**The Legacy of Advanced Informatization**

Preface

The present proceedings from the Symposium on the Legacy of Advanced Informatization, which was held at the Computer History Museum in Ljubljana on June 13 2022, represents an account of a variety of views on the challenges and the need for the systematic storage of software heritage. It introduces the topic into Slovenia’s contemporary professional discourse, with a view to laying the foundations for future frameworks that could help our society preserve the memory of the digital layer of human activity.

At the Computer History Museum, we have felt the need for such a broad discussion as a burning problem in our own work in media and digital archaeology, processes and attempts to reconstruct the ground-breaking digital imprints of Slovenian society. Our thesis for the expert panel was that all too often we let slip into oblivion what we interact with on a daily basis when we, the users of digital technologies, willfully ignore the layer that enables the no longer miraculous yet increasingly complex and misunderstood interface between humans and machines. What has been preserved has actually been preserved by chance thanks to conscientious individuals who were active either within organizations or privately. The goal behind this collection of papers is to prepare illustrative groundwork for future elaboration and suggest a more systematic approach to preserving what creates the digital human experience from inanimate hardware.

We intentionally invited a very diverse group of experts to the symposium to ensure a wide array of perspectives and thoughts, from the very developers of ground-breaking software in Ljubljana and Slovenia to authors of humanist interpretations and good practices that have been established in other segments of research or certain highly regulated industries. We opened legal questions regarding the vague definition of software intellectual property, and pointed to the effects of constant joint efforts in the development of the city and also on the confidence of software developers who grew in an environment boasting a distinct institutional and grassroots software activity.

In the introduction Dr Roberto di Cosmo frames the purpose of the symposium by presenting the genesis and work of the UNESCO Software Heritage foundation t, of which the Computer History Museum is an ambassador. He provides the principal arguments for recognizing the status of software as a form of intangible heritage as perceived by UNESCO, and discusses a practical example of software heritage archiving. Practical examples and an overview of decades of software development are offered by Dr Saša Divjak in computer sciences, Dr Primož Jakopin in linguistics, and Franc Zakrajšek in urban planning and digitalization of architectural cultural heritage. Divjak discusses the issues of software solutions obsolescence, while Jakopin explores the issues of archiving from the viewpoint of subsequent museum display. Zakrajšek delves into the development of digitalization in a specific field – urban planning – and uses various technological solutions to illustrate how development in Ljubljana kept up with global trends. Furtheron, Silvana Žorž, MA, connects the theoretical and practical aspects of the importance of values in software development based on an analysis of the company business and engineering culture in HERMES SoftLab, thus opening a discussion about the ability to preserve such intangible yet crucial heritage as a values system. This very human aspect is addressed also by Boštjan Špetič in an article uncovering insights into the reconstruction of the Iskra Delta Partner computer, archaeological findings from a hard drive, and the actual effects of presentation in the museum. Dr Andrej Pančur highlights best practices and regulatory frameworks in the digital humanities, while Dr Nataša Milić-Frayling illustrates best practices through examples of long-term software storage in a highly regulated industry environment, and in a demanding artistic environment where the authenticity of the user experience is of paramount importance. Matija Šuklje discusses the complexity of legal interpretations of software which should be taken into account in organizing systematic storage, reconstruction and presentation to the public.

The expert panel raised questions yet also offered starting points for forming conclusions. The underlying idea intertwining all points of view was outstandingly human.

Software is transient.

Software is transient yet immortal in its idea, in the materialization of human inventiveness, the encapsulation of the spirit of a certain era, and the value systems of specific groups of people. It is an expression of our culture. It is our heritage.

In Slovenia there is ample room for improvement in terms of the storage, registering, conservation, and restoration of software heritage and also for different interpretations and presentations to the public. The first step will be taken as soon as we realize that software is our intangible cultural heritage and as such deserves a thorough treatment to avoid it disappearing forever. The history being discussed here is truly recent, barely more than half a century old. But it is vanishing rapidly with each update of user systems, and with each generation of hardware. Hence now is the right time to wonder what will be left for us to show future generations in the next 50 years. Will we truly be able to properly explain how digital interactions shaped our society without authentic, interactive experiences? Will we collectively forget about them?

Gaja Zornada,

Head of Computer History Museum, Ljubljana, July 2022

# 1. Software Heritage: the universal archive of software source codes

Roberto Di Cosmo

Software is the engine of our industry, the fuel of innovation, the essential instrument we use to communicate, to maintain ourselves, to perform any kind of transaction and operation, to organize ourselves in society and form our political opinions. Software is crucial to the functioning of economic, social and political organizations, whether public or private, whether on mobile devices or in the cloud. It is also the indispensable mediator that enables access to all digital information, and it is, along with articles and data, one of the pillars of modern research (Noorden et al., 2014)

Software therefore represents *an important part* of our scientific, technical and industrial heritage.

If one looks closely, it is easy to see that the real knowledge that is contained in software is not in the executable programs, but in the "source code", which according to the definition used in the GPL , is "the preferred form for a developer to make a change to a program."[[1]](#footnote-0) Source code is a special form of knowledge: it is made to be *understood by a human being*, the developer, and can be mechanically translated into a form to be *executed* directly on a machine. The very terminology used by the computing community is telling: "programming languages" are used to "write" software. As Harold Habelson wrote as early as 1985, "programs must be written first so that other human beings can read them" (Abelson & Sussman, 1985).

The source code of software is therefore a *human creation in the same way as* other written documents, and software developers deserve the same respect as other creators.

Software source code is therefore valuable heritage, as already argued by Len Shustek in a fine 2006 article (Shustek, 2006) as well as by Donald Knuth (Knuth, 1984), and it is thus essential to work on its preservation.

This is one of the missions of Software Heritage, an initiative launched in 2015 with the support of Inria,[[2]](#footnote-1) to *collect, organize, preserve and make easily accessible* all publicly available *source code on the planet*, regardless of where and how it was developed or distributed.

## *A complex task*

Archiving all available source code is a complex task, and as detailed in the literature (Abramatic et al., 2018) one must deploy different strategies depending on whether one seeks to collect open- or proprietary source code, and one does not treat source code that is readily available online in the same way as source code that resides on older physical media.

For open-source code that is readily available online, the most appropriate approach is to build a harvester that automatically collects content from a wide variety of collaborative development platforms, such as GitHub, GitLab.com, or BitBucket, or from software package distribution platforms, such as Debian, NPM. CRAN or Pypi.

For the source code of old software, a real process of computer archaeology must be set up, and we have already started this work in a collaboration with the University of Pisa and UNESCO that has resulted in the [SWHAP](https://www.softwareheritage.org/swhap) process that has been used to find, document and archive software that is important in the history of computing in Italy, and which has recently been extended with the [Software Stories](https://stories.softwareheritage.org) project, which aims to highlight all the historical elements around software whose source code has been found.

## *A universal mission*

The founding principles of Software Heritage are (Abramatic et al., 2018; Di Cosmo & Zacchiroli, 2017): the systematic use of open-source software to build the Software Heritage infrastructure, so that its operation can be understood, and replicated if necessary; the construction of a global network of independent mirrors of the archive, because a large number of copies is the best protection against loss and attack; to have a non-profit, international, multi-stakeholder structure, to minimize the risk of having single points of failure, and to ensure that Software Heritage will indeed serve all.

For such a mission, institutional legitimacy is required, as well as a real capacity for openness to enable a broad consensus. The framework agreement signed between Inria and UNESCO on April 3, 2017, and renewed in November 2021, is essential in this regard.

## Past, present, future: much more than an archive!

Software Heritage now has an infrastructure that grows day by day, and if the bulk of the archive's content is the result of automatic harvesting, some real treasures are beginning to be uncovered through the patient work of recovering significant historical software, following an acquisition process that has been developed in collaboration with the University of Pisa and UNESCO.[[3]](#footnote-2)

Figure 1: Number of projects, source files, and versions archived in [Software Heritage](https://www.softwareheritate.org/archive) as of June 2022

While exhaustiveness is still far from being achieved, the archive already contains the largest corpus of source code available on the planet, with more than 180 million archived origins, for over 12 billion unique source files, each equipped with an intrinsic identifier based on cryptographic hashes (Di Cosmo et al., 2018)

This unique infrastructure has a multiple mission: of course, it is about preserving for future generations the source code of the *past* that made the history of Computer Science and the Information Society, but also, and above all, we are trying to build a *very large telescope* that will allow us to explore the *present* evolution of the *software development galaxy*, in order to better understand it, to improve it, and to build a better technological *future.*

## *A strategic issue, which needs to be known*

The Software Heritage archive is already the most important collection of source code in the world, but there is still a lot of work to do, and a wide range of players, from those working in cultural heritage to industry, from research to public administration, must be brought together to achieve this. To make this possible we are counting on a growing network of [ambassadors](https://softwareheritage.org/ambassadors/), including the Computer History Museum in Ljubljana, Slovenia.

It is clear that software has now become an essential component of all human activity, and therefore unrestricted access to publicly available software source codes is becoming a digital sovereignty issue for all nations.

The unique infrastructure that Software Heritage is building, and its universal approach, is an essential element to meet the challenges of digital sovereignty while preserving the common good dimension of the archive.

It is therefore of the utmost importance that institutional, industrial, academic and civil society actors grasp the importance of these issues, and that Europe positions itself quickly, by providing the necessary resources to make Software Heritage grow and last, by taking their place alongside other international actors who are already committed to this project, and by supporting the creation of an international non-profit institution that will carry out this mission over the long term.

2. Development and obsolescence of programs – the programmer’s challenge and nightmare

Saša Divjak

I belong to the older generation of programmers, with my first programming experience dating back to 1967 and the legendary Zuse Z32 computer. The beginning of the 1970s was marked by punched cards and perforated tape. Micro- and sometimes minicomputers were programmed in an interesting way back then. Computers were equipped with a teletype, a peripheral unit that allowed typing, printing, paper tape reading and punching. Software was prepared and run in this order:

The first step was to set the computer up with the text editor, saved as binary code on the tape. Then you wrote a program and punched the source code (often in assembly language) to a new tape. Next was reading of the assembler, which was again coded in binary form on a separate tape. And then reading of “your own” program in the source code could follow. As far as I can remember, reading was done in two steps as well because the assembler needed at least two phases to complete the process. Finally, if everything went (fairly) well, you could punch a new tape with your own program in binary code. Then came reading of the new binary program and its execution. What now takes a fraction of a second used to take quite a few minutes and you could only hope you hadn’t made a mistake, or the whole process had to be repeated.

When programming microcomputers, your own machine code program was usually “burned” into EPROM (an integrated circuit which formed part of the microcomputer’s memory). This is how we developed various microcomputer-supported automation protocols. But programmers tend to get things wrong. Because the whole cycle took a while to complete, we often (whenever possible) made corrections directly in the machine code, thus skipping the time-consuming punching process. As a result, the program worked correctly in EPROM, but the source code did not match anymore. Which is very wrong, of course.

Mentally jumping back and forth from assembly language to the machine code of the program was nothing special. Afterall back then , we often entered the bootstrap loader using the switches of the computer console. It became a habit, and it’s perhaps no surprise that we knew sequences of dozens of commands at the machine level by heart. This is an example of the kind of experience I had with the first generation of Digital PDP 11 computers in the 1970s.

Before I continue, I would like to point out that I lectured, at the Faculty of Computer and Information Science, on programming, systems software and operating systems, and, what I loved the most, computer graphics. This is reflected in some of the memories I mention further on.

There are now over 9,000 registered programming languages in the world, and the way we program computers has changed significantly. In the 1980s, we introduced the C programming language to the computer and information science study course at the then Faculty of Electrical Engineering and Computer Science, and to this day it serves as our “Latin”, and a solid foundation for many other programming languages. In 1997 we introduced the Java programming language. Being the main lecturer, I always worried about the constant development of this fresh programming language, which actually went through some major transformation in the following years. We used these programming languages for various projects.

On the other hand, in the late 1990s we already used JavaScript for programming web applications, and it is still just as popular. Later on object-oriented programming was joined by other approaches as they emerged. Among them was component-oriented programming, which used various problem-oriented libraries to build new applications by combining their own source code and function modules offered by the libraries. Why reinvent the wheel when solutions are already available? For instance, supporting 2D and 3D graphics or running various more or less complex, tested and effective routines? This type of approach requires the knowledge of APIs (Application Programming Interfaces), but speeds up the development process quite substantially. As a consequence, the rapid development of new versions eventually always leads to different components of our applications being incompatible with each other, therefore making the software obsolete. Particularly dangerous is the “mixing” of different technologies from different developers, who each follow their own standards and guidelines. One of the examples would be the now-forgotten Virtual Reality Markup Language (VRML), which emerged along with JavaScript and enabled quite decent 3D visualization and 3D scene animation at the time. Combining VRML and JavaScript languages enabled creating very attractive 3D visualizations and interactive simulations of natural phenomena. These examples are now completely obsolete, and can no longer be displayed (truth be told modern 3D graphics are something completely different). You may remember the Java applets, which enabled various applications (and 3D visualizations) in our browsers. Then it transpired that such technology had too many security flaws, because it could work outside of the advertised supposedly safe “sandbox”. One browser after another disabled these applets in their upgraded versions. Today it is only possible to see them only on computers with purposefully installed obsolete operating systems and browsers. Often we can achieve this using virtual machines on the computer. Developers were eager to find solutions in similar technologies in order to urgently address this issue. Many of them therefore switched back to the once-popular Flash and its programming language, called ActionScript. In some cases that helped preserve up to 80% of their code. Later it transpired we were hopelessly wrong. Flash, and ActionScript along with it, are now extinct. JavaScript and jQuery were the way to go, due to their high programming efficiency. Once again various libraries came in handy for the effective and uniform development of user interfaces and similar systems etc..

