LIST OF FOREWORDS

Efanov S. Unexpected meeting. How I got acquainted with DRAKON……………………..
Nadišauskienė R. J., Dobožinskas P., Jasinskas N., Kumpaitienė B., Kliučinskas M., Kudrevičienė A., Pukenytė E. Dambrauskas Z., Vileikytė A., Vaitkaitis D. DRAKON language in medical care and education as the crucial part of HybridLab® method…………………………………….
Garipov O. Taking a glance at real programming code through the lens of DRAKON language……………………………………………………………
Valevich B. About DRAKON editors by Stepan Mitkin………………………………………
Mitkin S. I want to program in DRAKON. Is that possible?..

UNEXPECTED MEETING.
HOW I GOT ACQUAINTED WITH DRAKON

Sergey Efanov, Sole proprietor software and electronic developer
efanov@lipetsk.ru
Lipetsk, Russian Federation

ONCE IN A TRAIN.
ERROR WAS NOT FATAL

Some time ago, I caught sight of the DRAKON language mentioning. I read a little, understood nothing, closed it, forgot it.

But for some reason it was not completely forgotten.

Then the trip happened in. I found it again, uploaded the file to an e-book, took it with me. In a train there is all the same nothing to do.

Slowly and with taste have read.

And — I have understood! It's just a treasure!

On my return I was only thinking about one thing: where to find a tool for work?

Fortunately, such a tool has been found. [1]

I tried a little example — as if some code is generated. Rewrote on the DRAKON a rather confusing function from a real project.

The function has started working right away! Moreover, when I transferred the algorithm into the DRAKON graphical schematics, I have found that it contained an error. This function had been working for a long time, not in one hundred of products. The error was not fatal, it was rare, and was compensated by reconnecting to the server. But it was!

In the C language text, it was not visible. But when trying to transfer the algorithm to the DRAKON drawing, the error has become not just noticeable — the errored algorithm in this place was impossible to draw up!

MUST HAVE VISUAL ALGORITHM

Since then, almost six years have passed. I program only in DRAKON.

Let me say a few words about what this gave me, and what the process looks like.

The writing of the program broke up into two stages — the development of the algorithm and then the actual programming.

The main thing in almost any program is an algorithm. In the DRAKON, it is drawn, more precisely — it is built up of the graphic elements. Very similar to the elements of flow-charts.

But there are a few strict rules that do not allow the schematics to turn up into an intricate tangle of lines, squares and rhombs.

The rules, at first glance, are simple. But the effect of their application is colossal!

Using DRAKON, you cannot draw an intricate and incomprehensible algorithm. Conversely, any complex algorithm, drawn according to these rules, becomes clear and intelligible.

When developing the algorithm, now it does not have to be kept in mind while writing the program text. The work is reduced to the visual construction of the algorithm. This is much easier. There is not required such concentration as usual. Since the drawn algorithm is very clear — the work can be safely interrupted at any time. And then it's easy to return to its continuation.

FROM THE ALGORITHM TO THE PROGRAM

And only when the whole algorithm is “licked all over” — we go to the actual programming. What does it mean now? It means that for each icon you need to write a code executing what is written inside this icon. Usually this is only one line. At high levels of the project hierarchy, it can be a call to one function, or one class method. (Note that all functions and classes are also drawn in DRAKON). At the lower level, this can be a single bit change.

In this place, for sure, many will break out the question: "Well, what for is all this kitchen-garden, if the code all the same you need to write yourself?"

Do not hurry!

What is the complexity of programming? (With the fact that programming is a difficult job, I think no one will argue).

Maybe, is it in writing of lines of type as printf ("Hello, World")?

Will the program become complicated because we write a thousand such lines? And ten thousand?

No, it will not become complicated. A program becomes complex thanks to complex interrelations between its parts.

So, at the stage of icons’ programming there is no need not to think about this longer.

Absolutely. At all. No way. No need, and that's it!

All you need is to carefully program ONE icon. Only one! When we program another — about the previous one you can already forget. There is also no need to look at the code generated by the editor. Just like we do not look at the machine codes obtained after compilation.

Programming at this stage has turned into a purely technical procedure. Uncomplicated.

I had already ceased to be surprised that programs work *right* after switching on ...

DRAKON AGAINST BUGS AND ERRORS

DRAKON has an interesting property. It really minimizes software errors. And this is not accidental thanks to specially provided ergonomic and mathematic measures.

Is it possible to help programmers and propose them DRAKON not as a silver bullet, but as a means of preventing errors? I think in many cases, though not always, yes, it is just the case.

What is needed for this? On my opinion, International standard organization (ISO) can play crucial role. Judge for yourself, drakon-charts are nothing more than improved flowcharts. Not only improved, but well-done, correctly constructed, mathematically strict and people-friendly flowcharts. From the very beginning drakon-charts were planned and realized to ***replace*** flowcharts fully and completely. Consequently, instead of ancient flowchart standard ISO 5807:1985 there should be adopted the golden drakon-chart standard.

I convinced that such a standard will be an effective means capable to noticeably reduce the “world evil” of software errors in the field of imperative and procedural programming. And, maybe, not only.

LACK OF GOOD NOTATIONS FOR ALGORITHMS AND WORKFLOWS

A common ISO motto declares: “Say what you do and do what you say.” The paradox is that people have no effective notations helping to write *what they do*. The real challenge is to describe “doing” processes with high precision and easy to understand form. Lack of such notations lead to inaccuracy when writing workflows, how-to's and algorithms. As a result, mass errors are generated.

That is why rude and awkward classic flowcharts inevitably produce errors. Opposite, the elegant and rigorous drakon-charts are free of such flaws.

MYSTERY OF FLOWCHARTS VIABILITY

The starry hour of software flowcharts was in the middle of the last century at the dawn of the computer era when Herman Goldstine, John von Neumann and later IBM put them on the podium of fame. But in the 1970s everything has changed. Structured programming dealt a crushing blow to the flowcharts and permanently expelled them from the program documentation. The famous flowcharts' enemy and persecutor Frederick Brooks in his angry essay "The Flow-Chart Curse" called them as "obsolete nuisance," and lamented:

> "The Apostle Peter said of new Gentile converts and the Jewish law, "Why lay a load on [their] backs which neither our ancestors nor we ourselves were able to carry?"… I would say the same about new programmers and the obsolete practice of flow charting". [2]

Despite all the attempts to excommunicate the flowcharts from the software church they miraculously survived as a phoenix bird and are still used in program design, learning. And some other areas outside software empire. Why?

This is a difficult puzzle. Computer scientists do not have a unified position. Which is better: textual programming or visual one? There is no end to disputes.

What do you prefer? Pseudocode or flowcharts? Striving to please both ours and yours, Lesley Anne Robertson in her program design textbook recommends pseudocode. And right here she describes in detail the flowcharts as an alternative in the app. [3]

How to solve this mystery? How to explain the huge ability of flowcharts to survive, despite all the critical volleys and the incessant hail of ridicule? You can answer by the words of the same Brooks: "The crying need is for a clear, sharp overview." [4] Really, there is a crying need for handy and clear visual representation of complicated algorithms.

HUGE FORCE OF STANDARDS

I think the "main secret" of flowcharts is simple. Flowcharts have inner latent potential (not yet fully implemented) which can satisfy the need of a human eye to distinctly see a clear and attractive two-dimensional algorithmic and workflow pictures.

World practice confirms this. And the best proof is the international standard ISO 5807:1985. Note, this is not one standard, but an extensive system of many standards. All the industrially developed and many developing countries adopted this standard and localized it as a national one. Here are just three examples from a few dozen:

- British flowchart standard **BS 4058:1987** (confirmed 1999). [5]
- Malaysian flowchart standard **MS 1140:1989** (confirmed 2007). [6]
- Russian flowchart standard **ГОСТ 19.701-90** (confirmed 2005). [7]

Thus, all the giant power of the world standardization system is mobilized to protect and maintain the intellectual authority, exceptional importance and utility of flowcharts.

Here we are neared to the most interesting and the crucial point. All this mighty pyramid of worldwide flowcharts standards is hopelessly obsolete and must be dismantled. Flowcharts are needed serious improvement and modernization to turn into DRAKON. Standards must in every possible way promote this.

