they need precision. They don't want to start cutting people up if they don't actually have cancer. Right? So, the confirmation tests have to be extremely precise. But in English language teaching, we don't seem to have that model at all. So I think that's the biggest challenge.

We can see that deep learning AI applications have massively improved over the last few years. So, I think they'll start to have a massive influence on TEFL teaching soon – for example in feedback and evaluation systems. There's no reason why they can't. Even now, AI systems can generate very natural writing and speech. There's even a system that can take a paragraph or even a fragment of a paragraph, and just continue in the same style and complete an essay – and quite convincingly too. So it seems obvious that it will soon be possible to reverse that process and have the AI system not generate the language, but evaluate the language that somebody has created and judge it in terms of naturalness and

accuracy and so on. It hasn't been done yet, but I think it's probably because companies have more commercially profitable tasks to work on. And, of course, English language teachers are pretty adverse to the technology. Many question the possibilities of these systems quite strongly and reject them. So, even if a system worked, I could imagine a lot of teachers not wanting to use the system because they didn't trust it. And, it also puts people out of jobs, which is another problem altogether.

There's another issue about this; auto-text generation and machine translation are incredibly good now. So, this raises the question about what the goals of a language program should be. For example, do students need to actually learn how to write in English anymore? If they can write in their first language and have a system very accurately translate it into a second language, is that enough?

I'm not saying it is for everybody, of course. But I think it raises a new question about what the learning goals should be. So here's a question that I have to address whenever we consider revising our own program: should we encourage students to use machine translation systems in their writing? We know that a machine translation of a student's first language can often be better than what they produce when they try to write in their second language. So, should we say to them, go ahead, just use Google Translate, and then edit the translation? Or should we say, don't use Google Translate because it's so good you might never want to directly write in English again? If we say to students use machine translation all the time, they can probably complete all their tasks, but it means that they're always going to have a crutch that they need to rely on to be able to do anything. The other problem with this is how do the learners themselves evaluate the output of these systems? How do they judge whether or not the system accurately captures their intended meaning? Of course, I can look at the students' machine translation output and say, this is great. But how do they do that?

So they're powerless. We could have a situation where they put some first language into a system, it then produces some output that they don't understand, and they just hope that it's actually representing what they want – which is not a good situation to be in. So, what should we do? I really don't have an answer yet.

So, what's your approach right now?

For my students, I tell them, use machine translation. I'm fine with it. Because I know that whatever I tell them, they're probably going to use it. So I may as well start off embracing that reality. But, I also then teach them aspects of good writing regardless of the language – discourse and argumentation and hedging. If they can apply these basic foundations in all their writing they will be better off, and any machine translation will also end up being pretty good, too.

We can also think of the problem at a more personal level. If you think of students in physics, for example, they've got their research, and they may have a great result that's going to change the world. But, if they're non-English speakers, they might not be able to get their work published because they can't write it up in English. But, should we force them to slow down their research and take the time to acquire English? Or should we encourage them to write up their research in their first language and put through it a machine translation system? I suppose it's a matter of balancing all of the different constraints, right? Time constraints, ability constraints and value constraints.

3 Developments of Technology Now and in the Future

What do you see as the most important developments in language learning and technology over the last 10 years?

When I was thinking about the answer to this, AI didn't cross my mind at all, which is interesting because

I actually don't see AI as having a major impact on language learning to date. I think students use AI as part of machine translation when they're trying to complete a homework task, despite the teacher telling them not to use it. In fact, I don't see many teachers embracing Google Translate and similar AI technologies in teaching at all. I would almost say they're aggressively blocking the use of those kinds of technologies. What I do see as the big development over the last 10 years is the democratization of learning – this is basically online learning with things like YouTube, MOOCs, Coursera, FutureLearn, and those kinds of platforms. Over the last 10 years they've vastly increased in numbers and scope. So I would say that's a huge difference; that's what technology has brought to language teaching. It's not AI; it's the fact that students now watch a video on YouTube or take an online course and learn language that way. The second one is video conferencing – using systems like Zoom. Over the past year, many of us have experienced very smooth meetings and classroom sessions with 10s or even 100s of people connected through video. Also, the prices for high quality home studios (cameras, mics, editing software, etc) have dropped making it much easier for somebody to design and create an online course and also lower the barrier for a student wanting to take it.

The things I've just talked about are in some sense not direct language learning technologies. In terms of direct language learning technologies, I would say that corpus tools have developed a lot over the last 10 years, but not as much as we would have hoped. They're faster, they're easier to use, they've got more features, but there are still many areas in which they can be improved more. And then there's the whole field of CALL (Computer Assisted Language Learning). A lot of new technologies are discussed in CALL research, and they have certainly improved language learning. But many of them were created in another field and just applied in language teaching later. Is that a development in language learning or just a reaction to more general developments in technology?

What's clear is that technological developments are changing the world in quite fundamental ways. Some of these changes have already had a direct impact on the needs of language learners and the ways we should approach language teaching. And, no doubt, as the technology develops further, we're going to have to ask ourselves more and more questions about what we should be doing in the classroom. In many ways, it's a worrying time, but I think it's also a very exciting time in our history.

References

Anthony, L. (2018). *Introducing English for Specific Purposes*. Abingdon: UK. Routledge Press.

Anthony, L. (2019). AntConc (Version 3.5.8) [Computer Software]. Tokyo, Japan: Waseda University. Available from https://www.laurenceanthony.net/software

Anthony, L. (2021). Programming for Corpus Linguistics, in M. Paquot & S. T. Gries (Eds.) *Practical Handbook of Corpus Linguistics*. Berlin: Springer.

Hutchinson, T., & Waters, A. (1987). *English for specific purposes*. Cambridge, UK: Cambridge University Press.

Nation, I. S. P., & Macalister, J. (2010). *Language curriculum design*. Abingdon, UK: Routledge.

Swales, J. M. (1990). *Genre Analysis: English in academic and research settings*. Cambridge: Cambridge University Press.

Stephen Jeaco, PhD, is an Associate Professor at Xi'an Jiaotong-Liverpool University. He has worked in China since 1999 in the fields of EAP, linguistics and TESOL. His PhD was supervised by Professor Michael Hoey and focused on developing a user-friendly corpus tool - The Prime Machine - based on the theory of Lexical Priming.