Nowadays we are surrounded by numerous mobile devices, such as smartphones, tablets and large or small laptops and desktop computers. Due to a large variety in screens, responsive design was developed to enable a similar user experience across all devices, regardless of the size and resolution of the screen. What is helpful in developing such applications? The jQuery Mobile library seems like a logical step, because it enables the planning of graphic user interfaces that can suit various devices. But there are other options, such as the popular Bootstrap. This was all well and good, but only for a short while, and jQuery was upgraded to the next versions (version 3 at the time of writing this paper). Development tools are suddenly able to send the developer warnings about parts of the code being obsolete or deprecated. If nothing else this is rather unpleasant, and we should worry about code that is becoming old and will probably become unusable over time. Well, we could migrate the code to comply with the new rules. This task, although time-consuming and painfully tedious because it requires a systematic approach, is necessary to keep track of progress. But there is a trap or two behind the corner. The development of jQuery Mobile stopped, for instance, and it no longer followed new versions of jQuery. We could do another migration or abandon the code of such an obsolete library.

Today we talk about extinct languages, and a programmer’s work is far from finished after an attractive application is complete. Its maintenance over time requires extra effort.

One of the questions a programmer may have is who to trust and follow so that his or her efforts do not go to waste too fast. Certain giant, global players could offer some answers, but even this is not risk-free. If I recall 3D graphics, I remember Microsoft’s Silverlight technology, which offered an array of beautiful 3D worlds and their animation. The trap this time was that it was a Microsoft product. Will others follow? Other operating systems exist beside Microsoft Windows. And so it happened that Silverlight did not catch on, and is now abandoned.

The dilemmas developers face nowadays are no different. The development of Android and iOS applications is very attractive due to the popularity of mobile devices. There are quite a few developer platforms available on the internet. But which ones do we think will last at least a little bit longer? And which will die out quickly? Is it React, or perhaps Flutter, which is based on a brand-new language called Dart? Why is this even necessary? We read forums and shape our opinions in the hope that our direction is the right one.

If we start a project from scratch, the first thing to do is to analyze the prevailing trends. Web applications and increasingly cloud computing are the most popular. For a while HTML5, CSS and AJAX/JSON have been the principal languages to use. JavaScript (or better yet JQuery) has superseded Flash. Applets have been long gone.

And program development has changed. Rapid incremental development with a sequential approach is gaining momentum. When designing one single project, programmers sometimes use various languages and have to know different APIs. Programs are increasingly complex.

One can observe the polarization of programming: On the one hand, we use high-level programming languages boosting programmers’ productivity, parallelization and ability to work in cloud environments. On the other hand, code effectiveness, execution speed and asymmetric calculations (also due to multi-core systems) can sometimes be important as well. Then there is “democratic” computing, which means that even a less knowledgeable (but motivated) user can develop a small segment, or at least tailor it to their needs. We must also not forget about “dangerous” computing. If something gets too complex a new framework can be designed, and it may upgrade the previous one. In this way, stacking looks a lot like a stack of dirty dishes, with ineffective code or security vulnerabilities somewhere within.

We may wonder what lies ahead. Let us not forget Moore’s famous law, which says that the number of transistors (i.e. the density of integrated circuits) doubles every two years. Then there are four laws postulated in 1997 by Nathan Myhrvold, formerly Chief Technology Officer at Microsoft, that discuss what is happening to programs. His laws of software spark off an interesting association with Newton’s laws. Let’s take a look.

1st Nathan’s law: Software is a gas.

It always expands to fit whatever container it is stored in (i.e. the computer’s capacity). Such expansion can be observed in numerous new versions of operating systems, such as Windows and Linux, and in the ever-growing length of browsers’ code.

2nd Nathan’s law: Software grows until it becomes limited by Moore’s law.

The growth of software is initially rapid, like a gas expanding, but is inevitably limited by the rate of increase in hardware speed. So every processor fails at one point. This usually happens just before new models see the light of day.

3rd Nathan’s law: Software growth makes Moore’s law possible.

People buy new hardware because the software requires it. Integrated circuits are faster than ever, but the price of computers remains more or less the same. We get better value for our money. This phenomenon goes on and on, because new programs emerge all the time.

4th Nathan’s law: Software is limited only by human ambition and expectation.

We never get enough. We get to work with new applications and new ideas of what is popular.

Programs and programming are always in a state of crisis. Whatever we achieve rarely meets the expectations of the users. The bar of expectation is constantly rising.

Programming is challenging even for experienced programmers. We constantly switch from one thought model to another, and translate various solutions to the code and back. In fact, programs are abstractions, and we often use concrete examples to understand them.

Also helpful are approaches such as object-oriented programming, and the use of high-level programming languages. Another common practice is the copy/paste approach, which is based on using code snippets.

It is a well-known fact that even children can understand something better with a visual example. We can easily understand what we feel, but words have to be analyzed first to get to their meaning. Short sentences are not a problem. Long texts, however, are more time-consuming and exhausting. And program code is no different. Some source programs are difficult to understand and sometimes it is easier to develop them from scratch.

It is true that programming languages are intended for people, not computers, and that we are still at the dawn of the history of programming. We took a leap from punched cards and tapes to interactive work behind computer screens. The next shift will have to take into account the fact that in a few decades we expect computers to have a human intelligence level. How will we program such computers? Will they learn the skills by themselves? Will they come up with new standards? The future will be exciting. And perhaps we should be worried.

# 3. From BESS to EVA, archival challenge

Primož Jakopin

## 1. Introduction

1. The initiative by the Slovenian Computer History Museum to organize a panel discussion on the legacy of informatization tools, of which software is the most important, has come at a very appropriate time. The year 1960 saw the first herald of the computer era in Slovenia, when a book by Professor France Križanič was published, *Elektronski aritmetični računalniki* (*Electronic arithmetic computers*), based on his experience with a [Ural-1](https://en.wikipedia.org/wiki/Ural_(computer)) computer at the Moscow State University (MGU). In 1962 the first computer in Slovenia was put to use, a [Zuse Z23](https://en.wikipedia.org/wiki/Z23_(computer)), and luckily quite a few of its users are still with us, to share their first-hand experiences.

In the paper the author’s post-mainframe software path is illustrated, involving text editors with language technology and corpus linguistic features, including a lexical web search engine. In the end the possibility of a Wikipedia-inspired solution to the presentation of his work as an online museum exhibit is discussed.

## 2. STRUCTRAN

1. At the end of the 1960s and even more in the early 1970s, it became clear that the programming languages of the day – in science and technology mostly [FORTRAN](https://en.wikipedia.org/wiki/Fortran), in business environment [COBOL](https://en.wikipedia.org/wiki/COBOL) – were not entirely suited for the tasks they were used for. Long and complex programming tasks produced immense volumes of source code, which was increasingly difficult to update and maintain, all because these two programming languages were still closer to machine code, and not abstract, structured enough to work with easily. In 1968 Professor Edsger Dijkstra published a letter titled “[Go To Statement Considered Harmful](https://en.wikipedia.org/wiki/Considered_harmful)”. Jumping from place to place, up and down, such code does indeed make one lose any overview of the intentions of the programmer very quickly, and profoundly, even more so after returning to the same code some time – days, weeks or months – later. This led to so-called structured programming, coding without Go To statements but with control structures, which have a beginning and an end, such as IF ELSE ENDIF, loops with FOR ENDFOR, REPEAT ENDREPEAT, WHILE ENDWHILE.

As writing a new compiler is a long and cumbersome task, Vladimir Batagelj from the Department of Mathematics at the Faculty of Natural Sciences and Technology, [University of Ljubljana](https://en.wikipedia.org/wiki/University_of_Ljubljana), suggested the development of STRUCTRAN, a program which would convert structured code into standard FORTRAN, to be compiled into an executable file. The idea was outlined in a paper titled “[STRUCTRAN](http://vladowiki.fmf.uni-lj.si/lib/exe/fetch.php?media=vlado:pub:conf:structran.pdf)”, authored by Batagelj and Egon Zakrajšek in 1975. The project was intended as a piece of cooperative work among students, during a programming course in 1975/76. As the result was not satisfactory, Zakrajšek produced the final code in 14 days at the end of the school year on a Control Data Cyber 72 mainframe computer, the main computer in the country that was also shared with others at the University of Ljubljana.

In the example below the right column was taken from the paper by Batagelj and Zakrajšek, with comment lines omitted for brevity and clarity. It is a very simple subroutine which sorts a list of integer numbers, contained in the array TAB, LENGTH long. As in FORTRAN all variables with names starting with I, J, K, L, M and N are by convention integer, unless specified otherwise, and only BOUND, CHANGE and TAB are declared. The FORTRAN version of the example (left column) contains no Go To statements but jumping is evident from the IF statements. Even to a non-programmer, the superiority of structured programming should be clearly evident.

Table 1: Bubble sort subroutine in FORTRAN and STRUCTRAN

STRUCTRAN was the first widely used general-purpose software that was produced in Slovenia. It made professional life much easier for many programmers, including the author of these lines.

## 3. From BESS to EVA

The m[icrocomputer revolution](https://en.wikipedia.org/wiki/History_of_personal_computers) started on the other side of the Atlantic in the 1970s, and reached, to the full extent, most of the “Old World” in the early 1980s. While the first personal computers appealed to many ordinary users as gaming machines, those who were familiar with [mainframe](https://en.wikipedia.org/wiki/Mainframe_computer)s saw them as an opportunity to have a computer on your desk, a real machine they could use whenever they wanted to, not a [terminal](https://en.wikipedia.org/wiki/Computer_terminal) with which one could access a large computer, shared with many other users. In 1981 or early 1982, Saša Albert, a friend from the [National and University Library](https://en.wikipedia.org/wiki/National_and_University_Library_of_Slovenia) and a technical all-rounder, brought a [Sinclair ZX81](https://en.wikipedia.org/wiki/ZX81) microcomputer with 1 KB of memory, a [cassette](https://en.wikipedia.org/wiki/Cassette_tape) recorder for external memory, and a [BASIC interpreter](https://en.wikipedia.org/wiki/BASIC_interpreter). It was made available to me for a few days, so that I – a known computer programmer at the time – could check if something serious could be done with it. A spare small TV set was used for the monitor. The BASIC code for computing the x, y and z coordinates of cave survey points was soon written on paper, it took one full page and about ten lines on a second one. The first page would fit into the memory, the next ten lines would not. Sašo received the machine back with the following verdict: It could be used for some serious purposes, but 2 KB of memory would be required.

The next machine was a 48 KB [Sinclair ZX Spectrum](https://en.wikipedia.org/wiki/ZX_Spectrum). Franci Ambrožič, a friend from the Faculty of Sport, had good connections in the UK and a machine was ordered in his name in July 1982. The demand was so high that the export price, which Ambrožič had to pay, was £228, as opposed to the domestic price of £175, not to mention the delivery time for foreign customers – almost half a year. In December 1982 the computer was smuggled from England to Yugoslavia, Franci enjoyed it for a few days and again, like Albert, decided to pass it to the author over the holidays around New Year, to see if something serious could be done with it. This time the answer was yes. As the machine had no text editor the author decided to make one. In about a week BESS (Basic Editor for the Sinclair Spectrum) was up and running. It was written entirely in BASIC and had 24 lines of 32 characters on screen. The interpretive nature of BASIC was most visible during text search, which took a minute per page. As an assembler was not available yet, the author wrote the text search subroutine directly in machine code, it was 50 bytes long and would search through 20 KB of text in 0.05 of a second. In 1983 and 1984 a new editor was devised, TESS (Text Editor for the Sinclair Spectrum), with 80% of the code written in Zilog Z80 assembly language, and in 1985 INES (INformation Editing System) was released, and this was commercially viable, mostly in Slovenia, although with a considerable impact in other parts of Yugoslavia. It was written almost entirely in assembly language, could work with up to 21 KB long files, had 24 lines of 64 characters (each in a 5 x 8 pixel matrix), white on black screen setting (or vice-versa), so that only 8% of TV screen was emitting harmful rays, a non-blinking cursor, no distracting permanent information on screen, smooth scrolling, and a new page of text was overwritten on the old one (not displayed on a new, blank screen). INES could also treat text lines as data records, with sorting, searching, mailing lists and several other data manipulation abilities. Graphics could be incorporated into text as an escape sequence string. There was a 68-page reference manual in Slovenian, and the editor was supported by additional utilities, such as for data entry and the sorting of longer files, up to 37 KB. English and German translations, of both the program and the manual, were also made and distributed to computer magazines in Germany and the UK. Early in 1986 INES was followed by EVE (Event Editor), also for the Sinclair Spectrum, and although this had more features it only survived long enough to get a short Quick Reference Guide. Further development was halted because the [Atari ST computer](https://en.wikipedia.org/wiki/Atari_ST) had arrived, with its beautiful, all powerful [Motorola 68000](https://en.wikipedia.org/wiki/Motorola_68000) processor with 16 32-bit registers.

EVE’s successor was STEVE (atari ST EVent Editor). Like its predecessors the name was an acronym, yet of a different gender. But as with [ship names](https://catamaranguru.com/why-are-boats-named-after-women/) it was an exception to the rule, and both subsequent editors, EVA and NEVA, had female names. As with INES, the author remembered well another famous statement by Professor Dijkstra – [microcomputers are not great](https://www.cs.utexas.edu/users/EWD/transcriptions/EWD06xx/EWD634.html) (IFIP 1977). To get the most out of the microprocessor-powered machines – increasing the speed of the software by at least a factor of three – the programmers were forced to start programming in machine language. STEVE was 75,000 lines of such code, handling the registers directly, without the elegance and ease of structured programming.