ON THE SHOULDERS OF GIANTS

One of such giants is Edsger Dijkstra, the inspirer and motor of structured programming. In his influential article “Go To Statement Considered Harmful” (1968), Dijkstra has argued that the programming statement GOTO, found in many high-level programming languages, is a major source of errors, and should therefore be forbidden. [8] It was the beginning of the great battle with “world evil” of software bugs.

Forty years later (2009), Bertrand Meyer continued the battle when he exposed the next enemy — BREAK statement, which should also be prohibited as a disguised GOTO in sheep's clothing. [9]

Is it possible to continue this holy war? Is it possible to find and detect other suspicious and dangerous programming statements that are the source of errors?

This is the noble and carefully thought-out objective pursued by DRAKON. DRAKON seeks to find out and identify potentially dangerous operators (statements) and even separate programming keywords. And carefully replace them with well-proven safe means.

DRAKON methodology allows you to completely reject the impressive army of software keywords. Nobody needs them. These are parasitic keywords that can be painlessly removed. There should be disappeared not only the disgraced programming statements: ***goto*** and ***break***, but also the respectable symbols of true faith: ***if, then, else, case, switch, while, do, repeat, until, for, foreach, loop, exit, when, last, continue,*** etc., as potentially dangerous ones.

The DRAKON goal is to replace them with mathematically rigorous control graphics, implementing exactly the same function as eliminated “bad” keywords. It means to use two-dimensional (2d) structured programming. The last one you may consider as further development of one-dimensional (1d) structured programming, created by excellent scientists of the past: Edsger Dijkstra, Robert Floyd, Tony Hoare, Ole-Johan Dahl.

The negative role of the listed keywords (which are working gears of modern programming) is that they spoil and litter the visual picture, create unneeded visual interference, provoke the appearance of annoying errors and complicate understanding the meaning of the algorithm in the terms of the subject area.

HIDDEN PROPHECY OF FREDERICK BROOKS

DRAKON is a new idea. It is my working tool for visual programming and I like it. I’m sole proprietor and DRAKON helps me to earn for my living.

Nowadays textual programming reigns undividedly in almost all state-of-the-art integrated development environments (IDE). But, I hope, it will not always be so. Visual programming is slowly but surely gaining momentum.

It seems to me that sooner or later much may change, and much will change. I believe that visual approach sooner or later will triumph.

The time will come when the current apostles of the software mainstream will be forced to repeat after Frederick Brooks: "By and large we have failed. I think we have used wrong methods." [10]

Let us wait and let us see.

BIBLIOGRAPHY

[1] DRAKON constructor by Gennady Tyshov. Software for making DRAKON charts and ascetic IDE without several important options.

https://cloud.mail.ru/public/ecbde70c784a/%D0%98%D0%A1%20%D0%94%D1%80%D0%B0%D0%BA%D0%BE%D0%BD.

[2] The Mythical Man-Month. Essays on Software Engineering. Frederick P. Brooks, Jr. Addison-Wesley Publishing Company, Inc., 1975. — 195 p. — p. 169.

[3] Robertson L.A. Simple program design: A step-by-step approach. Fifth edition. — Cengage Learning, 2006. — 368 p.

[4] The Mythical Man-Month. Essays on Software Engineering. Frederick P. Brooks, Jr. Addison-Wesley Publishing Company, Inc., 1975. — 195 p. — p. 166.

[5] British Standard BS 4058:1987. Specification for Data processing flow chart symbols, rules and conventions — [ISO title:….

[6] Malaysian standard MS 1140:1989. Specification for information processing, flowchart symbols, rules and conventions. Department of Standards Malaysia, 2007.

[7] ГОСТ 19.701-90. Схемы алгоритмов, программ, данных и систем. Data, program and system flowcharts, program network charts and system resources charts. Documentation symbols and conventions for flowcharting..

[8] Dijkstra E. Go To Statement Considered Harmful // Communications of the ACM, Volume 11, No. 3, March 1968, pp. 147-148. — Association for Computing Machinery, Inc..

[9] Meyer B. Touch of class. Learning to program well with objects and contracts. — Springer Verlag Berlin Heidelberg, 2009. — LXIV, 876. — Chapter 7. Section 7.8.

[10] The Mythical Man-Month. Essays on Software Engineering. Frederick P. Brooks, Jr. Addison-Wesley Publishing Company, Inc., 1975. — 195 p. — p. 164.

DRAKON language in medical care and education as the crucial part of HybridLab® method

R. J. Nadišauskienė[1], P. Dobožinskas[1,2], N.Jasinskas[1,2], B. Kumpaitienė[1,2], M. Kliučinskas[1,2], A. Kudrevičienė[1,2], E. Pukenytė[1,2], Ž. Dambrauskas[1,2], A. Vileikytė[2], D. Vaitkaitis[1,2].

Lithuanian University of Health Sciences[1], Crises Research Centre[2]

"If one only hears – one shall forget, if one sees – one shall remember, if one performs – one shall understand".
The old Chinese proverb

World Health Organization (WHO) indicates that in developed countries: 1 in 10 patients is harmed while receiving hospital care. Industries such as aviation and nuclear plants have much better safety record than health care [1]. Why health care system is many times less safe? It can be due to much higher complexity in health care, due to lack of regular training or lack of time to fulfill tasks. At the same time different culture and attitude in medicine and aviation plays an important role. The main problem of medical care is the lack of standardization of the procedures, insufficient culture of infallibility, the lack of effective method of counteraction to errors that adversely affects and harms patients.

Now the patient safety comes to the fore. In modern health care, only interdisciplinary team work can lead to success especially in emergent situations. Teaching hospitals and health systems are making the transition from the old culture of autonomy and independence to the new world of shared accountability, interdependence and team work. Patients safety training cannot be done in a day and consist of one-time lecture. There should be algorithms and checklists to reduce failure for limits of human memory and attention. Well-designed algorithms help to ensure patient safety and also consistency and completeness in carrying out a task.

Throughout 1980's and 1990's guidelines of all sorts were created in medicine. However, multiple studies demonstrated that the guidelines have limited impact of on physicians' decision making [2]. Diagnoses and treatment modalities differ all around the globe and some could lead to a variability of implementations and result in multiple outcomes.

Well-designed medical algorithms could standardize and improve the path of diagnosis and treatment [3]. Studies suggest that algorithms improve the health care system and allow achieve a higher degree of standardization across patient management. Accurately applied algorithms based on the best available evidence lead to a more efficient treatment and better outcomes [4, 5]. The vast volumes of literature on algorithms provide very little information on how to produce algorithms, especially ergonomic ones. Most of the resources simply include examples of algorithms without the unifying system. The quality of medical algorithm is based not only on how concise and inclusive it is. It is also substantial that the sequence of algorithm steps is logical and the terminology clear [6]. Obviously, one of the most important aspects of an algorithm is its efficiency [7].

For the first time we have used DRAKON algorithmic language to describe the path of processes and procedures in emergency obstetrics and neonatology in the national guidelines (Fig.1) [8]. It was well accepted by clinical doctors, midwives and nurses.

The DRAKON algorithms are widely used in the undergraduate and postgraduate education at the Lithuanian University of Health Sciences (LUHS). DRAKON is an essential part of new HybridLab® learning method, invented by Crises Research Centre (CRC) researchers.

CRC created a peer-teaching model that included e-learning modules combined with online checklist driven skills and decision DRAKON flowcharts — HybridLab®. The methodology stresses success-based learning where peer-teachers used checklists to assist the learner down the correct pathway. Learners build automaticity in their skill responses, practicing until competent. Remote faculty provide additional feedback for both learners and peer-teachers after viewing video performances of skills. The goal in HybridLab® learning based on mastery-learning concepts is to ensure that all learners accomplish all educational objectives with little or no variation in outcome.

The HybridLab® method was used to teach 5th year students in obstetrics and gynecology rotation. The method includes gynecological patient investigation skills (history collection, breast examination, gynecological examination, Pap smear taking). Besides, there are used obstetrical patient investigation (fetal position, presenting part, symphysis-fundal height, and heart rate), initial neonatal resuscitation (mouth-to-mouth ventilation, and chest compressions). We evaluated the retention of clinical skills over time with or without the use of a checklist reminder. Impressively good results of the acquisition in retention of practical skills were received [9, 10].

The number of DRAKON algorithms have been elaborated for postgraduate education programmes. These algorithms cover basic and advanced life support, basic and advanced obstetrical emergencies and neonatal resuscitation skills. These programmes have been successfully implemented in the national (Lithuania) and international projects (e. g. Kazakhstan).

Three medical DRAKON algorithms are described below, each one being neatly divided into structured parts, which are called as “branches”.

ACTIVE MANAGEMENT OF THIRD STAGE OF LABOUR

The birth of a new person is an important and exciting event. Childbirth is also known as labour and delivery. "Third stage of labour" is a medical term to signify the period from just after the child is born until just after the placenta is expelled.

Placental expulsion begins as a physiological separation from the wall of the uterus. The placenta is usually expelled within 15–30 minutes of birth.

It is much better to actively manage placental expulsion, for example by giving oxytocin via intramuscular or intravenous injection followed by cord traction to assist in delivering the placenta.

Let us see the algorithm describing delivery of the placenta and how to assess the postpartum blood loss (Fig. 1, 2). Active management during third labour stage is recommended in all deliveries. It is strongly proven that active management decreases amount of blood loss, frequency of postpartum hemorrhage and therefore risk for anemia and blood transfusion is minimized.

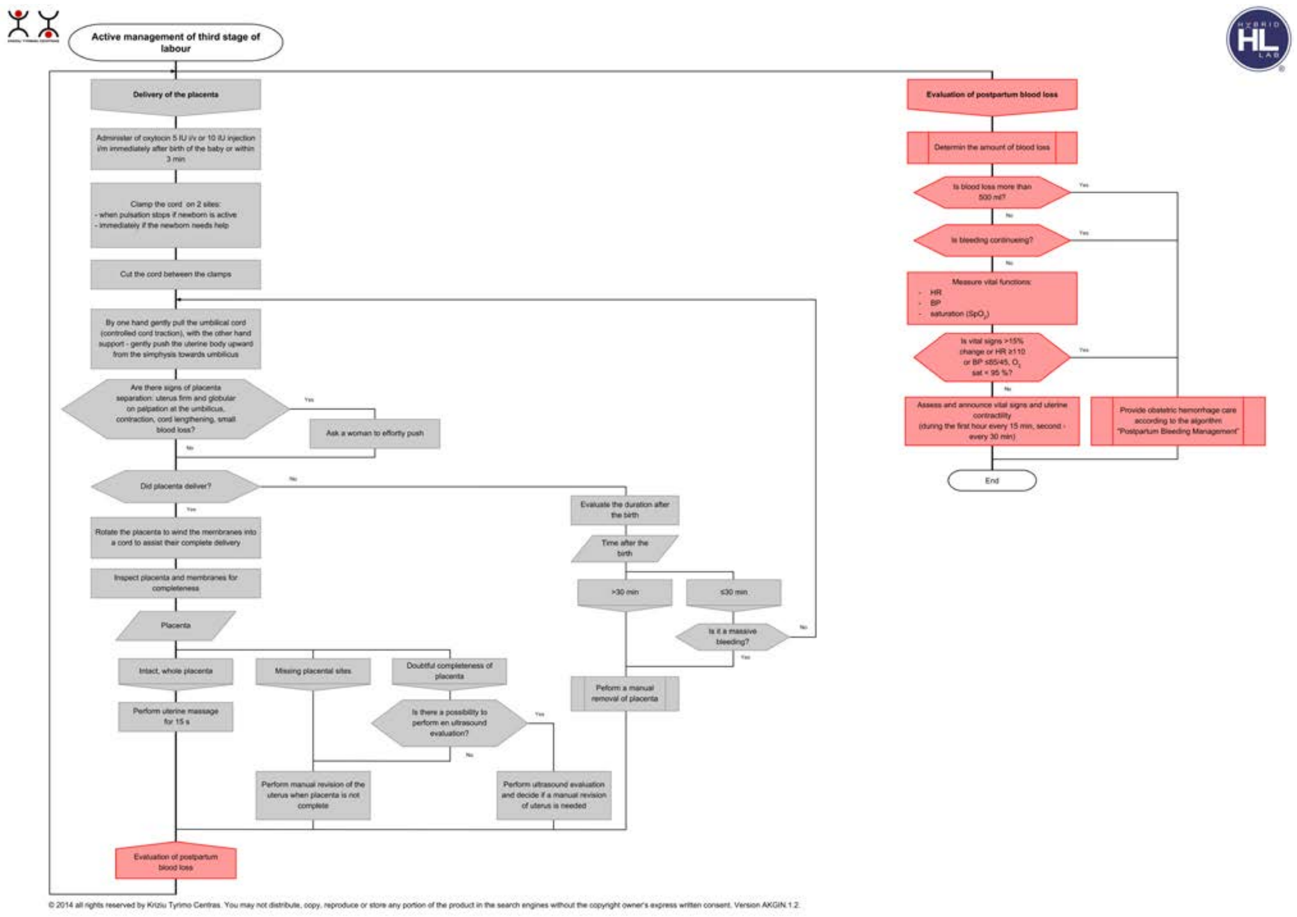
Each medical DRAKON algorithm has only one beginning and only one end. The beginning in Fig. 1 is located in the oval icon (on the upper left) where is written: “Active management of third…” All the paths of the algorithm go from this point to the End icon (on the lower right). These paths are described in detail below.

This algorithm has two branches:

- Delivery of the placenta.
- Evaluation of postpartum blood loss (Fig. 1).

The name of the branch shows the phase that will be completed after all the actions in the branch are done.

Branches in the algorithm are read from left to right, and the actions in each branch are performed from up to down.



Active management of third stage of labour
Delivery of the placenta
Administer of oxytocin 5 IU i/v or 10 IU injection i/m immediately after birth of the baby or within 3 min
Clamp the cord on 2 sites:
- when pulsation stops if newborn is active
- immediately if the newborn needs help
Cut the cord between the clamps
By one hand gently pull the umbilical cord (controlled cord traction), with the other hand support - gently push the uterine body upward from the simphysis towards umbilicus
Are there signs of placenta separation: uterus firm and globular on palpation at the umbilicus, contraction, cord lengthening, small blood loss?
Yes
Ask a woman to effortly push
No
Did placenta deliver?
No
Yes
Rotate the placenta to wind the membranes into a cord to assist their complete delivery
Inspect placenta and membranes for completeness
Placenta
Intact, whole placenta
Missing placental sites
Doubtful completeness of placenta
Perform uterine massage for 15 s
Is there a possibility to perform en ultrasound evaluation?
Yes
No
Perform manual revision of the uterus when placenta is not complete
Perform ultrasound evaluation and decide if a manual revision of uterus is needed
Evaluate the duration after the birth
Time after the birth
>30 min
≤30 min
Is it a massive bleeding?
No
Yes
Peform a manual removal of placenta
Evaluation of postpartum blood loss
Evaluation of postpartum blood loss
Determin the amount of blood loss
Is blood loss more than 500 ml?
Yes
No
Is bleeding continueing?
Yes
No
Measure vital functions:
- HR
- BP
- saturation (SpO2)
Is vital signs >15% change or HR ≥110 or BP ≤85/45, O2 sat < 95 %?
Yes
No
Assess and announce vital signs and uterine contractility (during the first hour every 15 min, second - every 30 min)
Provide obstetric hemorrhage care according to the algorithm "Postpartum Bleeding Management"
End
© 2014 all rights reserved by Kriziu Tyrimo Centras. You may not distribute, copy, reproduce or store any portion of the product in the search engines without the copyright owner's express written consent. Version AKGIN.1.2.

Fig. 1 Medical algorithm “Active management of third stage of labour”

“DELIVERY OF THE PLACENTA“ BRANCH

The first Action icon (rectangle) shows when and what amount of intravenous or intramuscular *Oxytocin* should be administered to a patient. The medical care specialist reads this instruction and executes it (Fig. 1).

After completing it, the midwife[1] reads the next instruction inside the following Action icon (Fig. 1). It indicates to compress immediately the cord in two places if the baby needs help, or after the cord stops pulsating if the baby is active. The next step is to cut the cord between the clamps, followed by instruction to perform controlled cord traction. The midwife gently pulls the cord with one hand and pushes the uterus away from the symphysis towards umbilicus with the other hand.

A little bit lower there is a Question icon which has the form of a hexagon. It contains a question: "Are there signs of placental separation"? This means midwife must assess: Are there relevant signs?

If there are signs of placental separation, follow the "Yes" path. The midwife implements the instruction inside the Action icon and “asks a woman to effortly push”.

Otherwise, follow the "No" path. The midwife reads the next question: "Is placenta delivered?" The health care specialist has to evaluate if the placenta is out.