The Slovenian version of STEVE was ready in 1986, together with a 248-page manual (a second, 290-page manual was made in 1989, with Hinko Muren), followed by Croatian and German (translated by Günther Weber) versions in 1987, another German manual by Klaus Detlef Olof and Peter Wieser in 1988 (356 pages), and the ultimate [STEVE Reference Manual](https://jakopin.net/steve/STEve_Reference_Manual.php), with 608 pages in English, in 1989. The Atari ST machine with its megabytes of linear memory, a floppy and a hard disk, and an excellent monitor, opened new horizons, showing that ideas were possible far beyond mainframes. STEVE had quite a few novel features, one of them being that all its resources, from screen messages, keyboard layout, screen and printer fonts, to command abbreviations, were contained in a separate, STEVE-editable resource file, named STEVE.RSF. Other features included that the file size limited by available RAM, editor lines could contain text or black and white graphics, there were two graphic editors, database routines could also handle chained files where the only size limit was disk space, it had its own desktop publishing system and computer-aided instructions, as well as a [graphical user interface](https://en.wikipedia.org/wiki/Graphical_user_interface).

STEVE was distributed in Yugoslavia, Germany, Austria, Switzerland, the Benelux countries and Scandinavia.

The early 1990s brought the demise of the Atari brand, and so the author had to change the computer platform to the [PC](https://en.wikipedia.org/wiki/IBM_PC_compatible). PCs at that time (PC/AT) were based on the [Intel 80286](https://en.wikipedia.org/wiki/Intel_80286) chip, vastly inferior to the Motorola 68000, and it made no sense – to the author and others – to program it in machine language. A suitable operating system for the PC computers, with a graphical user interface, comparable to that of Atari ST or [Apple Macintosh computers](https://en.wikipedia.org/wiki/Macintosh), did not come until [1995](https://en.wikipedia.org/wiki/Windows_95).

To port STEVE with its large customer base to PC, just as it was, unchanged, would be a clever business idea. But people, especially developers, always strive to make something new, something better, more exciting. So the author, based on all his previous experience, decided to make a new, much enhanced editor, keeping all that was viable from STEVE, yet with added features which he could put to good use in his post-independent-software-developer professional life. It was named EVA, not an acronym this time, and it started as EVA for DOS in 1992 and was adapted for Windows in 1996. It is written in C language – as are the examples in [Charles Petzold’s](https://en.wikipedia.org/wiki/Charles_Petzold) book *Programming Windows 95*. In C++ the author could not come close enough to the machine to implement all the features he needed, especially in the user interface. EVA has a simpler graphic editor, no computer aided instruction, limited DTP, but a true 16-bit character set with 2500 implemented characters, lines which can contain 8-bit text, 16-bit text, compressed black and white graphics and single-line bit image black and white graphics, expanded database routines with XML data format support, Optical Character Recognition (OCR) as standard (STEVE had it on demand), Part of Speech Tagging (POS) and several other language technology features such as the statistical modelling of language. One of the main attractions of STEVE, retained in EVA, was the large monospaced white screen characters on a dark background, which helped the author – who uses this latter system for most of his work, even for the construction and updating of webpages – to preserve his vision. This is quite unlike most of his colleagues who have stared at screens throughout their careers, who now typically wear glasses and have their daily dose of computer work limited, on a doctor’s recommendation, to an hour per day or less.

As of June 2022 the EVA source code is 158,211 lines long and is composed of 2,684 routines. While STEVE has 414 implemented command codes, EVA has 1,385.

## 4. NEVA

In 1998, while the author was completing his doctoral thesis at the Faculty of Electrical Engineering, University of Ljubljana, titled “[Upper Bound of Entropy in Slovenian Literary Texts](https://www.jakopin.net/primoz/disertacija/abstract.php)” his main thesis supervisor, Professor Nikola Pavešić, alerted him to the fact that the thesis length must not exceed 200 pages. However, the thesis dealt with a text corpus of some three million words and several interesting extracts deserved space in the appendices, and altogether this would exceed the page limit several times. The author was thus advised to publish the appendices on the Internet. Since a searching capability would be a welcome addition to corpus presentation, the author learned the [Common Gateway Interface](https://en.wikipedia.org/wiki/Common_Gateway_Interface) with its scripting language and wrote a Windows-server based search engine, also in C. This is a subset of EVA, mainly the relevant database routines with an enhanced mailing list process for the production of webpages with search results. He called it NEVA, for Networked EVA. In June 2022 NEVA had 491 routines in 25,306 lines of code.

After receiving his doctorate in June 1999 the author, employed by the [Faculty of Arts](https://www.ff.uni-lj.si/en), [University of Ljubljana](https://en.wikipedia.org/wiki/University_of_Ljubljana), decided to increase the core corpus installed on the faculty server – consisting of around three million words of fiction in Slovenian – with additional material that was mostly taken from the main Slovenian daily newspaper, [*Delo*](https://en.wikipedia.org/wiki/Delo_(newspaper)). *Delo* had an email service for the visually impaired people (*Delo za slepe*), who received a text-only copy every morning. With permission to use it for research and educational purposes he managed to increase the corpus size, now named CORTES (CORpus of TExts in Slovenian, established 1999), to some 28 million words by early 2000. This attracted the attention of the [Fran Ramovš Institute of the Slovenian Language ZRC SAZU](https://isjfr.zrc-sazu.si/en/predstavitev) (ISJ), where the author also worked part-time. As the Faculty of Arts could not provide an institutional framework for CORTES, and as both institutions are connected in many ways, the author was easily persuaded to move the corpus to a server at the [Research Centre of the Slovenian Academy of Sciences and Arts](https://zrc-sazu.si/en) (ZRC SAZU), of which ISJ is a member. This happened in May 2000, when the corpus was renamed Nova beseda (New Word) and the entire web site to [BOS – Bank Of Slovenian](http://bos.zrc-sazu.si/index_en.html).

The author then moved to the ISJ to lead the newly established Laboratory for the Corpus of Slovenian Language, and continued to work part-time at the Faculty of Arts, as a teacher in the field of language technology.

*Figure 1: English version of the web site* [*http://bos.zrc-sazu.si/index\_en.html*](http://bos.zrc-sazu.si/index_en.html)

Over time many of lexical resources of the ISJ were made available through the BOS web site, with some, as shown above, also presented in English (in 2015 the new [Fran](https://fran.si/) web site was established as the principal dictionary site of the ISJ in Slovenian). The most prominent and the most used of these resources were the Dictionary of Standard Slovenian Language (SSKJ, 1970–1991, 93,500 entries), List of Slovenian Words (BSJ, 356,000 headwords, 2006) and the Nova beseda text corpus (318 million words, 2010). While the SSKJ is the representative monolingual dictionary of Slovenian, the BSJ – compiled from SSKJ, Dictionary of Lesser Used Slovenian Words (178,457 entries), Nova beseda text corpus and the index of the Slovenian web search engine [NAJDI.SI](http://www.najdi.si/) shows all the vivacity and flexibility of Slovenian in forming new words. Its size also makes sense if compared to the [number of words in English](https://www.merriam-webster.com/help/faq-how-many-english-words). The two examples below illustrate the use of NEVA in examining two components of the name (Slovenian) Computer History Museum – *Računalniški muzej*.

Table 2: *Muzea* or *muzej* search results from the list of Slovenian words

The results show 32 words, nine of which also appear in the SSKJ and are highlighted. There are 20 nouns, 10 adjectives and two verbs. It is of interest to note special kinds of museums are also included in the results, *ekomuzej* (ecomuseum) and *fotomuzej* (photomuseum).

Table 3: *Računal* search results from the list of Slovenian words

The results in this case show 82 words, and just six of these also appear in the SSKJ, as this dictionary was only compiled until 1991, before the Internet started to affect the language. Only one entry, *obračunalec* (settler (finance)) is suspicious, and while this could have non-computer related meaning a detailed inspection reveals that it comes from word games.

## 5. Archival challenge

Every exhibit in a museum, especially if it is made by human hand and heart, evokes many feelings, impressions and thoughts, be it the [first Venus](https://commons.wikimedia.org/wiki/File:Venus_of_Tan-Tan.jpg), a [55,000 year old flute](https://en.wikipedia.org/wiki/Divje_Babe_flute) or a more modern [earring](https://cdny.de/p/t/5/088/4469638.jpg) made of precious stones. How did the maker craft it? What brought them the idea? Who was the person that used it? All these considerations are even more vivid for computer-related exhibits, from the pieces of hardware, harbingers of the digital age, to software, intended to make the best use of these old, and not so old, machines.

The author’s experience with the evolution of basically the same product, a text editor which over time took under its wings many tools that helped solve a myriad of tasks which he faced in his professional life and extracurricular activities, will towards the end of this year be 40 years long. The first incarnations of this project, from BESS to INES and to a lesser extent STEVE, were made to make the most out of the machines’ limited microprocessors. Later on the hardware was fast enough to allow structured programming in a high-level language, and so the program was easier to maintain and to expand. Many user needs and suggestions were taken into account, and the dynamics of change were very fast in the first ten years, with ups and downs in the second decade, more moderate developments in the third, and occasional ones in the fourth. The code for the early versions – BESS, TESS and INES – is more or less lost. During the author’s frequent changes of residence many pieces of older media disappeared, Sinclair ZX Microdrive cartridges were not particularly reliable and also very small and easy to lose. STEVE and EVA for DOS were kept on floppy disks which are still in the author’s possession, but their condition is not known. Atari ST formatted floppies are not readable on PC machines, so assistance from the computer museum would be welcome to make a catalog of that collection. The STEVE reference manual source file, in STEVE DTP format which is also readable by EVA, was luckily still accessible, and so was converted to HTML and [published on the web](https://jakopin.net/steve/STEve_Reference_Manual.php) during the preparations for the June 2022 SRM panel discussion. The source code for EVA and NEVA, developed and maintained within [Microsoft Visual C](https://visualstudio.microsoft.com/vs/community/) and [Bloodshed’s Dev-C++](https://www.bloodshed.net/) environments, is kept by the author.

How to best preserve the legacy of this project? How to make a proper presentation of it in the Slovenian Computer History Museum? And how to make it an online exhibit as well? One possibility, to begin with, is its inclusion in [the Internet Archive Software Collection](https://archive.org/details/software), yet a much better option would be to keep it functional. Unlike most of our digital history, EVA and NEVA are both still operational and used.

The role of a librarian has changed from a person in pre-digital times who could tell an interested user where, and in which book, to look for certain information, to someone who can advise a user where to look on the Internet for what he or she needs. So the role of a museum curator and museum guide should change, too. With better, in-depth knowledge about a software exhibit, it can be presented to the museum visitors in a vivid, interactive fashion, incomparably better than any description presented online. Even more so if the software is still operational, like EVA and NEVA. Such a curator would also be a valuable source of information to people who have to tackle a problem that was presumably already solved, yet the solution is not generally known or is poorly or not at all documented on the Internet.

To facilitate the making of such a museum presentation it should be prepared by someone well acquainted with the subject, ideally the author, and by the museum curator who will deliver it. The process would certainly take time, but it would be the time well spent.

The presentation of software as an online exhibit is a connected story, but again a different one. [The Internet Archive Software Collection](https://archive.org/details/software), though valuable, does not present software in a unified, curated fashion, and the sources are also not there (the [SourceForge](https://sourceforge.net/) open-source platform, where the number of entries is smaller by two orders of magnitude, is not an archive collection). To make EVA and NEVA an online museum exhibit would require a medium very much like [Wikisource](https://en.wikisource.org/wiki/Main_Page), but adapted to the presentation of software. Unlike books, where the reader can enjoy the artistic reflection of human life in fiction by reading it page by page, software can be enjoyed by watching its performance with a set of data, from a tiny seed in a random number generator to a text processor which can handle the [four billion words](https://en.wikipedia.org/wiki/Wikipedia:Size_of_Wikipedia) of Wikipedia articles (April 2022).

# 4. Towards the sustained use of software for long-term access to digital heritage

Nataša Milić-Frayling, CEO

In this paper we reflect on the transformational impact of adopting digital media for encoding and storing information and the importance of software for processing digital data and transferring knowledge. Unfortunately, the rapid rate of innovation causes rapid software obsolescence, and affects our ability to use digital content. This is particularly challenging for highly interactive and dynamic digital artifacts that use software computation to derive and convey insights from data and study phenomena. In fact, the faster we innovate, the faster software is replaced with new products and becomes unsupported and obsolete. That makes it difficult or impossible to reproduce past data analyses, play old games, and use interactive content such as digital art. Fortunately, advances in computing also provide us with the means to counteract the effects of software obsolescence. At Intact Digital we created a Software Library platform that uses virtualized computing environments to provide stable and protected installations of legacy software and enable the secure and easy use of digital content from decades ago. This is one effective approach to enabling digital continuity that is essential for transferring knowledge to future generations and building on our digital heritage.

## Digital media

With the onset of digital revolution in the mid 20th century, we have experienced the fastest growth in the production of content and information and unprecedented speeds in transferring and exchanging data and knowledge through computing technologies. Now, 70 years later, we cannot imagine the world without the Internet services and mobile devices. They shape every aspect of our everyday life.

Scientific and engineering fields have been equally transformed. Digital technologies embedded in instruments and tools are enabling data analyses and knowledge discovery beyond imaginable. With increased capacity of portable storage devices and a shift to cloud computing and cloud storage, we are now commonly dealing with terabytes of personal content and petabytes of scientific and commercial data. However, digital media and digital computing depends on highly sophisticated technologies that require continuous updates to stay functional and usable. Thus, it is of utmost importance to consider technological, economical and educational factors that affect the continuity of the digital media use and take action to ensure that our digital heritage lives and reaches future generations.

## Digital continuity and the importance of software

In order to ensure the long-term use of digital content, we need to understand the essential aspects of digital media and the ways it ages and deteriorates.

Every digital content is created, collected, stored and consumed by using compatible software. Thus, digital media is fundamentally computational since the software features shape the encoded data and information. In order to reuse the products of our work, we keep digital documents, images, videos, and databases as files. Each such file is written and read by a particular piece of software or a range of compatible software that can process the files. Without compatible software, digitally encoded content cannot be interpreted, presented and experienced. However, those who focus on data storage alone overlook the importance of software.

*Digital files without software are like musical scores without instruments or musicians. We will never be able to play and experience stored digital content without working software and without the skills to use it.*

While some software is created for specific hardware devices, most can be installed on a variety of hardware or in virtual machines. As long as the hardware runs an operating system that is compatible with the software, one can install and use it. While hardware obsolescence is also an important problem, for the sake of this discussion we will focus on the issues that arise from the obsolescence of the operating systems and the software itself.

Software can be an application, like a word processor used for creating new documents, or a game that we enjoy playing through a carefully designed interactive experience. Each piece of software depends on many other technical components, from software that enables the use of mouse, keyboard and screen, to the security patches that make the operating system and the whole computer safe. Thus, for any software application to remain functional and usable, it must be constantly monitored and updated if other supporting components change. The most frequent updates are due to security threats, and once it becomes unfeasible for the software producer to keep customers safe, they have to pull the product from the market.

For example, quite recently, in December 2020, Adobe discontinued support for Adobe Flash and, from 12 January 2021 blocked Flash content from running in Flash Player[[4]](#footnote-3). It instructed all users to uninstall Flash Player in order to protect themselves from security risks. Unfortunately, this had a detrimental effect on web publishers and artists who had been producing digital art using Flash and enabling online audiences to use dynamic content and Flash animation through a browser for the past two decades. Software is thus key to the use of digital content, and the lack of functional software has a direct impact on what digital content we can continue to use and what knowledge we can transfer to future generations.

## Software history and rate of obsolescence

There are many different types of software. A good place to see a variety of software is the Internet Archives Software Collection[[5]](#footnote-4). The collection includes over 862,000 software packages, from operating systems, media production software and statistical packages to games and specialized software using 3D visualizations, maps and animations. The Computer History Museum in Ljubljana[[6]](#footnote-5) and similar organizations around the world[[7]](#footnote-6), reconstruct, preserve and display old software programs and let us experience the exciting journey through the development of computing technologies. However, outside such museums, much software is not in use anymore.

*Software obsolescence is a universal phenomenon that affects all software. It is a natural consequence of innovation: as we create new software versions, old ones become unsupported, unsafe, and unusable.*

In essence, the faster we innovate, the faster software becomes obsolete and the more of our digital assets are in danger of becoming inaccessible and unusable without compatible software. The impact of software obsolescence is particularly detrimental for digital content that has long-term value, such as knowledge resources and digital cultural heritage. For example, scientific experiments cannot be reproduced reliably without the original software. Digital art cannot appreciate in value if it cannot be shown and interacted with. It is thus critical to enable the long-term use of software. Fortunately, there are ways to enable the use of past data, reconstruct past studies and present digital art from decades ago.

## Software library for legacy software

In 2016, Intact Digital[[8]](#footnote-7) began a concerted effort to create technically effective and economically sustainable services for long-term hosting and maintenance of the software needed to enable digital continuity.

Since software comprises a source code and installable executables, one can adopt different strategies to ensure that software continues to run. Source code is normally available for open-source software and, in principle, if the developer community retains the knowledge and skills needed to continue software development and can ensure that old data can be used with new versions of the software, there is no danger from software obsolescence.

However, a large section of the software industry and our digital economy is based on proprietary software that is highly customized and for which the software source code is not publicly available. The new versions of the software may not be backward compatible, and modifying and re-developing software would be costly or unfeasible due to the lack of documentation and know-how. More importantly, if the software producer goes out of business, the software becomes completely unavailable and users are left without upgrades and, eventually, without the ability to use their data.

*The more successful the software product, more damage is caused by its obsolescence.*

In some instances, there is modern software that can serve as a substitute and make use of data files from obsolete software products. Otherwise, our best approach is to create a computing environment in which old software can still run without security threats. That can be achieved by installing software in virtual machines.

In order to establish a principled way of dealing with legacy software dependence, Intact Digital devised an Executable Archive framework[[9]](#footnote-8) that complements the traditional archives with a Software Library platform and services for ensuring long-term hosting and maintenance of legacy software.

Long-term software care is achieved by adopting a systematic approach to managing the software files and documentation needed to create software installations. That includes quality assurance practices that are applied during the software installation and ongoing maintenance of virtual computing environments.

Nowadays, software virtualization is broadly used in data centers and public clouds for flexible management of computing resources that are needed by organizations and individuals. Many end users also use virtual machines on their home computers. For example, Apple Mac users can also use a virtualized Windows environment on their machine.

One can think of a virtual machine as a desktop computer but without its own hardware. Inside the virtual machine one can install old operating systems and old applications. Such a virtual machine can be hosted on a modern computer and all the instructions from the installed software applications are translated into commands on the host machine. The users can thus use the software in the same way as in the past, as it appears as an application in a virtual desktop that is familiar to them.

*Figure 1. Executive Archive Framework complements the traditional electronic archive practices with procedures and technical components that ensure the long-term use of software. The Software Library platform enables software hosting, software storage and the remote use of software to process archived data.*

## Reconstruction of data analyses

In highly regulated sectors, such as pharmaceuticals and life-sciences, digital data must be retained for decades. Complex scientific protocols typically involve the use of sophisticated instruments to collect data and highly specialized software to interpret and analyze it. Files containing raw instrument data are stored in electronic archives and regularly assessed for data integrity, using, for example, checksum methods. The corresponding software must remain functional for reconstruction of past studies and for reproducibility of data analyses. This is a challenge, because both the related instruments and the software are decommissioned after a while and removed from operational use.

The INTACT Software Library is therefore used to create virtual installations of the software needed to reconstruct past studies. The key requirement is to ensure that the software installation in the virtualized environments leads to the same results as the software originally run in the labs where the studies were conducted. Furthermore, access to the virtualized software needs to be controlled to adhere to the licensing agreement with the software vendor and to provide a detailed audit trail of software use that is required by internal policies and regulations.

From the user’s perspective, the virtual machines in the Software Library are conveniently accessed through Virtual Desktops from any modern browser (*Figure 2*). This is achieved by the careful isolation of the virtual machines that host old operating systems and, therefore, should not be exposed to the Internet. A copy of the data from the electronic archive is safely transferred into the Software Library using a Transfer Desktop.

*Figure 2. The Software Library can be used from any modern browser. Through the specially configured Transfer Desktop, the user can transfer a copy of the data into the Software Library and then process it with a specific piece of software. Each virtual machine with the software appears in the Browser Tab as a separate desktop.*

Once in the Software Library environment, the data is used within the virtual machine that hosts the software needed to process the data. Each virtual machine appears as a separate tab in the browser. Figure 3 shows an example of a virtual machine hosting Windows XP operating system and Analyst 1.4.2 software produced by Sciex. The user can use the software and the data in the same way as originally done in the lab.

*Figure 3. Analyst v.1.4.2. software by Sciex is installed in a virtual machine running Windows XP and used through a virtual desktop that can be accessed through any compatible browser. The virtualized Windows XP desktop appears in a separate browser tab and cannot be accessed by services from the Internet.*

## Reconstruction of digital art

Over the past few decades, artists have used Internet technologies to create Internet art and reach broad online audiences. An important enabler of such artists’ creativity was Adobe Flash, formerly commercialized by Macromedia Flash and FutureSplash. Flash enabled flexible use of text, vector graphics, video and audio for the production of animations, games, rich web and desktop applications, and browser embedded video players. End users could conveniently view Flash content via the Flash Player within web browsers.

Furthermore, with the standardization and adoption of Virtual Reality Modeling Language (VRML)[[10]](#footnote-9), authors could also specify platform-independent 3D “worlds”, including objects with rich structures, textures and interaction models. The Cortona3D[[11]](#footnote-10) viewer for VMRL also enabled non-standard support for combining VRML with Flash textures, providing additional creative opportunities.

Unfortunately, with the recent obsolescence of Adobe Flash and Flash Player, all digital art that uses Flash is affected and requires a concerted reconstruction effort to remain accessible and usable.

For example, Intact Digital has worked with a contemporary artist Michael Takeo Magruder[[12]](#footnote-11) on the reconstruction of *World[s]* (2006(v1.0), 2009(v1.1))[[13]](#footnote-12) [12] that relies on VRML and Flash plug-ins to enable the textured 3D rendering of audio-visual art elements. The artist maintains a website (<http://www>.takeo.org) with detailed descriptions of art pieces, including documentation, videos, and still images, and manages a repository of digital media files and selected versions of software that were used to create and publish the artworks.

Through a collaborative effort, the artist and the Intact Digital technical staff have created an installation of the art within the Software Library environment that can be used by audiences online (*Figures 4 & 5*).

*Figure 4. Software Library hosts an installation of the World[s] artwork by Michael Takeo Magruder that includes Flash and Cortona3D plug-ins for the browsers, in a secure Software Library environment.*

*Figure 5. Cortona3D v.7.0 with Internet Explorer 11.1790.17763.0 and Macromedia Flash ActiveX plug-in 8r42 are installed within an isolated VM with a GPU, to provide reliable and secure access to World[s] from any modern browser.*

## Long-term care of software

While the technical aspects of software obsolescence present obvious challenges, enabling digital continuity requires a holistic approach considering a broader range of concerns. For long-term software maintenance within the Software Library, we have developed quality assurance practices that cover technological, legal, operational and human factors (*Figure 6*).

*Figure 6. Long-term software care requires consideration of multiple factors and quality assurance procedures for identifying and mitigating potential risks.*

Among the technology factors we cover the security and integrity of the long-term storage of software files and documentation, computing environments needed to install and run the software, and methods for secure access and use of installed software.

The legal aspects, for example, involve the licensing of all the technical components involved in the installation and use of software. Operational activities involve continual maintenance of the computing environment, and the protection and possible re-installation of software in order to prolong its use. Human factors are often overlooked, yet they are absolutely essential. It is thus important to provide training for using legacy software installations, to retain skills and ensure that younger generations can operate historical software and historical data reliably and efficiently. The secure use of software installations is particularly important, since old software is fundamentally non-secure if exposed to the contemporary ecosystem. Without upgrades it may not run on the latest operating systems and may be vulnerable to new types of cyberattacks.

Overall, with a systematic and principles approach offered by the Executable Archive framework and the convenient use of software hosted in the Software Library, we can effectively address a wide range of digital obsolescence scenarios. Building on these foundations, we can make considerable progress towards digital continuity and keep on innovating, knowing that our digital heritage is safe and accessible for as long as we need it.

# 5. Partner – Forever young

Boštjan Špetič

The Computer History Museum in Ljubljana, a private museum established in 2004, collects hardware and software of special significance for Slovene society. According to the Museum’s collection policy,[[14]](#footnote-13) objects made in Slovenia or by Slovenian inventors hold the highest priority within the collection criteria. The Computer History Museum in Ljubljana thus houses what is probably the largest collection of Slovenia-made computers. Even more than that the conservation and restoration departments, alongside the museum experimental laboratory are actively working their way towards making all of them operational again. This is a complex endeavor requiring discrete protocols and the involvement of various subject experts,[[15]](#footnote-14) and it includes digital archaeology as well as reconstruction efforts. This museum process thus offers a unique insight into the challenges and opportunities of software preservation.

During a routine backup of the hard drive of one such Slovene computer, we found a treasure trove of original software. Software is generally the hardest part of computer history to find and to preserve, because the majority of computers such museums work with do not have hard drives, and because floppy disks and other storage media were not preserved well technically or systematically by generations before us.

In this article, the author will explain the significance of what was found on this hard drive, and outline the complexities involved in the archiving of heritage software.

## The Iskra Delta Partner Computer

In the year after the famous merger of Iskra and Delta[[16]](#footnote-15), six experts of the joint company took less than two months to develop a prototype of a desktop microcomputer they named Partner. It was intended mostly for new and smaller organizations, but was also marketed as a system for research activities, schools and even private individuals.