[1] The below description refers not only to midwives, but also to physicians.

If the placenta is not delivered, we go out from the Question icon to the right through "No" exit. After that the midwife reads the instruction: “Evaluate the duration after the birth”. The Choice icon (below) "Time after the birth” gives two possible time periods:

- ● If more than 30 minutes passed since birth, the midwife performs the command written in the Insertion icon: "Remove placenta manually". (The actions inside Insertion are described in detail in other algorithm). After manual removal of the placenta, we move downstairs through Address icon “Evaluation of postpartum blood loss” (Fig. 1). The Address icon is named just so because it contains the destination address. When we go into Address, we immediately jump upstairs to the very beginning of the branch with the same name (“Evaluation of postpartum blood loss”).
- ● If less than 30 min have passed since birth, the midwife reads the question: "Is there a massive bleeding"? This is an instruction for the midwife to assess the bleeding. If there is no massive bleeding, follow the "No" path. Return to the left along the arrow and repeat the actions, starting from “By one hand gently pull the umbilical cord…” If the answer is "Yes", the midwife is instructed to manually remove the placenta.

Let’s return once more to Question icon “Did placenta deliver?” If the answer is “Yes”, midwife must implement two instructions to ease delivery (Fig. 1):

— “Rotate the placenta to wind the membranes into a cord to assist their complete delivery”.

— “Inspect placenta and membranes for completeness”.

The rescuer assesses the integrity of placenta.

The Choice icon (below) says "Placenta" and shows three possible options:

1) If the variant is selected: “Doubtful completeness of placenta”, read the question: “Is there a possibility to perform an ultrasound evaluation”?

- ● If "Yes", the midwife must: “Perform ultrasound evaluation and decide if a manual revision of uterus is needed”.
- ● If "No", the midwife must: “Perform manual revision of the uterus when placenta is not complete”.

2) If the variant is: “Missing placental parts”, the midwife have to “Perform manual revision of the uterus when placenta is not complete”.

3) If we see: “Intact, whole placenta”, the midwife must “Perform uterine massage for 15 seconds”.

After completion of the necessary actions in the first branch, the algorithm go (through the Address icon) to the second branch, which is named: “Evaluation of postpartum blood loss”.

“EVALUATION OF POSTPARTUM BLOOD LOSS” BRANCH

The health care specialist must implement the instruction in the Insertion icon: “Determine the amount of blood loss” (Fig. 1). *(The double-sided rectangle shows that the necessary actions are described in detail in another algorithm).* The midwife assesses the situation and answers two question:

— “Is blood loss more than 500 ml?”

— “Is bleeding continuing?”

If at least in one case the answer is "Yes", the midwife must provide obstetric hemorrhage care according to the procedure "Postpartum Bleeding Management". In short, she starts to execute the algorithm "Postpartum…".

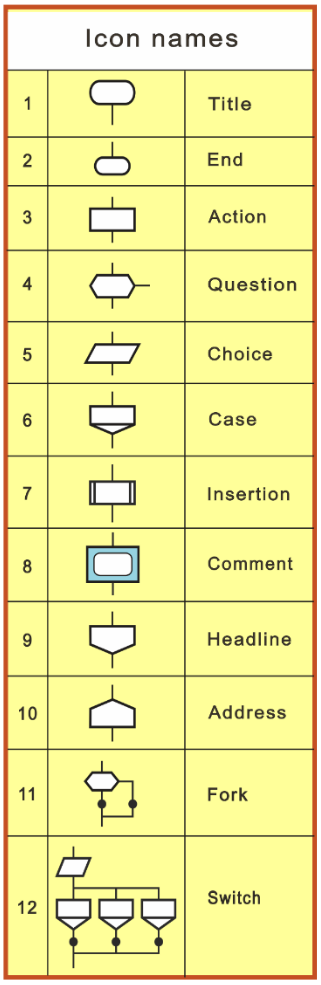
If in both cases the answer is "No" (this is good), the midwife must "Measure vital functions". After measuring HR (Heart Rate), BP (Blood Pressure), O_2 Sat (Oxygen Saturation), the health care specialist must answer the question: "Is vital signs >15 % change or HR≥ 110 or BP ≤85/45, O_2 Sat<95%"?

If the answer is "Yes", the midwife must go to the procedure "Postpartum bleeding management".

If the answer is "No" (vital signs are normal), the midwife must assess and announce vital signs and uterine contractility (during the first hour every 15 minutes, the second one — every 30 min).

When the midwife completes the last command, the algorithm is finished.

By using this DRAKON-algorithm, the health care specialists learn the correct "Active management of the third labour stage".



Icon names		
1		Title
2		End
3		Action
4		Question
5		Choice
6		Case
7		Insertion
8		Comment
9		Headline
10		Address
11		Fork
12		Switch

Fig. 2 Names of the icons

VENTILATION MOUTH-TO-MOUTH AND NOSE DURING INITIAL NEONATAL RESUSCITATION

This algorithm explains clearly and in detail how to perform effective ventilation to a non-breathing neonate (Fig. 3). It is intended for medical and non-medical persons, when ventilation equipment is unavailable.

Neonatal mouth-to-mouth and mouth-to-nose ventilation consists of three activities:

- Preparation for ventilation.
- Assessment of efficiency of ventilation and correction.
- Effective ventilation.

Therefore, there are three branches in the algorithm. The name of each branch shows one of three major activities. Each branch contains a number of graphic elements that show the actions the resuscitator must perform.

The branches should be read from left to right. The resuscitator performs actions written in the graphic elements of the branch by reading them from up to down.

"PREPARATION FOR VENTILATION" BRANCH

The first instruction in the rectangular Action icon says: "Position the neonate on his back". Having read it the resuscitator must lay the baby in supine position (Fig. 3).

The next instruction declare: "Stand on the one side of the neonate". Executing it, the resuscitator takes a position at the side of the baby.

Let's make the next step and read: "Place the roller (thickness 2 cm) under the shoulders". The rescuer performs the command — puts a 2-cm roller, made from blanket, under the neonate's shoulders.

Further we see the Question icon: "Are there secretions in the mouth and nose?"

If the answer is "Yes", the resuscitator must "Clean the mouth, then nose using a small piece of tissue" (Fig. 3). It means to clear the secretions out of the mouth and nose with any kind of cloth.

If the answer is “No” (there are no secretions), the resuscitator performs the command "Cover neonate’s mouth and nose with your mouth”.

After this icon, there is an Comment icon with advice to the resuscitator: "If you are unable to mouth up mouth and nose, mouth up the neonate's mouth only".

The first branch ends with an Address icon (pentagon) which shows the name (address) of the next branch: “Assessment of efficiency…” The Address icon means “goto” instruction: jump up to the beginning of the corresponding branch.

“ASSESSMENT OF EFFICIENCY OF VENTILATION AND CORRECTION“ BRANCH

The resuscitator reads the actions in the second branch from top down and performs them (Fig. 3).

The first Action icon commands: “Inflate as much air for the neonate as you hold in your mouth.” The rescuer must inhale air to the neonate's lungs.

After this follows the Comment icon: “If you are covering the neonate’s mouth, press up the neonate’s nose with your two fingers.” What for? If the baby’s nostrils are not covered by the mouth, the resuscitator has to pinch the neonate's nose with two fingers during inhalation.

A little lower there is a Question icon: "Does the chest rise with ventilation?" This means that while giving breaths, the resuscitator has to evaluate concurrently if the neonate's chest rises, i.e. the mouth-to-mouth operation is effective.

If the chest rises, follow the "Yes" path (Fig. 3). And then through the Address icon go up to the next branch.

If the chest does not rise, follow the "No" path. It shows three instructions (three Action icons) that must be performed so that the chest become raise. Here they are:

- “Correct position of the roller” *(under neonate's shoulders).*
- “Clean the mouth, then nose using a small piece of tissue”.
- “Inflate more air”.

The rescuer performs the instructions, clears the mucus out of the mouth and nose and blows more air into the neonate's lungs when giving breaths.

Then s/he reads the question: “Does the chest rise while inflating?”

If the answer is "No", the path go up along the arrow and the three instructions in the Action icons ("Correct…”, “Clean…”, “Inflate…”) are repeated by rescuer until the neonate' chest begins to rise and fall with ventilation.

If the answer is "Yes", the second branch ends with Address icon, which tells to go to the third branch.