*Figure 1: Partner computer from the collection of Computer History Museum, Ljubljana. (photo: Computer History Museum in Ljubljana archive)*

The Delta Partner was a desktop computer conceived as a practical business or development system, mostly used for accounting, bookkeeping and statistical processing. The computer could be integrated into networks with larger computers, where it functioned as a terminal as well as an independent system. After 1983 several models with identical basic characteristics and optional extensions for each model were being produced. The system was built on top of the then already somewhat out-of-date 8-bit microprocessor Zilog Z80A with a 4MHz measure, usually containing 128KB RAM memory and 8KB ROM memory.[[17]](#footnote-16)

The Partner computer was sold in a bundle with dedicated software, initially consisting of only four user programs and an operating system, CP/M 3. The four MIPOS user programs served individual or integrated business data processing needs. They enabled bookkeeping through a main book, accounts for both customers and suppliers, but also warehouse logistics management and invoicing. The software tools gradually enabled other office work, such as text document processing, and more tools were also released to support scientific calculations in the fields of energy management, machine engineering, and construction, as well as tools for assisting processing technology in the food-processing and pharmaceutical industries. These were offered in addition to software for interactive programming, classical compilers for Cobol, Basic, Pascal and C, and a macro-assembler for symbolic machine language. The first versions of the system only displayed text, but newer ones were enhanced with a graphics card with a resolution of 1024×512 points. Data could thus be shown in graphic format with the help of the BGRAF, IDRIS and DIAS program packages. Among the first three versions of the Partner computer, the WF/G model was available with a floppy disc drive and a 10Mb Winchester disc, while models 1F/G and 2F/G were available with one or two floppy disc drives. All Partner computers had a serial port intended for connecting with a printer and mouse, and it was also possible to install additional ports. Usually the system included a TRS835 printer, and other expansions included a network port and video-out port.[[18]](#footnote-17)

Production was first established in the city of Kranj, and later moved to Ptuj. Production for western markets was gradually set up in Delta’s new Austrian factory, and a special model of Partner with a different software programs package was produced by Novkabel in Novi sad. In August 1983, the first ten systems out of the regular production line were ready for testing and by the end of the year 170 systems had already been assembled in Kranj. A total of 200 units were sent to the Austrian St. Jakob plant, where at least 100 Partners were to be assembled by the end of year for export. Out of all the Slovene computers, this was by far the most widely adopted model, the total production number being several thousand Partner computers, while the planned production was up to 10,000 systems. According to the 1988 registry of computer equipment,[[19]](#footnote-18) approximately 450 Partners were still being used in Slovenia. Today, we know of 12 Partner computers, and expect at most two or three times more to be stored by private collectors or just waiting in garages to be found.

A handful of schools in Slovenia had a classroom equipped with Partner computers, the first amongst them being the Technical high school in Kranj. The computer course there in the late 1980s was run by the future head of the machine learning research department at the Jožef Stefan Institute (Institut “Jožef Stefan” – IJS), Marko Grobelnik.[[20]](#footnote-19)

## Emulating the Partner environment

The demise of Iskra Delta, the dissolution of Yugoslavia, transition to capitalism and the advance of IBM PC architecture made Partner obsolete, and soon also practically forgotten.

In 2018, a student of computer science in Slovenia reverse-engineered the internals of the computer and wrote an emulator which enabled the original software to run again for the first time in 25 years.[[21]](#footnote-20) This student, Matej Horvat, is also part of the Computer History Museum team, with a special expert focus on Partner computers.

Since then a small community of “Part-Time Nerds”[[22]](#footnote-21) has taken on the challenge of writing new software for Partner computers using this emulator. Horvat’s emulator also runs in an online environment, so we were able to put it on the museum webpage for anyone to try.[[23]](#footnote-22)

## Partner software is a rarity and a historical curiosity

The emulator was written using a Partner computer that the University of Ljubljana Faculty of Computer and Information Science had carefully preserved. Their and ours are the only two Partner computers we know of that still have a functional hard drive, making them the only two sources of original software to date.

The complete list of currently preserved software contains some Microsoft programming languages, some original Iskra Delta productivity software, a partially localized WordStar text editor and a number of user-written applications.

One of the most interesting examples of original development the Computer History Museum found in its media archaeology endeavors is a version of a game Snake,[[24]](#footnote-23) called *Glista*.[[25]](#footnote-24) It came with impressively extensive documentation. Here is a translation of its splash screen which clearly describes the intentions of the author, thus encapsulating a certain human effort of the time:

“This game was created as a means to manage boredom, which usually happens during school classes (and during breaks). It doesn’t make any sounds other than the clicking of the floppy drive. If you disable ‘click’ it will be completely silent (verified). The game doesn’t require much focus, so you can still follow the professor with the left side of the brain. If at first you don’t manage to get many points and are losing hope, know that only practice will help you get better.”

*Figure 2 Screenshot of the splash screen of the game Glista. (photo: Computer History Museum in Ljubljana archive)*

*Figure 3 Screenshot of the main screen of the game Glista. (photo: Computer History Museum in Ljubljana archive)*

So yes, it’s “just” a game, and a copy of a well-known game, at that, but the attention to detail and the specificity of the user scenario encouraged us to look deeper.

## Archiving software as a core museum procedure

At this point, please allow the author a short deviation to explain how we understand software archiving at the Computer History Museum. As we presented at the annual NetPreserve Conference,[[26]](#footnote-25) our job as museum professionals is to preserve as much as possible for future generations of researchers. In one dimension we are interested in as much “information” being preserved as possible. In another we look at the aura of the physical objects. But our job is also to maintain the knowledge necessary for technical understanding of the objects, and to communicate with the public. We do all of this when we manage to bring a piece of code all the way to live reconstruction.

For some examples of the various stages involved in this, here is a mock-up, a design document, for text-based software that would probably run on a Partner computer:

*Figure 4 Design wireframe example of text-based business application.*

At another extreme, here is a complete production rack of Slovenia’s largest BBS. The Computer History Museum team claimed it directly from the room in which it was last running live and serving users, and we are currently working on disk recovery. This is an example where software was off-the-shelf, but the communication and data left on the server by the users is invaluable.

*Figure 5 Picture of production stack of largest Slovenian BBS. (photo: Computer History Museum in Ljubljana archive)*

We follow the stages of software preservation as follows:

1. Evidence of its existence. Mentions in interviews or contemporary articles.
2. Documented inputs & outputs. Screenshots or manuals.
3. Binary files. Often salvaged from disks and tapes.
4. Binary + data. Taken from some kind of production environment.
5. Source code.
6. Bonus: original hardware.
7. Extra bonus: the actual original machine that ran this specific software.

The ultimate goal after any stage is live reconstruction. Every higher level of preservation makes the reconstruction easier, more possible and more complete. In practice we feel extremely lucky every time we are able to get to at least the (3) binary level, and thus beyond just evidence of existence.

The programs we found on the only two Partner computers known to have a hard drive were all in binary form, allowing us to run them on the original hardware or in an emulator. They give us a glimpse into how the software was used, but not into the authors’ way of thinking.

*Glista* is an exception, because it had so much attention to detail that it included the full documentation, the splash screen with motivation that we shared above, and on top of all this the software was signed.

## The inseparable entities of the user and the machine

Our research department likes to interview the users and authors whenever possible, so we have a record of first-hand experiences. However, the opportunity to do this is extremely rare.

We sought out the author of *Glista*, who is now a high-ranking bureaucrat at the Ministry of Public Administration. He visited the museum, and we could show him a working Partner computer, the actual computer he last used 35 years ago at that Kranj high school, under the supervision of professor Grobelnik.

Mr. Gabrijel is a very serious person, and the attention to detail in *Glista* shows that even back in his high-school years he was taking fun very seriously, as well. When reunited with his software he was sincerely touched by the experience and thanked us deeply afterwards.

## Conclusion

This demonstrates the human aspect of software preservation, and the ability of software to provoke emotions. For the software creator the running code is a part of their self, encapsulated in digital form, as much as an artwork is for an artist.

Software heritage is the heritage of human ingenuity. It encapsulates the spirit of *homo sapiens* as we have known it in recent centuries, and it testifies to the rise of the possible future of *homo digitalicus*.

## Postscript

Partner computer internals don’t allow for the representation of dates after 1999, so Horvat decided to fix that in the emulator by resetting any date that might be in the 21st century back to the base from 1990.

The programs in the emulator are thus running in a perpetual 1990s – a golden era for computing that Partner computers never actually experienced, because PCs took over the market, and because Iskra Delta filed for bankruptcy in 1990, starting the first liquidation process in independent Slovenia, one that only ended in 2020. Unfortunately, the liquidation manager destroyed all technical documentation early in this process.

*Glista*, however, will now live forever inside our Partner emulator, partying like it’s still the 1990s.

# 6. Legal challenges of archiving software

Matija Šuklje

In this article we will briefly deal with the main legal challenges of archiving software in a museum. Since copyright is the main “intellectual property” right responsible for protecting and limiting the use of software, we will primarily analyze the situation through that prism, but will mention other rights where relevant.

We will try to be as generic as possible, but since both the Computer History Museum and this article’s author are from Slovenia, this is the main jurisdiction we will concentrate on.

## “Intellectual property” rights

With the term intellectual property[[27]](#footnote-26)rights (IPR) we understand exclusive rights to intangible assets.

Note that the original creator – such as the author in copyright, or inventor in patents – may not always be the actual holder of these exclusive rights. They could have transferred their rights to someone else via a contract, or due to having made the subject of the rights as part of their work for their employer.

Unless the law or the rights holder give you specific rights to the related creation, the rights lie solely with their holder.

We will tackle each IPR that is relevant to archiving software in the sub-chapters below, in descending order of importance and risk.

### Copyright

Copyright (= authors’ rights) is the main IPR that covers software.

It covers expressions (not just “mere ideas”) and automatically starts the moment an original work of art is fixed in a medium, and is protected globally.

The originality test is important here, but also somewhat strange. On the one hand it provides the option for two identical pieces of work to exist and both carry copyright protection, as long as both authors have made their work completely independently and the similarities are accidental. On the other hand, the originality test also prevents trivial things like facts, configuration files and so on from being copyrightable.

The term of copyright in most jurisdictions is for the duration of the author’s life plus 70 years after that; or in the case where the copyright holder is a legal entity, 70 years from fixation. After that term, the work falls into the public domain and anyone can do anything with it.

### Patents

Patents, at least in the EU, “do not exist on software *per se*”. In practice though there are several ways to circumvent this, as it is in the business interests of both some (not all) IT companies and patent offices. As such we need to also take patents into account.

In fact, in the early days of computer software there was a lively debate as to whether software should be protected by patents or copyright. Copyright won, but in the end software is the only creative work that can at the same time be protected by both copyright and patents.

A patent covers an invention, which has to be novel, have an inventive step, and be industrially applicable – mere abstracts and mathematical algorithms are not enough. In practice, it depends on how diligent the jurisdiction and patent office are at running these tests. Which means many (software) patents in the wild[[28]](#footnote-27)could be invalid, but someone has to bring a suit to invalidate them.

Patents have to be registered in each country where the inventor filing it seeks protection, and typically last 20 years from filing. After that they expire, and the invention falls into the public domain.

### Trademarks & service marks

Trademarks protect the origins/branding of goods and service, marks[[29]](#footnote-28)those of services. In order to play their role – and therefore to be valid[[30]](#footnote-29) – they need to be distinctive and need to be enforced.

Typically you would register a trademark (®) for 10 years, and can renew this as often as you want. A registration is limited to the country and the classes of goods/services you filled it for.

The concept of “notorious brands” also exists, which means that if a brand is extremely well known globally its owner can prevent others from using the trademark, even for classes that the owner has not registered for (e.g. Nutella-branded keyboards).

Furthermore, the law gives certain protection also to unregistered trademarks (™) as long as the “owner” enforces that trademark.

With software you would typically run into trademarks in company, project, product and services names and logos.

### Data in/with software

Often a piece of software is useful and interesting only because of the data it manipulates. When we deal with such data we need to take into account the distinction outlined below.

#### Data

Facts – and as such pure data – are not protected by copyright. But a piece of data can be protected by copyright (e.g. literary text, images, audio, video etc.), trademarks (brand names, logos) and so on.

If the data you are using is personal data, a further consideration is that you need to follow the relevant personal data protection laws, such as the GDPR.

#### Database

Often data will be collected and ordered in a database. Databases *as a whole* are often protected by database rights, such as the Database Directive in the EU[[31]](#footnote-30), while (some narrow) copyright may still apply to an original arrangement, selection and presentation of data.

According to the Database Directive, database rights last for 15 years and are automatic. Each time a database is substantially modified, however, a new set of rights are created for that database. An owner has the right to object to the copying of substantial parts of their database, even if data is extracted and reconstructed piecemeal.

### Design & UI/UX in software

Often a computer program will have a (graphical) user interface. The following rights can apply to the general “look & feel” of a piece of software or webpage:

* *copyright* – weak and narrow (due to the idea-expression dichotomy), but can still protect icons, audio and video if they carry enough originality
* *trade dress* (e.g. *Geschmacksmuster* in German) – the special look and feel of something has to be distinctive and non-functional; does not need to be registered, but is hard to fulfil
* *industrial design rights* (e.g. design patents in the US) – need to be registered (e.g. the battle between Apple and Samsung with regard to similar icons)

In practice, though, these are unlikely to cause issues for presenting software in a museum or archive.

### Hardware

#### Hardware circuits, schematics, etc.

Although not something you would encounter often when archiving software, computer museums still have great interest in different hardware schematics, circuit plans, etc.

These can be protected by different integrated circuits rights. In the US these are commonly referred to as “mask works”, whereas in the EU we talk about “legal protection of topographies of semiconductor products”. Such rights are typically more limited than copyrights or patents, and in several jurisdictions they are automatic.

## Domain names

### Internet domain names

Internet domain names do not fall squarely into any other IP rights, though the European Court of Human Rights ruled that they are “property” (*ECHR: Paeffgen GmbH vs Germany 25379/04*). In practice, trademarks can be used as legal defense in domain name disputes.

## How to obtain rights

Unless you are the actual rights holder, you need to obtain the rights through a license or otherwise.

### Free (as in “freedom”)

There are two ways that a piece of software can be free to use by everyone in a reliable and legal fashion.

The first is that the copyright and eventual patents in the software – perhaps even trademarks, but these are of lesser concern – have already expired. This means that the software has fallen into the public domain[[32]](#footnote-31) and is now free to use for everyone.

The other option is that the software is released as Free and Open Source Software (FOSS). Since FOSS gives all the recipients freedoms to *use*, *study*, *share* and *improve*, this is more than enough (by far) for a museum to work with.

### Exceptions by law

Another way that software can be used for free by anyone is based on the related statutory limitations and exceptions, and thus we are given these rights by the law itsel[f](#bookmark=id.3ygebqi)[[33]](#footnote-32).

*Communication to the public on screens* (ZASP[[34]](#footnote-33), §49.b) could be used for exhibitions, but for certain in order to give people access (to software or other material) via a computer screen to research and learn from the exhibit.

*Private and other internal reproduction* (ZASP §50) allows public museums, archives, etc. to make copies on any medium, under the condition that they do so from a copy they already own and that they are not making a profit with the reproduction.

*Free reproduction and offering to the public of orphan works* (ZASP 50.a) can be used when a work without a known author is found. In such cases the law permits public museums, archives and so on to digitize, freely reproduce, display and even distribute the work in question. However, the law does prescribe a careful search and logging process that needs to be followed in such cases. A further and significant complication is that §50 does not count software as potential orphan works, and it would make sense to propose a change of law here.

### License

Since very little software is already in the public domain and ZASP does not have many exceptions for software, you will most likely need to obtain a license.

When you negotiate a license with the rights holders, it is important to think broadly and include the general public, or other museums/archives, at the very least.

As such, the easiest way to obtain all needed rights possible would be to negotiate a FOSS license. The more permissive[[35]](#footnote-34) the license, the fewer limitations you (and others) would run into when using the procured software.

If that is not possible, try to negotiate a custom license for *museastic (and archival) purposes*. Limiting the license to this specific use case would still enable other museums (and archives) to use this work, and also clear any potential issues with your museum having an exhibition elsewhere.

In exceptionally difficult negotiations, an option is to put the software under escrow and/or temporary NDA. That way you could secure the software in your museum/archive and agree with the rights holder to release it (ideally as FOSS) to the public after a certain amount of time has passed or a certain condition has been met.

### Public bodies

If the software in question was written or procured by a public body, one might be able to use the Public Information Access Act (ZDIJZ) to request the source code and the rights to reuse.

EU-wide this is known by the slogan *Public Money, Public Code*, and in some countries (e.g. Italy, France) the law explicitly requests this. In Slovenia the law does not explicitly mention software, but a reasonable interpretation could lead to this, although it has not yet been tested in court.

## Other considerations

Online repositories as source code forges (GitHub, GitLab, etc.), image galleries, social media websites and so on all have their *Terms of Service*. Even if you are a museum, these apply and often include IP rights.

Moreover, when working in this context it is likely that you will eventually be faced with complaints and requests to *take down* (claimed) copyright violations and remove personal data (under the GDPR) – so be proactive and have a policy on how to handle these.

# 7. Pioneering Computer Support in Urban Planning, 1970–1985

Franc J. Zakrajšek

## 1. Introduction

At the beginning of the 1980s, the world witnessed the beginning of the intensive introduction of computer technology into various professional fields and disciplines, causing a radical transformation in many areas, including in terms of content, work processes, and use.

In the article, I share my contribution to the research, development, and implementation of computer support in urban planning in this period. My pioneering work in this field could only be done with the cooperation of the innovative and efficient interdisciplinary team of other mathematicians, computer scientists, architects, geographers, economists, sociologists, and other experts. I am a researcher, developer, and consultant employed at the Urban Planning Institute of the Republic of Slovenia.[[36]](#footnote-35) The framework of this article allows only a modest presentation of selected fragments of that period to give the reader some impressions of that time and our role and contribution to it.

## 2. Information system for spatial planning

We developed an information system for spatial planning (urban/regional) (ISSP) with two components: a database and a set of procedures. The database consists of data about past and present states and data about planning decisions in the form suitable for use within the planning processes. The essential part of the database is geocoded (geographical reference) data. The set of procedures consists of procedures for database creation and updating, as well as for database use and analysis. The essential part is data use procedures such as data retrieval, methods of statistical analysis, and urban and regional models.

ISSP is a decision-support system designed to satisfy data needs within various planning and monitoring processes on communal, municipal and/or regional levels. It is a user-oriented information system. The system’s main function is not data collection but data processing for end-use. ISSP is not a stand-alone information system, it is highly connected with the other information systems in the country. It thus uses data originating in various information systems (e.g., administrative data, statistical data), and produces data that could be used outside the planning area (e.g. data for local administration).

The primary development in this system’s approach (in the sense of innovation and challenge) was integrating the latest knowledge from the theory of information systems, especially that promoted by the field guru at the time, James Martin. I am thinking here of his manifesto of information systems, and the idea that data entities and thus data are more stable than the procedures we want to automate, or that relational databases are in the third normal form, along with work on information technology engineering, computer-aided software engineering, fourth-generation programming languages, RAD (Rapid Application Development), and more. I successfully integrated all this with my mathematical knowledge of topological structures (point, line, polygon ...) and their mutual relationships, which we now know as the basis of geographical information systems.

*Figure 1: Representation in physical space, mathematics of geo-topological kernel, and third normal form. Source: Zakrajšek, F.: Designing an information system at the level of the city of Ljubljana. Ljubljana: Urban Planning Institute of the Republic of Slovenia, 1981*

*Figure 2: Standard database of the information system for spatial planning. Source:* Zakrajšek, F. Information systems for urban and regional planning. In: Gerckens, L.C. (ed.), et al. *Planning in transition: regional development, urban form, information system: symposium 1985, Columbus, 21–22 May 1985*. Columbus: The Urban Institute of Slovenia: The Ohio State University, The City and Regional Planning Department, 1985.

## 3. Data capture example: centroids of house numbers

In the 1980s, capturing graphic data was a special challenge. Scanners of larger dimensions did not exist, and those smaller than A4 were difficult to access. Therefore, we had to develop semi-automatic, partly manual procedures for digitization as well as a whole series of software interfaces for transferring data to/from punched paper tape. Let me mention our special innovative approach for digitizing house numbers. The coordinates of house numbers are essential in urban systems because, in this way, we can monitor the state of “living” activity (residents and apartments) and the activity of “work” in the space. We first successfully digitized the house numbers of Celje (approx. 20,000 units) and, a little later, the house numbers of Ljubljana. Their positional accuracy was 5 m. The Surveying and Mapping Authority of the Republic of Slovenia then adopted our method for digitizing all house numbers for the Republic of Slovenia. It transferred the data of the digitized house numbers of Celje and Ljubljana to its EHIS database, which is still regularly maintained. It is worth emphasizing that Slovenia was among the first countries in the world to digitize all house numbers on its territory.

*Figure 3: Data Capture, Centroids of House Numbers. Source: Zakrajšek, F. et al.: Information design of the Municipality of Celje during the preparation of the spatial plan. Ljubljana: Urban Planning Institute of the Republic of Slovenia, 1979*

## 4. Data output example: first digital maps

Similar to large-format scanners, there was also a problem with large-format plotters and printers at the beginning of the 1980s. To make the first digital maps available to the greatest extent possible, we initially used ordinary line character printers and created fewer and darker areas of the map by overprinting. The problem of character size (rectangle and not square) was solved by mathematical 2-dimensional interpolation.

We prepared a library of computer programs for handling (processing and storage) the cellular organization of the spatial database. The cellular organization of a spatial database is called an organization, where the basic spatial unit is a grid cell (square) for which we have a specified vector of data. The purpose of the program library is to simplify the programming of applications for entering data, analysis (statistical, spatial ...), models (projections, simulations ...), and for outputting results (graphic displays, tables, diagrams ...). The library contains programs for:

* creating, opening, closing, importing, and exporting data from the database (C-files, CF-files),
* print data to a line printer (GRAFC7, GRAFC8),
* data plotting on a Versatec plotter (GRAFCP).

We wrote most of the programs in the programming language FORTRAN (FTN), but some of them in COMPASS and PASCAL 6000. Cyber record manager-basic access methods (word addressable) are used to manage the databases with the support of the NOS operating system, /BE 1.3 on a CYBER 72 computer.

*Figure 4: Printouts of maps for the Celje municipality. Source: Zakrajšek, F. et al.: Information design of the Municipality of Celje during the preparation of the spatial plan. Ljubljana: Urban Planning Institute of the Republic of Slovenia, 1979*

## 5. Methods example: expert systems

In this period, we developed several procedures for database creation and updating as well as for database use and analysis:

* elementary algorithms dealing with segment-oriented database (e.g. point-in-polygon, minimal path), preparation of experimental software system for management (update, retrieval) of the segment-oriented databases;
* models of population projections and simulations (“cohort survival method”);
* spatial interaction models (Lowry-like models based on the principle of entropy maximizing with inequality constraints);
* employment model for estimation of the jobs needed, based on population projection;
* housing model for estimation of the apartments needed, based on population growth and the existing housing fund;
* multi-zoning population projections and simulations (“cohort survival method”), maps overlapping, grid-based maps of different physical indicators (e.g., type of soil, relief, land use) are overlaid with a combination of “and” and “or” relations;
* grid-based catchment area simulation model for the location of local services (e.g. primary school, nursery school, health station).

We realized very early that using rule-based expert systems or an automatic learning system is very promising for use in urban and spatial planning. We collected some examples of the use of expert systems in related fields, identified potential opportunities in spatial planning, and reviewed the availability of existing end-user software. After rethinking the gathered material we developed several expert system prototypes, named UEXPERT (*Figure 5*).

*Figure 5: UEXPERT prototype case. Source:* Zakrajšek, F. *Možnosti uporabe ekspertnih sistemov v lokacijskem postopku*. Ljubljana: Urbanistični inštitut SR Slovenije, 1985

## 6. Implementations: Celje and Ljubljana municipalities

For me, the reason for getting involved in implementation projects is twofold: to advise and assist in the actual development of an information system for urban and/or regional planning in a specific area, and get practical experience for further research work and preparation of legal acts in the spatial planning field. In the period 1970–1985, we implemented the following ISSP:

* Municipality of Celje (60,000 inhabitants, 250 km2); Two databases were established. First, the “basic database” with a house number as the basic georeferenced unit to which microdata from several registers are linked (population, housing, business premises, vehicles). Second, the “planning database”, where the base georeferenced unit was a grid cell (100x100 m), data consisted of aggregated data from a basic database and directly captured data on physical spatial characteristics (altitude, terrain slope, orientation, soil stability, agricultural land, forest land, protected areas, etc.). The database was used in the preparation of the Celje municipality spatial plan.
* Municipality Ljubljana–Center (32,000 inhabitants, 5 km2); In designing the contents of Ljubljana information system we used the general principles of ISSP, and also the following specific demands have been considered: specific areas in the Ljubljana–Center commune (CBD, renovation areas), besides urban planning it also supported specific administrative procedures (e.g., a process for location permission), existing data sources were used as much as possible (e.g., 1981 census data, register of business premises, register of territorial units), this was the pilot implementation project for the establishment of the register of buildings. The information system included a very detailed database with micro data on business premises, buildings, households, and inhabitants (see the scheme enclosed).
* Municipality of Ljubljana (200,000 inhabitants, 500 km2); The essential part of the implementation of ISSP Ljubljana was monitoring the actual land use compared to planned land use. The purpose of this monitoring was to get permanent and systematic evidence of all changes in planned land use, simple and publicly accessible insights into the regulation status of planned land use, permanent and analytical comparisons between actual and planned land use, and simple and objective evaluations of the consequences of suggested changes in planned land use. The central part of the information system was a database where various types of data were located in planning zones. The information system was used in the preparation of spatial planning acts, part of a project called Ljubljana 2000.
* The Ljubljana region (16 communities, 600,000 inhabitants, 5,000 km2); The regional locational model has been implemented in the Ljubljana region within the preparation of the regional development plan as part of the Ljubljana 2000 project. Six different alternatives to urbanization have been prepared and examined with the support of the regional locational model: concentration, deconcentration, north, south, residential neighborhoods, and balanced.

*Figure 6: Information base of the information system for planning the space of the municipality of Ljubljana. Sources: Zakrajšek, F. et al.: Information design of the Municipality of Celje during the preparation of the spatial plan. Ljubljana: Urban Planning Institute of the Republic of Slovenia, 1979, Souvan, T., Zakrajšek, F. et al. ISUP: informacijski sistem za urejanje prostora občine Ljubljana Center. Ljubljana: Urbanistični inštitut SR Slovenije: Zavod za izgradnjo Ljubljane, TOZD Urbanizem-LUZ: Uprava za avtomatsko obdelavo podatkov Skupščine mesta Ljubljane, 1985*

## 7. Implementation: Guyana

From 1983–1985, I had the opportunity to participate as part of an international consulting team in Guyana in the project “Institutional Support for National Systems of Planning and Projects in Guyana”. The project was financed by the InterAmerican Development Bank. I managed the sub-project “Information System for Planning and Projects”. The Cooperative Republic of Guyana was a then developing country in South America with about 800,000 inhabitants. The purpose of developing the information system for planning and projects was to provide data support for national economic and social planning and central coordination of sectoral and regional planning, monitoring/planning projects, and corporation monitoring at the national level. We developed the information system in two phases:

1. Recommendations: diagnosis, the concept of the information system, information engineering for the information system, recommendations for immediate improvement.
2. Initial implementation: the clarified concept of the information system, the survey of existing data, the survey of computer resources, initial data dictionary, experimental database – list of current projects, experimental data base – 1980 census data.

At that time we developed MDBMS2 (Micro Data Base Management System Version 2). It was a simple database management system written in BASIC for ID-80 (Iskra Delta-80, the microcomputer that came before the ID Partner). It was used for a database of urban planning documents held by the City of Ljubljana. In Guyana we rewrote the program in BASIC for an Apple IIe. The program was used for database management for a list of projects in Guyana (*Figure 7*).