“EFFECTIVE VENTILATION” BRANCH

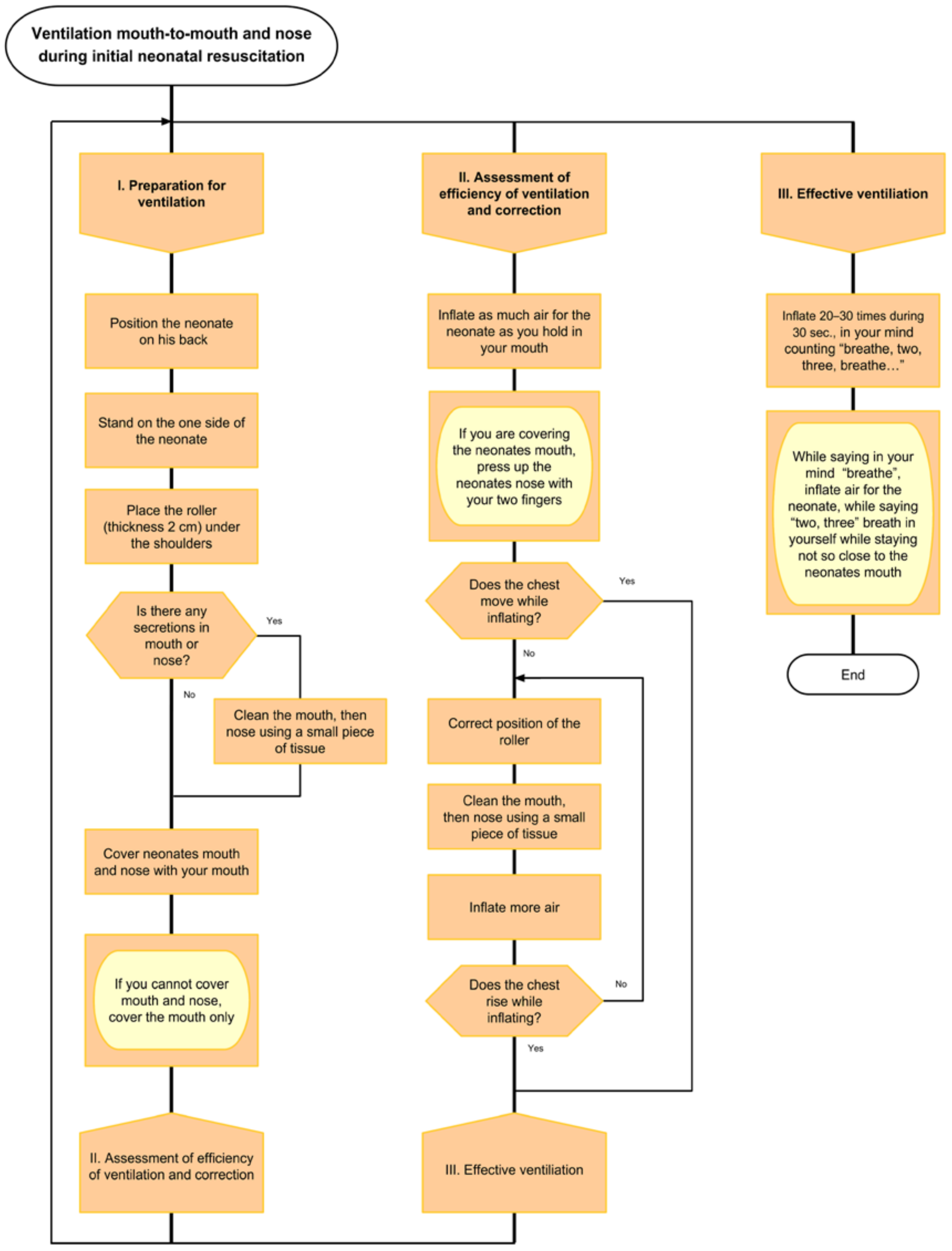
The resuscitator reads the instructions and performs them.

The first instruction tells: “Inflate 20–30 times during 30 seconds, in your mind counting ‘breathe, two, three, breathe’…” (Fig. 3).

After that follows the Comment icon which explains in detail how to give breaths: “While saying in your mind ‘breathe’, inflate air for the neonate. While saying ‘two, three’ breath in yourself while staying not so close to the neonate’s mouth”.

It is important that the resuscitator while saying ‘two, three’ and inhaling in himself, must move away a little bit from the neonate's mouth.

Using this algorithm**,** the resuscitator learns to perform effective mouth-to-mouth and mouth-to-nose ventilation correctly.



Ventilation mouth-to-mouth and nose during initial neonatal resuscitation
I. Preparation for ventilation
II. Assessment of efficiency of ventilation and correction
III. Effective ventiliation
Position the neonate on his back
Stand on the one side of the neonate
Place the roller (thickness 2 cm) under the shoulders
Is there any secretions in mouth or nose?
Yes
No
Clean the mouth, then nose using a small piece of tissue
Cover neonates mouth and nose with your mouth
If you cannot cover mouth and nose, cover the mouth only
II. Assessment of efficiency of ventilation and correction
Inflate as much air for the neonate as you hold in your mouth
If you are covering the neonates mouth, press up the neonates nose with your two fingers
Does the chest move while inflating?
Yes
No
Correct position of the roller
Clean the mouth, then nose using a small piece of tissue
Inflate more air
Does the chest rise while inflating?
No
Yes
III. Effective ventiliation
Inflate 20–30 times during 30 sec., in your mind counting "breathe, two, three, breathe…"
While saying in your mind "breathe", inflate air for the neonate, while saying "two, three" breath in yourself while staying not so close to the neonates mouth
End

Fig. 3 Ventilation mouth-to-mouth and nose during initial neonatal resuscitation

CARDIOPULMONARY RESUSCITATION AND DEFIBRILLATION

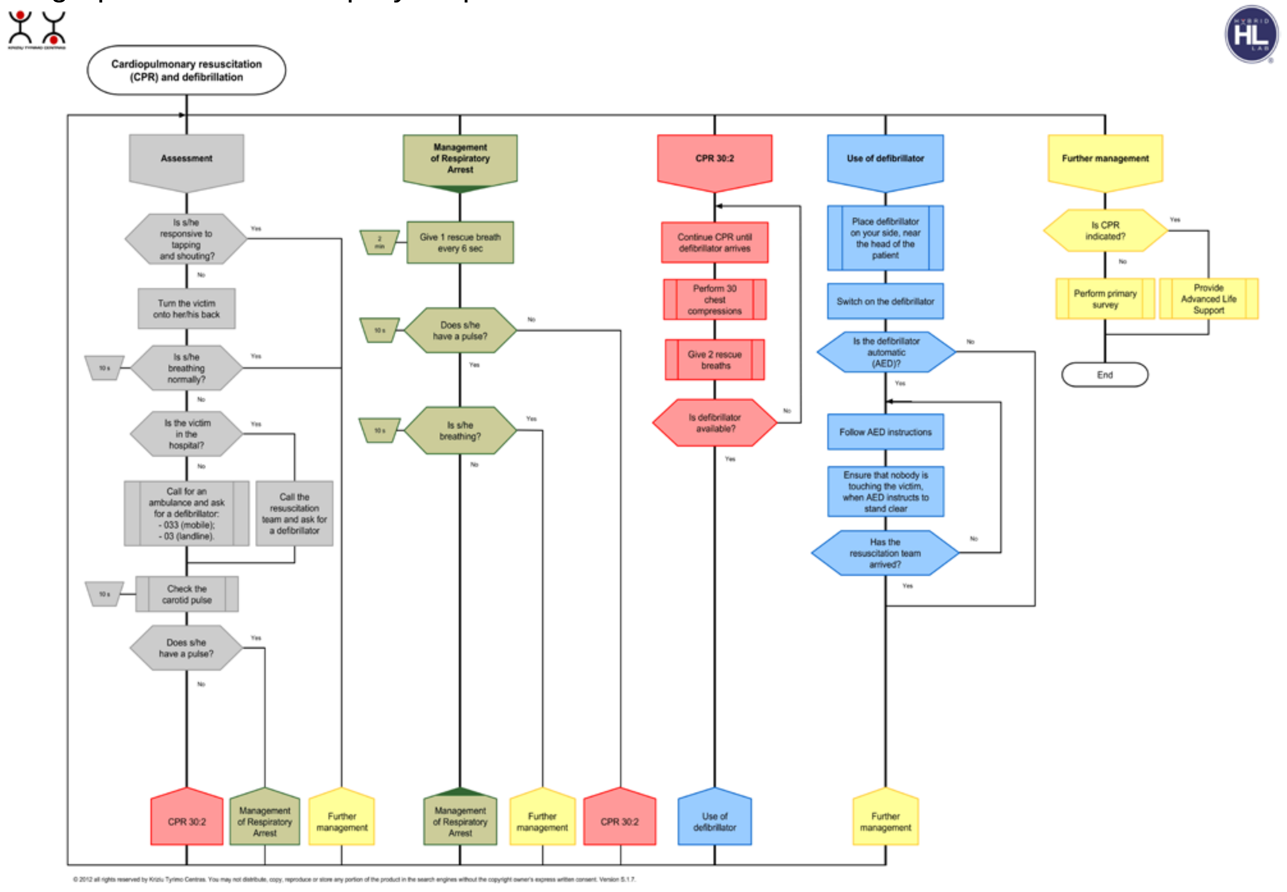
This algorithm is used to teach medical and non-medical specialties to provide cardiopulmonary resuscitation and use automatic external defibrillator (AED) (Fig. 4).

As a skill, cardiopulmonary resuscitation (CPR) can be divided into four branches:

- assessment,
- ventilation,
- chest compression,
- use of defibrillator.

The algorithm also covers the actions in case the patient has respiratory arrest only (pulse is present - CPR is not necessary) and what to do if both respiration and pulse are present, or help arrives during cardiopulmonary resuscitation. Each branch has a number of graphic elements (icons) showing the steps the resuscitator must perform.

The rescuer is reading the algorithm from top down and performs actions described in the graphic elements step-by-step.



Cardiopulmonary resuscitation (CPR) and defibrillation
Assessment
Is s/he responsive to tapping and shouting?
Yes
No
Turn the victim onto her/his back
10 s
Is s/he breathing normally?
Yes
No
Is the victim in the hospital?
Yes
No
Call for an ambulance and ask for a defibrillator: - 033 (mobile); - 03 (landline).
Call the resuscitation team and ask for a defibrillator
10 s
Check the carotid pulse
Does s/he have a pulse?
Yes
No
Management of Respiratory Arrest
2 min
Give 1 rescue breath every 6 sec
10 s
Does s/he have a pulse?
No
Yes
10 s
Is s/he breathing?
Yes
No
CPR 30:2
Continue CPR until defibrillator arrives
Perform 30 chest compressions
Give 2 rescue breaths
Is defibrillator available?
No
Yes
Use of defibrillator
Place defibrillator on your side, near the head of the patient
Switch on the defibrillator
Is the defibrillator automatic (AED)?
No
Yes
Follow AED instructions
Ensure that nobody is touching the victim, when AED instructs to stand clear
Has the resuscitation team arrived?
No
Yes
Further management
Is CPR indicated?
Yes
No
Perform primary survey
Provide Advanced Life Support
End
CPR 30:2
Management of Respiratory Arrest
Further management
Management of Respiratory Arrest
Further management
CPR 30:2
Use of defibrillator
Further management
© 2012 all rights reserved by Krizu Tyrimo Centras. You may not distribute, copy, reproduce or store any portion of the product in the search engines without the copyright owner's express written consent. Version S.1.7.

Fig. 4 Cardiopulmonary resuscitation and defibrillation

"ASSESSMENT" BRANCH

This branch is the first. It represents the beginning of the algorithm which starts with the icon "Assessment". A little bit lower we see a question "Is s/he responsive to tapping and shouting"? (Fig. 4). This means the resuscitator must tap and call the patient. The response determines the fork and describes two different ways through the algorithm.

If the patient responds to tapping and shouting, we go out to the right through "Yes." And then move (through the Address icon) to the extreme right branch "Further management".

If "No", we choose command "Turn the victim onto her/his back". The next icon shows the question "Is s/he breathing normally?" This means the doctor must check breathing. The icon to the left represents time. This is an important hint: the resuscitator must check breathing no longer than time shown, in this case, 10 seconds.

The next question says: “Is the victim in the hospital? If Yes, it is necessary to implement the instruction “Call the resuscitation team and ask for a defibrillator.” And so on. Now we know what help is needed and how to call for help.

The next instruction after calling help is "Check pulse". To the left is shown the time to perform the command — 10 seconds.

Depending on whether the pulse is present or not, you may go onto one of the two different branches: "Management of respiratory…" or "CPR 30:2".

“MANAGEMENT OF RESPIRATORY ARREST” BRANCH

This branch shows the resuscitator’s actions, when the patient does not breathe. The first command declares: "Give 1 rescue breath every 6 seconds" (Fig. 4). To the left of this instruction is a Time icon. It shows how long a rescuer must perform this command: 2 minutes. In short, to give breaths every 6 seconds during 2 min.

A little bit lower is a Question icon "Does s/he have a pulse?" A Time icon is located to the left. It says: we should check the pulse for 10 seconds. Depending on the result (pulse is present or not), we either continue actions on this branch or jump (through the Address icon) to the next branch "CPR 30:2". If the answer is positive (pulse is present), we go to a Question icon "Is s/he breathing?"

If the patient is not breathing, we are going vertically down to the Address icon "Management Respiratory Arrest". This means order to jump upstairs to the beginning of the self-titled branch. And then perform all the branch actions from the start.

If pulse is not present, the path of the algorithm leads (through the Address icon) to the third “CPR 30:2” branch.

“CARDIOPULMONARY RESUSCITATION 30:2” BRANCH

In this branch, we can see instructions describing how to do chest compressions and give breaths (Fig. 4):

- “Continue CPR until defibrillator arrives.”
- “Perform 30 chest compressions.”
- “Give 2 rescue breaths.”

There are two possible paths after the question "Is defibrillator available?"

If a defibrillator is not available, the path leads us to the beginning of this branch along the arrow. And we repeat CPR actions one or many times. If the device is got, the path leads us (through Address icon) to the next branch "Use of defibrillator".

"USE OF DEFIBRILLATOR" BRANCH

Here we see instructions given to the resuscitator in the same manner, one after another. They explain where to put the defibrillator, how to turn it on and what to do next (Fig. 4).

The first instruction (procedure) is “Place defibrillator on your side, near the head of the patient.” The second one is “Switch on the defibrillator.” After that we see the question: “Is the defibrillator automatic (AED)?”

If “No”, we go out downstairs and leave this branch through the Address icon. The last one tells us the destination address. Then we jump to the “Further management” branch.

If “Yes,” the rescuer is implementing the two instructions:

- “Follow AED instructions.”
- “Ensure that nobody is touching the victim, when AED instructs to stand clear.”

Here is the question: “Has the resuscitation team arrived”?

If “No”, the actions are repeated once more. If “Yes,” we go (through the Address icon) to the extreme right branch.

"FURTHER MANAGEMENT" BRANCH

Firstly, the resuscitator has to answer the question: ”Is cardiopulmonary resuscitation indicated”? The answer determines, if the resuscitator should “Perform primary survey” or continue to “Advanced Life Support”.

You can see that some of the icons in the algorithm have double vertical borders (Fig. 4). This means that if the rescuer is unsure how to perform an action, there is an extra procedure (additional algorithm), which shows in detail how to perform actions step-by-step.

Using this algorithm, resuscitators learn the sequence of CPR actions. All the possible scenarios are provided and described in the algorithm:

- patient unconscious, but breathing,
- patient has a pulse, but is not breathing,
- patient is not breathing and does not have a pulse.

The necessary actions are described clearly in these scenarios. The algorithm also shows how much time can be spent performing one action or when to continue to the next step.

ELECTRONIC DATA RECORDS TRAINING IN AMBULANCE SERVICES

We have developed the new self-direct simulation training programme with carefully developed DRAKON algorithms for learning to fill in electronic data cards (EDC). It was implemented in Kaunas Ambulance Service since December, 2015. Immediately after programme completion the teaching course the ambulance staff have started successfully use EDC in the clinical practice.

We also have experience of the advantage of carefully developed DRAKON algorithms in clinical skills training (e.g. intubation, defibrillation, vein cannulation etc.). Our specialists have determined the proper DRAKON pathway of certain specific diagnostic and treatment procedures (e.g. stroke, myocardial infarction, sepsis, postpartum hemorrhage, etc.). Recently we have developed DRAKON language algorithms for learning non - technical skills.