*Figure 7: Scheme of information system for planning and projects for Guyana. Source: Zakrajšek, F. Recommendations on development of an information system for planning in Guyana. Georgetown: Urban Planning Institute of Slovenia, Development Planning Center, 1983*

## 8. Comparisons and disseminations

When innovating complex information systems, it is necessary to constantly check the development and implementation of foreign efforts in this field. Nowadays, this is much easier due to the Internet and the wider availability of information. At that time, it was necessary to rely on the latest professional books, scientific articles, publications from professional conferences, and direct visits to individual professional institutions. Hence, to get information in 1983 I had to visit some universities, institutes, and other European institutions and make an evaluation. At the end of this trip, I wrote some observations:

* In the field of urban planning information systems, we can claim that our research is entirely comparable to that in foreign countries (although they are otherwise more financially supported and, above all, have better computer equipment at their disposal), so we count on the exchange of specific research results.
* The development is towards the implementation of less complex, cheaper information systems, which provide financial effects in the form of savings in a short period.
* According to the state of information systems that actually work in practice in this area, Sweden and some cities in Great Britain are well ahead of us, while we can note that we surpass Scotland in some areas in terms of implementation.
* In the countries visited, they placed a great deal of emphasis on introducing microcomputers and developing software for microcomputers. Still, during the visit they did not yet have any significant experience in this field.

On the other hand, and despite the easier access to information, it is still vital today to verify research results in professional journals, publications, and conferences. During this period, we published research results almost every year at least at two conferences within the former Yugoslavia, one in the field of spatial planning and the other in the field of computing and informatics. A fundamental form of dissemination was also the publication of our achievements, and some title pages are presented in *Figure* 8.

We printed several hundred copies of the publications and “sold” them to institutions throughout the former Yugoslavia. Despite the publications being in Slovenian, there was great interest in them.

*Figure 8: Cover pages of the publication* Information systems for spatial planning*. Volumes: Guidelines for development, The concept of the system, Microcomputer equipment for managing relational databases, Information system of regional facilities, The possibilities of using expert systems in the process of location permit, Spatial planning methods at the municipal level – population projections, The possibilities of using expert systems in spatial planning, Starting points for the selection of hardware and software, System of planning indicators for monitoring the land use in Ljubljana, Renewal of the concept of information systems*

## 9. After 50 years

After 50 years, we are still very present in the research, development, and implementation of computer support in urban planning. Some links for our recent work are provided in *Figure* 9.

*Figure 9: After 50 years*

*Figure 9a: 3D Urbanism, Ljubljana, 2010 onwards. Link:*

[*https://3durbanizem.ljubljana.si/3DUrbanizem/3DMesto/*](https://3durbanizem.ljubljana.si/3DUrbanizem/3DMesto/)

*Figure 9b: Agent-based Geographical Modeling of Public Library Locations, 2010 onwards. Link:*

[*https://www.sciencedirect.com/science/article/pii/S0740818819302890*](https://www.sciencedirect.com/science/article/pii/S0740818819302890)

*Figure 9c: SMUR, Simulation Model of Urban Development, 2010 onwards. Zakrajšek, F. et al. Development and implementation of a regional simulation model for the Ljubljana urban region: a regional simulation model. Ljubljana: Urban Institute of the Republic of Slovenia*

*Figure 9d: Simple Cellular Automata Model for Flood Estimation and Determination of Culture Heritage on Risk, 2018 –2022, STRENCH: STRENgthening resilience of Cultural Heritage at risk in a changing environment, Central Europe Programme, 2020–2022*

# 8. Are values in software development time-critical or can they be stored?

# Silvana Žorž

The purpose of this paper is to investigate whether software can be analyzed retrospectively from the point of values through Values Sensitive Design methodology (VSD), which in the current ethics discourse in computing enables us to account for human values in software design. It consists of a tripartite investigation process – conceptual, empirical and technical – where the empirical[[37]](#footnote-36) focuses on examining the human context in which the technology is, or in our case has been, situated (Žorž 2017).

## VSD Empirical Investigation – HERMES SoftLab, Slovenia

HERMES SoftLab (HSL) was born in the first wave of business start-ups after a change in the prevailing social order. Based on the experience of working with Hewlett Packard (HP) from 1984–1990 (a period of self-governed socialism), the four founders – Rudi Bric, Tomaž Schara, Andrej Kuščer and Zoran Zagorc – started a software engineering company in October 1990, with the main business activity being creating software for others. At its high point in 2002 the company grew to 700 employees and absorbed an important percentage of available software developers in Slovenia, establishing branches at home and abroad.

Honesty in software programming was the main motivator for Rudi Bric to follow a certain set of values in HSL software development. This honest approach to software development was based on an experience Bric had with regard to a bug in a piece of software made by a foreign company that he encountered while working for ISKRA Računalniki in 1980, long before starting HSL.

The bug in system software actually enabled the American company Control Data Corporation (CDC), which produced CYBER-18 mini computers and application software, to gain a position of power against the buyer of the software, the Mexican company Diconsa. Diconsa had to pay to “resolve the bug” repeatedly in order to be able to continue using the application software. Moreover, the bug was not actually fixed by CDC, which preferred to accuse the Mexicans of being ignorant users. Bric knew exactly where the problem was, and how to solve it. Ultimately the problem was solved by ISKRA engineers, but the dishonest relations between some big companies and their customers made an impression on Bric, as well as the idea that you have to fully understand all the implications of the use of the developed software in order to adequately control and assure the quality of software development. This experience led Bric to implant, years later, into HSL’s culture the value of responsibility in development, as expressed in the saying “Good Work Creates New Work”.

HSL’s first really big client was Hewlett Packard (HP). In the 1990s it was considered the standard in terms of high-quality software development. HSL thus started to form its own development process according to how this was done in HP. Bric was interested in finding out if the company was capable, organized and knowledgeable enough to be able to compete in the field of software development with anyone in the world. Approached from an engineering perspective, HP was taken as the gold standard, as explained by Krajnik. Already at the very early beginning of HSL specific people were brought in whose role was to think about the software engineering process and all related aspects, with the aim of being able to produce software that could compete globally. Here, much knowledge of such processes was transferred from the relationship between HP and HSL, defined by Bric as a lucky coincidence that involved the founders of HSL, people from the previous company HERMES (already the distributor for HP in Yugoslavia), and Prof Zvonko Fazarinc and Dr Franc Rode, senior scientists employed in HP (and personal friends of both Hewlett and Packard). In 2000 HP bought a 5% share in the company itself. This in-depth relationship combined with the vision of good work led to the transfer of work practices into HSL.

As explained by Miro Germ, Director of Quality Control at HSL, who came from HERMES Plus, these practices[[38]](#footnote-37) of HP were transferred through the HP Yugoslavia representative company into the mindset of people creating HSL. HP recommendations were used in creating business strategy and quality control processes through the Total Quality Management (TQM) tool, as well as Total Quality Essentials by Sarv Singh Soin, the TQM Director of HP Pacific.

The HP Way, which defined the corporate objectives of HP and “the way things are done around here” (Lacy and Mullins, 2002) was also fully transferred, as the management was working for HP Yugoslavia in Vienna. The HP Way clearly states the following as an aim with regard to uncompromising integrity as a company and goal for employees:

“*People at every level are expected to adhere to the highest standards of business ethics and must understand that anything less is unacceptable.*” (HP Alumni, 2022)

This can be understood as the explanation of the 7th corporate objective, citizenship, which is defined:

“*To meet the obligations of good citizenship by making contributions to the community and to the institutions in our society which generate the environment in which we operate*.” (HP Alumni, 2022)

Germ also made a presentation at the European Organization of Quality (EOQ), showing that HSL was included in the quality conversation at that time, and later also served as assessor of quality in different quality management societies. Primož Krajnik, CEO of Zaslon (a Slovenian company in the field of banking software that HSL bought in 2000) was the youngest assessor of business quality for the European Foundation for Quality Management at that time. HSL had a very defined quality assurance process which came first, and only after this did the programming work begin. At one point a project dashboard existed for each project, where quality managers signaled if the project ran in adherence to the rules defined before the start of the project. This process was also transferred from HP.

Another important notion set out by Bric was the fact that it was not only thought about how to set up the process, but also how to develop people to properly manage the software development process and the management of people who were part of this process. This led to the creation of an engineering culture where constant learning, development, sharing of best practices, improving of mistakes and focus on excellence in quality shaped teams into a formation of equals working towards a standardized goal. At the same time the diversity of teams was high, as at one point there were 14 different nationalities within the company (Tagesspiegel 1998).

In 1998 the Republic of Slovenia’s Business Excellence Prize (PRSPO) was awarded to HSL, making HSL the first winner of such a prize in Slovenia (Rozoničnik and Valenci, 2017) and an example of a company showing excellence on the EFQM model of excellence.

Besides the recognition from the industry and different quality management associations, the idea behind HSL – as explained by Krajnik, the person who was in charge of transferring the HSL “way” into Zaslon – was to create work that has the client’s best interests in mind, rather than simply trying to get the most profit out of the client, and instead to really solve the client’s problem through software solutions and ensuring their independence. Any resulting profit was the effect of this focus on high standards of quality software production.[[39]](#footnote-38)

As explained by Zoran Zagorc, a founder of HSL, this high-standard practice of project management was based on the know-how developed through the work of the founders and the quality control management practices set out by Germ. This differentiated HSL from others, and helped realize the goal of delivering high-quality software to international clients, which was the starting vision of the founders. Their initial idea that good work creates new work focused on overdelivering on deadlines and quality, which then translated into know-how with regard to project management, development and quality control. HSL had full control over software in terms of development, and an advantage in development knowledge that gave it the opportunity to do whatever it wanted with software, which could lead to moral issues. But going against the clients’ interests was not the mindset of the company. As Zagorc explains it, the individuals working for HSL and the founders were themselves ethical people, due to differences in the ethical component of society at that point in time. Zagorc also mentions the need for ethics in the case of HSL creating software for privatization in Slovenia, and for public sector use. In this case ethics were central to the development process at HSL, leading to software that secured the role of the user and not the owner of the software. This also gave Zagorc insights that could give him an unfair advantage, which his ethical character and honesty prevented him from abusing. It also signals the presence of moral character in the company, which has also been noted by others.

An important notion expressed by Luka Renko, Chief Technologist of HSL, is the fact that HSL was doing product development in which the final users’ needs, and not the company that bought the software, were of greater importance to HSL. The focus was on delivering the most value to the user, which could lead to differing perspectives between HSL and the client. This led to HSL being proactively attuned with the needs of the market[[40]](#footnote-39) and users sooner than the client, and this produced many fruitful business relationships between HSL and its clients for years to come. The wish for the greater independence of clients led to hybrid teams and the intellectual property of software being in the hands of clients and not HSL. HSL was, due to its values and quality standards, able to develop software for competing clients by keeping teams separate, so they did not know what the others were working on. But there was no other direct competing client with regard to HP for which the company was developing software. What is important to note is that within the company there was a friction between the cultures of how work was being done for international or local clients, where the difference was between product development for international clients or local client servicing. The notion of responsible software was not actually a defined issue or concept, as at that point consumer software was not the standard of how software was used.

HSL in its essence tried to achieve in its development the values of good work, and independence, which could be retroactively translated into the VSD values of ownership, universal usability and autonomy values allowing us to start the VSD conceptual investigation (Žorž, 2017).

## Current conclusion

The VSD analysis on the case study of HERMES SoftLab (HSL) shows a series of issues with regard to analyzing the values of past software.

The first is the fact that the historical background – context – shaping the people’s culture at that time is no more. As mentioned by Zagorc and Krajnik, the values of the software industry at the time of HSL were the values of a certain period and cultural background, where everyone had been raised to believe that what is best for society is more important than what is best for an individual. As mentioned by Bric, the HP Way was similar to the belief model that was promoted in Slovenia and Yugoslavia, the societies in which the founders of the company and employees actually grew up.

The other issue is that due to the fact that HSL was sold to Comtrade in 2008, we are actually unable to run a VSD technological investigation as additional research should be needed to determine which software could serve as the artifact representative of HERMES SoftLab.

From this we can conclude that values in software development are time-critical and hard to store, due to the shifting nature of software, usage, and ownership, as well as documentation practices and standards.

*Figure 1: Bronze sculpture for HERMES SoftLab office at Litijska 51, Slovenia, by Andrej Ajdič (1997) where the wings of an angel consist of four surfaces on which the following is written: Hermes to HERMES SoftLab Hewlett Packard contract transfer, the HP Way, the Award for Entrepreneur of the Year 1997, and 1996 employee phone registry (Image courtesy Rudi Bric)*

## 10. Biographies

## Roberto Di Cosmo

Roberto di Cosmo is a computer scientist, and in 2010 he was the first director of IRILL – Initiative de Recherche et Innovation sur le Logiciel Libre – an initiative for innovation and research with regard to free and open software. After a decade of teaching at the Ecole Normale Supérieure in Paris, he became a tenured professor of computer science at University Paris Diderot in 1999. He is active in research in the field of theoretical computing, especially functional programming, parallel and distributed programs, programming language semantics, system types and linear logic. He now focuses on new scientific problems created by the ubiquitous use of free and open-source software, which was core to the European research project Mancoosi. He is monitoring the social changes induced by information technology with great interest, is a long-time proponent of free software, and contributed to its establishment with a bestselling book, *Le Hold-up planétaire* (*The Planetary Hold-Up*) in 1998. In 2007 he founded a thematic group for free programming at Systematic, which has led to the creation of more than 60 open-source research and development projects over the last decade. Di Cosmo was an early member of AFUL, the French Linux society. His most known contribution to Linux is the first live distribution of Linux (2000 to 2002), demolinux, which enabled running Linux from a CD-ROM. In 2015 he established and now leads Software Heritage, an initiative for creating a universal archive of all publicly accessible source code.