TESTIMONIALS

There are testimonials of health care specialists on implementation of DRAKON language in medical care as well as in education.

Aistė Vileikytė, project coordinator at the Crises Research Centre:

> “DRAKON language ensures standardized and ergonomic approach even to the most complex procedures. Verbal communication of professionals often doesn’t reveal any discrepancies in their opinions and the way they perform procedures. Every time the professionals try to document their expertise, a number of

disagreements and discussions emerge. The correct DRAKON recording of procedures guarantees successful performance.

Sequence of actions is extremely important in most cases. If procedures are written according to the DRAKON rules, you cannot skip any action and must act consistently.

Every step plays an important role in well-crafted algorithm. No action goes without saying and should be written down in the algorithm. For instance, 'wash your hands', 'prepare your workplace', etc. should be noted before every procedure. Such consistency ensures correct and successful performance from the very first time" (Fig. 5).

Aušrelė Kudrevičienė MD, PhD, neonatologist, Associate Professor:

"DRAKON language is an amazing instrument, which helps to teach practical skills and standardize them. It allows to reveal even the smallest yet very important actions. DRAKON helps to understand, perform actions and form skills even if a person is not medical care specialist."

Birutė Kumpaitienė, MD, anesthesiologist–reanimatologist:

"Writing algorithms in DRAKON language is useful for both: the creating person and the learner. DRAKON helps to creator to standardize every skill and procedure execution. Learners using DRAKON algorithms gets clear orders and path how to execute skill or what decisions to make in every situation" (Fig. 6).

Žilvinas Dambrauskas MD, PhD. surgeon, Professor:

"For me personally, the biggest advantage of the DRAKON language is that it breaks down the complex procedures and processes into stages (branches) and describes each stage in a detailed step-by-step manner. This allows you to train medical decision making and/or clinical skills in the fashion similar to that of the professional athletes. One can easily create a mental description of the process or procedure and can practice it in the head multiple times before the real-life performance. Currently there is enough evidence that this type of mental training is a valuable tool to boost the performance of elite athletes and health care specialist alike".

SUMMARY

The importance of clinical algorithms in the patient diagnosis, treatment, and intervention is obvious. Similarly, algorithms are also useful in education of medical practitioners, especially while working as a team. In order to maximize the utility of algorithms, they need to be simple, inclusive, and ergonomic, so that every user executes a specific algorithm in the identical manner.

Medical care requires an instant implementation of latest scientific achievements. DRAKON algorithm can be used for practical "skills and drills" of undergraduate and postgraduate students. The DRAKON flowcharts standardize, ensure quality of diagnostic and treatment procedures. It minimizes the possibility of error, thus, increases patients' safety.

Fig. 5 Emergency doctors are checking the algorithm of the patient intubation

Fig. 6 Anesthesiologist-intensive care doctor is assessing an algorithm

REFERENCES

1. http://www.who.int/features/factfiles/patient_safety/en/index.html
2. Keffer JH (2001) Guidelines and algorithms: Perceptions of why and when they are successful and how to improve them. Clinical Chemistry 47(8): 1563-1572.
3. Hussey PS, Timbie JW, Burgette LF, Wenger NS, Nyweide DJ, et al. (2015) Appropriateness of advanced diagnostic imaging ordering before and after implementation of clinical decision support systems. JAMA 313(21): 2181-2182.
4. Schoenbaum SC, Gottlieb LK (1990) Algorithm based improvement of clinical quality. Br Med J 301(6765): 1374-1376.
5. Trivedi MH, Claassen CA, Grannemann BD, Kashner TM, Carmody TJ, et al. (2007) Assessing physicians' use of treatment algorithms: Project IMPACTS study design and rationale. ContempClin Trials 28(2): 192- 212.
6. Cook R (2005) Clinical algorithms and flow charts as representations of guideline knowledge. Health care and informatics review online.
7. Mitkin S (2011) DRAKON The human revolution in understanding programs.
8. https://sam.lrv.lt/uploads/sam/documents/files/Veiklos_sritys/Programos_ir_projektai/Sveicarijos_parama/Akuserines%20metodikos/Vaisiaus%20peciu%20uzstrigimas.pdf
9. Kudreviciene A, Nadisauskiene R, Tameliene R, Garcinskiene J, Nedzelskiene I, Tamelis A, Dobozinskas P, Vaitkaitis D, New HybridLab Training Method of Clinical Skills Learning in Neonatology: Students' Opinion, MedEdPublish, 2016, 5, [2], 41, http://www.mededpublish.org/manuscripts/418 doi:https://doi.org/10.15694/mep.2016.000069
10. Nadisauskiene R, Vaitkiene D, Bariliene S, Andriejaite I, Kliucinskas M, Malakauskiene L, Kemekliene G, Dobozinskas P, Vaitkaitis D, Kudreviciene A, A New technique for clinical skills training in obstetric, MedEdPublish, 2016, 5, [2], 38, http://www.mededpublish.org/manuscripts/498 doi:https://doi.org/10.15694/mep.2016.000066

TAKING A GLANCE AT REAL PROGRAMMING CODE THROUGH THE LENS OF DRAKON LANGUAGE

By Oleg Garipov, CTO, Integrator Software Inc., Halifax, Canada
ogaripov@integratorsoft.com

WORKING WITH CODE IS DIFFICULT

Software developers experience difficulties trying to solve issues with code in complicated software projects. It takes too much time and effort to understand and remember countless details in megabytes of textual code, often in multiple programming languages, scattered around thousands of files and folders. Integrated Development Environments (IDE) like Eclipse, used by most developers to work with code, simplify certain coding tasks, but still require too much effort to navigate through and understand working code.

When fixing issues with existing code, developers spend a lot of time navigating in IDE: scroll files up and down, open other files, click through, search for relevant terms in the code, debugging etc. The purpose of this activity is to locate the lines of code which either need to be changed according to a new requirement, or contain the bug. However, it is quite difficult, because the code is so complicated - there are many navigation possibilities, but IDE only displays a few lines of code at once, most probably irrelevant to the issue.

The text editor window can only display about 10 to 20 lines of code at a time. Just to look through a file with 1,000 lines, the developer will press a Page Down key or click a mouse on the scroll bar more than 50 times. Every time the content of the text editor window is completely replaced with yet another piece of code. It is hard to see two or more related pieces of code located in various positions within the text file or in different files. This creates a huge load on developer's memory with the need to remember dozens of separate pieces of information to construct a mental model of that code.

Another problem is that the text code display in IDE does not show the trajectory of the "instruction pointer" program runner connecting the lines in the order of execution of decision tree branches, known as "control flow." It's easy to get lost in complications of logic of a simple yes/no question when "Yes" block is several screen pages away from the "No" block.

DRAKON TO THE RESCUE

DRAKON is a friendly and intuitive graphical algorithm notation, which makes algorithms easy to grasp. DRAKON presents algorithms in the code clearly, with all intricacies available to be seen at a glance, in the bright sunlight of the day. When developer researches an issue in the code shown on a DRAKON diagram, there is no manual effort to look at different parts of the code and explore their connections. Big picture of the whole issue is seen explicitly, so there is no need to reconstruct it in the memory.

None of the existing IDEs show code as DRAKON, as I know from my more than 20 years of work experience as a software developer in New York City, Boston and Moscow. Is DRAKON useful for working with real world code? I used it in my work with my own and client's code and I think it is. Looking at your own code in DRAKON can be an eye-opening experience!