#### Primož Jakopin

Primož Jakopin was born on June 30, 1949 in Ljubljana, Slovenia, into a family of linguists. His father Franc was a professor of eastern Slavic languages with publications in the field of grammar and onomastics, while his mother Gitica translated more than 50 novels from English, German, French, Russian and Polish. Because his main extracurricular activity was caving he intended to study geology, but followed his father’s advice with a degree in mathematics at the University of Ljubljana in 1972. In 1981 he received a Master’s degree in information science with the study “[Entropy of Names and Family Names in Slovenia](http://www.jakopin.net/primoz/magisterij/index.php)” at the University of Zagreb, and in 1999 a PhD in information theory for “[Upper Bound of Entropy in Slovenian Literary Texts](http://www.jakopin.net/primoz/disertacija/index.php)” at the University of Ljubljana. From 1972 to 1984 he constructed a numerical model for the approximation and visualization of karst caves. It was used to compute the volume of several caves and collapsed dolinas in Slovenia, and the results were reported at the [8th International Congress of Speleology](http://www.jakopin.net/primoz/clanki/1981_MCV/index.php) and [3rd European Symposium of Stereology](http://www.jakopin.net/primoz/clanki/1981_MECS/). After he failed to continue this research at the Karst Research Institute in Postojna, Jakopin changed his area of interest to language technology, closer to his parents field of work. Another great passion was also computer programming, and with software he developed mostly at home he managed to make a career in computational linguistics, both in research, as head of the [Corpus Laboratory](http://web.archive.org/web/20190406003335/http://bos.zrc-sazu.si:80/index_en.html) at the Fran Ramovš Institute of Slovenian Language, from its foundation in 2001 until 2012, and in teaching, at the Faculty of Arts in Ljubljana (1993–2012) and at the School of Humanities in Nova Gorica (2012–2016). The major pieces of software he worked on include: STAT (statistical package, Control Data Cyber, 1977), IBIS (software package for data of variable length, mainly used for handling secondary library information such as data on books and articles, Digital DEC 10, 1981), INES (text editor with database and graphics, Sinclair ZX Spectrum, 1985, widely used in ex-Yugoslavia), STEVE (16-bit character text editor with database, graphics and DTP, ATARI ST, 1987–1992, distributed in ex-Yugoslavia, Germany, Benelux and Norway, with manuals in English, German, Slovenian and Serbian/Croatian), EVA (text editor with database and language technology that features, 135,000 lines of code in C, for DOS in 1992 and Windows operating system in 1996–), NEVA (EVA-based search engine for Windows servers, 1999–).

#### Matija Šuklje

A legal counsel by education, a hacker at heart, Matija Šuklje feels most at home at the crossing of both those worlds. Since the late 1990s he has dedicated his skills to FOSS (Free & Open Source Software) and collected a vast array of experience in the field in the public, NGO and private sectors. He spent many years leading the largest global network of legal experts, specialized in open-source topics, where he is still an active member. He currently holds the position of Senior IP Counsel at Liferay.

#### Boštjan Špetič

Boštjan Špetič is Head of Museum Collections at the Computer History Museum, Software Heritage Ambassador, entrepreneur and start-up mentor, currently living in Slovenia, but keeping close connections with New York, where he lived for most of the past decade. Boštjan Špetič co-founded, built and sold two products within the same start-up, Zemanta. The first was a personal assistant based on semantic web technologies, which was acquired by an independent publishing-technology company Sovrn, the second product was the first programmatic advertising platform (DSP) specialized for native advertising, and was acquired by Outbrain. Boštjan currently works in Outbrain’s development group as a product specialist, is a mentor to several entrepreneurs worldwide, and a board member of Rock Content.

Saša Divjak

Saša Divjak is the author of various program applications, and lead and collaborator on a series of international and local projects in the field of computer automatization, robotization and multimedia technologies. He has been active on the boards of various international and local professional associations. He was head of the Department of Electrotechnics at the Institute Jožef Stefan, assistant to the CEO of Iskra Delta, Dean of the Faculty for Computer Science and Informatics at the University of Ljubljana, and Director of the Laboratory for Computer Graphics and Multimedia. He was a lecturer for the subjects Programming 2, System Software, Operating Systems, and Computer Graphics.

#### Silvana Žorž

Silvana Žorž, Master of Digital Humanities (KU Leuven) and Master of Economics (Master in Business and Administration – IBM, University of Ljubljana). In her Master’s thesis, she researched the discovery of discriminatory factors in software design, under the mentorship of Dr B. Berendt (KU Leuven, Belgium). She obtained her Master’s *cum laude*, and dedicated her work to the value system in software design methodology. She has collaborated with many companies (Google and Outfit7, among others) in the field of digital marketing, digital product development, market research and consumer habits. She is currently a senior advertiser with the digital agency Red Orbit.

#### Nataša Milić-Frayling

Dr Nataša Milić-Frayling is a founder and the CEO of Intact Digital Ltd, a company that provides a platform and services for hosting legacy software installations to enable long-term readability and use of digital data. Intact Digital works with highly regulated sectors such as pharma and life sciences to support compliance with data integrity regulations, reconstruction of research studies and reproducibility of data analyses, including machine learning and AI.

Nataša Milić-Frayling has 25 years of experience in computer science research and innovation, including 17 years at Microsoft Research (MSR). She has authored over 100 research publications and has dozens of approved patents to her name. Besides her research, Nataša has also led the MSR Research Partnership Programme, promoting collaboration with innovation partners on strategic challenges, including digital preservation and long-term access to digital content ([PLANETS](http://www.planets-project.eu) and [SCAPE](http://www.scape-project.eu/) EU projects). She is Professor Emerita at the University of Nottingham, where she spent five years serving as Chair of Data Science, contributing to the university’s research strategy on data science and AI.

She is actively engaged with the broader professional community on critical issues that arise from the prolific use of digital technologies, ranging from professional ethics, privacy and design transparency to digital obsolescence and responsible innovation. She is a member of the Preservation Sub-Committee within the UNESCO Memory of the World Programme and serves as Chair of the Research and Technology Working group for the UNESCO PERSIST project. Nataša is an active member of the Association for Computing Machinery (ACM). She served on the ACM Europe Council and as Chair of the ACM Women Europe Executive Committee.

Franc J. Zakrajšek

Franc J. Zakrajsek is a mathematician, senior researcher, computer programmer, and consultant, working since 1970 developing geographical information systems in urban planning and related areas. His work focuses on innovative approaches, methods, simulation models, and introducing artificial intelligence in e-planning. He was a member of the expert group on digitization and digital preservation established by the European Commission until 2015. He was one of the founders of the Register of Cultural Heritage in Slovenia. He participated in several EU projects introducing geospatial information and 3D modeling in the field of cultural heritage, museums, libraries, and archives as a part of the Europeana-European digital library and developed the Europe-wide eCultureMap. He also led numerous local and national projects and was appointed as a lecturer by UNDP, UN-HABITAT, and TAIEX. He was a consultant for the InterAmerican Development Bank and has published more than 500 articles and other publications.

1. GNU91. GNU general public license, version 2, 1991. Retrieved September 2015. [↑](#footnote-ref-0)
2. Created in 1967, Inria is a public scientific and technological institution specialized in mathematics and computer science, under the dual supervision of the French Ministry of Higher Education, Research and Innovation and the Ministry of Economy and Finance. [↑](#footnote-ref-1)
3. See SWHAP at https://www.softwareheritage.org/swhap. [↑](#footnote-ref-2)
4. [Release Notes Flash Player 32 AIR 32 (adobe.com)](https://helpx.adobe.com/flash-player/release-note/fp_32_air_32_release_notes.html)  [↑](#footnote-ref-3)
5. [The Internet Archive Software Collection: Internet Archive](https://archive.org/details/software) [↑](#footnote-ref-4)
6. [Računalniški muzej, Ljubljana, Slovenia](https://www.racunalniski-muzej.si/) [↑](#footnote-ref-5)
7. [Computer History Museum, Mountain View, CA](https://computerhistory.org/); [Centre for Computing History, Cambridge, UK](http://www.computinghistory.org.uk/); [List of computer museums around the world - Wikipedia](https://en.wikipedia.org/wiki/List_of_computer_museums) [↑](#footnote-ref-6)
8. [Intact Digital Ltd](https://www.intact.digital/) [↑](#footnote-ref-7)
9. Milic-Frayling, Natasa, and Marija Cubric. "Executable Archives: Software integrity for data readability and validation of archived studies." International Conference on Digital Preservation 2021 (iPres2021), Beijing, China. [↑](#footnote-ref-8)
10. [VRML Virtual Reality Modeling Language (w3.org)](https://www.w3.org/MarkUp/VRML/) [↑](#footnote-ref-9)
11. [Cortona3D Viewers for Windows](https://www.cortona3d.com/en/cortona3d-viewers-windows) [↑](#footnote-ref-10)
12. [Michael Takeo Magruder](http://www.takeo.org/#) [↑](#footnote-ref-11)
13. [World[s]: Michael Takeo Magruder: 2006](http://www.takeo.org/nspace/ns018/) [↑](#footnote-ref-12)
14. Computer History Museum, Zbiralna politika (internal research documentation), Ljubljana 2019 [↑](#footnote-ref-13)
15. The Computer History Museum collaborates with an international network of experts for various subset level expertise access, but in the case of Slovene computers the expertise is entirely homegrown. [↑](#footnote-ref-14)
16. In 1981, after years of searching for a sensible solution that would prevent the duplication of development and work between several companies in the country. [↑](#footnote-ref-15)
17. Computer History Museum, internal research documentation, Miha Urh 2022 [↑](#footnote-ref-16)
18. Computer History Museum, internal research documentation, Miha Urh 2022 [↑](#footnote-ref-17)
19. M. Hlavaty, Zmogljivosti in uporaba opreme za avtomatsko obdelavo podatkov 1988, Zavod RS za statistiko, Ljubljana, 1990 [↑](#footnote-ref-18)
20. Grobelnik [↑](#footnote-ref-19)
21. http://eprints.fri.uni-lj.si/3945/ [↑](#footnote-ref-20)
22. https://github.com/tstih/idp-dev [↑](#footnote-ref-21)
23. https://www.racunalniski-muzej.si/wp-content/partner/ [↑](#footnote-ref-22)
24. https://en.wikipedia.org/wiki/Snake\_(video\_game\_genre) [↑](#footnote-ref-23)
25. https://zbirka.muzej.si [↑](#footnote-ref-24)
26. Netpreserve.org International Internet Preservation Consortium General Assembly & Web Archiving Conference 2022 [↑](#footnote-ref-25)
27. The term, while already well established, is somewhat misleading, since property deals with tangible and rivalrous goods, while intellectual property deals with non-rivalrous and intangible goods. Creating a copy of a book does not take the book away from its original owner. In this effect it is a means to artificially create scarcity where naturally there would be none. [↑](#footnote-ref-26)
28. Due to the rise of non-practicing entities (“patent trolls”), several patent pools and protection schemes have emerged. [Open Invention Network](https://openinventionnetwork.com/) and [LoT Network](https://lotnet.com/) are worth mentioning here, which have many members and complement each other. [↑](#footnote-ref-27)
29. To simplify we will use the term “trademark” for both. [↑](#footnote-ref-28)
30. If a trademark has been used so broadly the average consumer does not connect it with a specific brand/origin anymore, we call that a dilution of a trademark and it loses its force. In Slovenia the words “superga”, “edigs”, and “selotejp” are now generic terms, similar to “kleenex” or “sellotape” in English. [↑](#footnote-ref-29)
31. In the US databases receive very limited copyright protection, and broadly speaking in practice the two solutions play out somewhat similarly. [↑](#footnote-ref-30)
32. In certain jurisdictions like the US and the UK it is possible for an author to dedicate their work to public domain, waiving their copyright. But in the majority of the world (incl. continental Europe) that is not possible due to inalienable moral rights. As a work-around licenses like CC0-1.0 and Unlicense can be used. [↑](#footnote-ref-31)
33. The English translations of §49.b and §50.a titles are mine, as I could not find an (un)official English translation of them yet. [↑](#footnote-ref-32)
34. Copyright and Related Rights Act (Slov. *Zakon o avtorski in sorodnih pravicah*). [↑](#footnote-ref-33)
35. Such as CC0-1.0, Unlicense, MIT, BSD-2-Clause, BSD-3-Clause, or Apache-2.0. If there are still active patents, Apache-2.0 is a good choice (and avoid CC0-1.0 in that case). [↑](#footnote-ref-34)
36. The Urban Planning Institute of the Republic of Slovenia is one of the leading institutions in Slovenia and ex-Yugoslavia regarding research, implementation, and consulting within urban and regional planning. It was established in 1955, and is highly connected with similar institutions in Europe and the USA. [↑](#footnote-ref-35)
37. The methodology used for this part of VSD is snowball effect interviews with the starting point being Rudi Bric, founder of HERMES SoftLab (HSL), to help retroactively analyze the possible existence and usage of values in the HSL software development process. [↑](#footnote-ref-36)
38. These practices in HSL were Friday morning presentations where employees were given information on the company’s current workings, what was going well and what was not, along with future steps, and an open door policy where everyone could come in with their questions. [↑](#footnote-ref-37)
39. As noted by Bric, the role of profit in Packard’s HP Way was based on profit being the best single measure of HP contribution to society and source of HP corporate strength, which sounds very “socialistic” in today’s American value set, but was at that time very similar to HSL’s worldview. [↑](#footnote-ref-38)
40. At that time the focus was on the added value of the product on the market (i.e. the product solving a problem), not the added value of the product for the user. [↑](#footnote-ref-39)