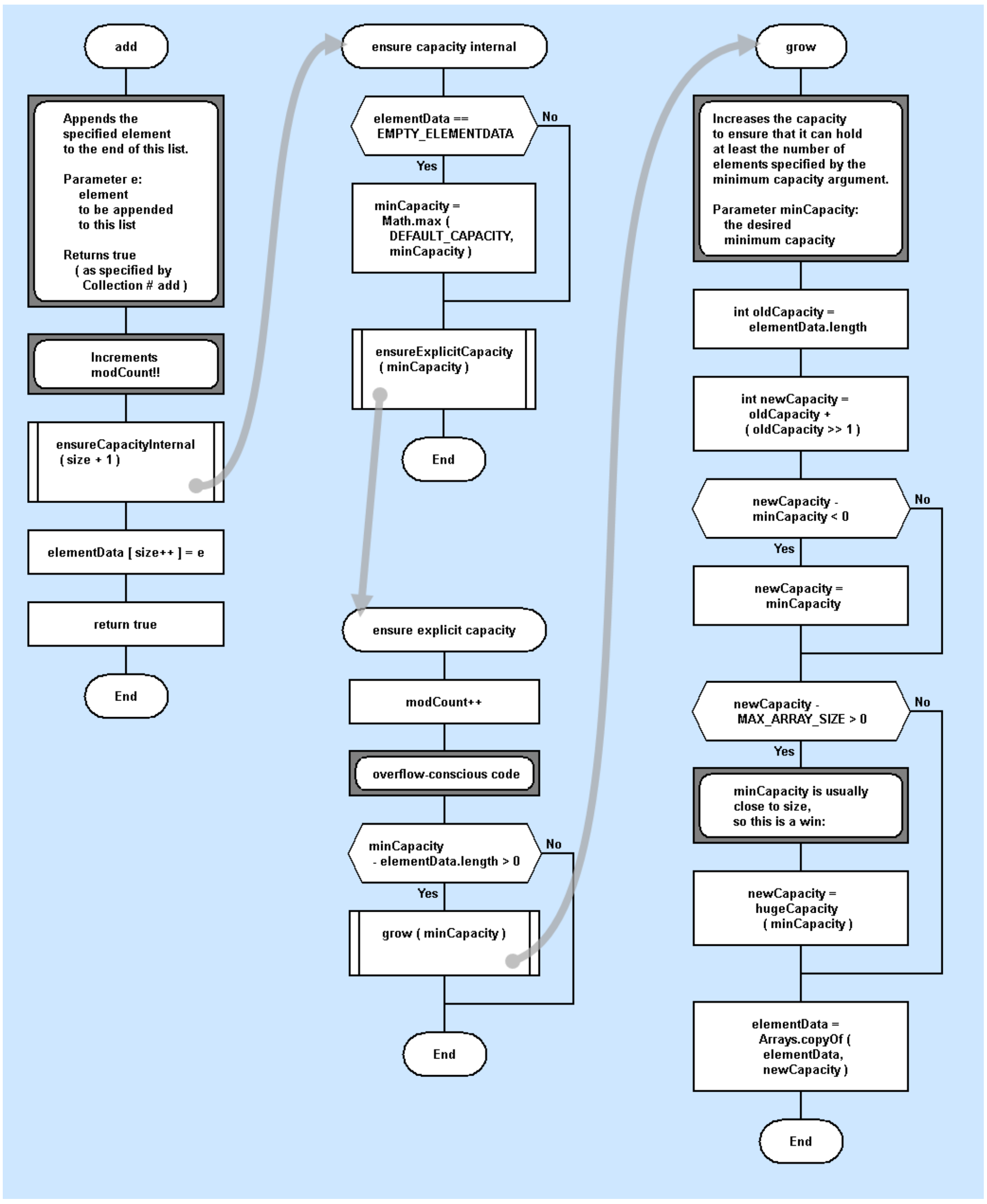

add
Appends the specified element to the end of this list.
Parameter e: element to be appended to this list
Returns true (as specified by Collection # add)
Increments modCount!!
ensureCapacityInternal (size + 1)
elementData [size++] = e
return true
End
ensure capacity internal
elementData == EMPTY_ELEMENTDATA
No
Yes
minCapacity = Math.max (DEFAULT_CAPACITY, minCapacity)
ensureExplicitCapacity (minCapacity)
End
ensure explicit capacity
modCount++
overflow-conscious code
minCapacity - elementData.length > 0
No
Yes
grow (minCapacity)
End
grow
Increases the capacity to ensure that it can hold at least the number of elements specified by the minimum capacity argument.
Parameter minCapacity: the desired minimum capacity
int oldCapacity = elementData.length
int newCapacity = oldCapacity + (oldCapacity >> 1)
newCapacity - minCapacity < 0
No
Yes
newCapacity = minCapacity
newCapacity - MAX_ARRAY_SIZE > 0
No
Yes
minCapacity is usually close to size, so this is a win:
newCapacity = hugeCapacity (minCapacity)
elementData = Arrays.copyOf (elementData, newCapacity)
End

Fig. 1 DRAKON-charts of Java Collections Framework code which adds an object to Array List. This picture was created by Integrator CodeView automatically from breakpoint in Eclipse IDE.

Let’s take code as-is from download of Java Development Kit 1.7, Copyright (c) 1997, 2013, Oracle. This code implements a use case of adding an object to array-based implementation of a resizable list as implemented in Java Collection Framework’s java.util.ArrayList class. Functionality of Array List is actively used in the code of most Java existing applications.

Let’s turn this code into four DRAKON-charts using my program “Integrator CodeView” which is free for evaluation (Fig. 1) .

You can see four algorithms shown as on a single visual scene. IDE requires about four page downs to browse through this code.

Three arrows show calls between DRAKON-charts. Usual IDE does not display the fact that one line of code calls another method. Every possible call requires a manual click-through to explore.

The advantage of DRAKON-chart is that it helps to reduce load on developer’s memory to reconstruct the big picture of the code.

The calls are understandable at a glance, in the tradition of DRAKON.

HOW DRAKON MAKES CODE LOOK SIMPLE AND FUN

DRAKON turns each algorithm into a physical flat object, completely well-defined entity with name on top and clear ending.

The flow of the program runner is clearly seen as a straight skewer line, which goes down, in the intuitive direction of gravity, without visual obstacles.

Questions and choices are easy, even fun to follow, like tracing through a maze of logic.

DRAKON removes obstacles for understanding algorithms. It seems like most people who haven't heard of DRAKON are able to understand what goes on in a complicated DRAKON algorithm, because the flow depiction is unambiguous.

DRAKON makes it easy to locate the precise 'place' of the issue on the big map of algorithm by a mere movement of the eye or finger - possibly the easiest any work action can get. Compare that to thousands of navigation jumps in usual IDE.

DRAKON makes algorithms definite and not confusing, which is helpful since much time is wasted by the teams in the absence of clear understanding.

Seeing the entire DRAKON algorithm in front of their eyes allows people to be in the best position to learn, teach, discuss, review and enhance that algorithm.

LOOKING AT BIG DRAKON PICTURES

DRAKON charts are best viewed on a really big monitor with a really big resolution. Such monitor would display the whole complicated algorithm or a set of algorithms at once, without dividing it into parts. For example, a monitor 100 wide by 30 inches high with a resolution of 10,000 by 3,000 pixels would be able to display DRAKON of a Java file about 100 KB big with 2,500 lines of code. Big companies with big software projects, teams and budgets can afford such displays, because they will help the teams solve real world issues in enterprise code easier.

Currently, most available monitors offer resolution of 1,920 by1,080 pixels (HD). Those can be assembled into a video wall, for example a wall of 5 by 3 HD monitors will provide resolution of 10,000 by 3,000 pixels. Since monitors have bezels, the display will be divided into a 5 by 3 grid, which is a minor inconvenience, the big display is still certainly workable. At a cost of $200 per HD monitor, the monitors for this wall will cost 15 x $200 = only $3,000. PC workstation equipped with a multi monitor video boards will cost another $2,000 . There are companies specializing in constructing video walls like this for advertising, info kiosks, financial data displays etc. Because DRAKON algorithms are displayed 'live', editing them is as easy as touching on the display, entering a change and immediately seeing the updated version.

On an even tighter budget, big DRAKON algorithms can be printed on paper using wide format printers. For example, printer of size Ledger (A3) will allow to readably print algorithms twice as big as the regular Letter (A4) size printer. The same logic, the bigger the better, applies to A2, A1, A0 printers. Printing services offer wide format printers which allow to easily print 10 by 3 feet pictures on a single sheet of paper.

Printouts are displayed on a wall, desk or stand. Teams of developers and non-developers can review them, make notes right on them. Printout relieves from the need to endlessly navigate around a thousand lines of code to answer any questions and solve issues.

It is similar to using geographical maps and nautical charts. The amount of displayed information might be big, but it is easy to zero in on any detail. And when the most precise and dependable knowledge about the geography of a place is required, it is impossible to substitute a map with mere talking about it, albeit at length, or with scribbling quick sketches on a napkin.

BIG PICTURE OF REAL CODE

Fig. 2 shows an example of a big picture of code with several use cases: fifteen algorithms in 300 lines of code, six arrows showing calls. Debugging is paused at the breakpoint on the highlighted operator. Discussing a big readable printout of this picture on the wall might be a good way to introduce a novice into the workings of this code.

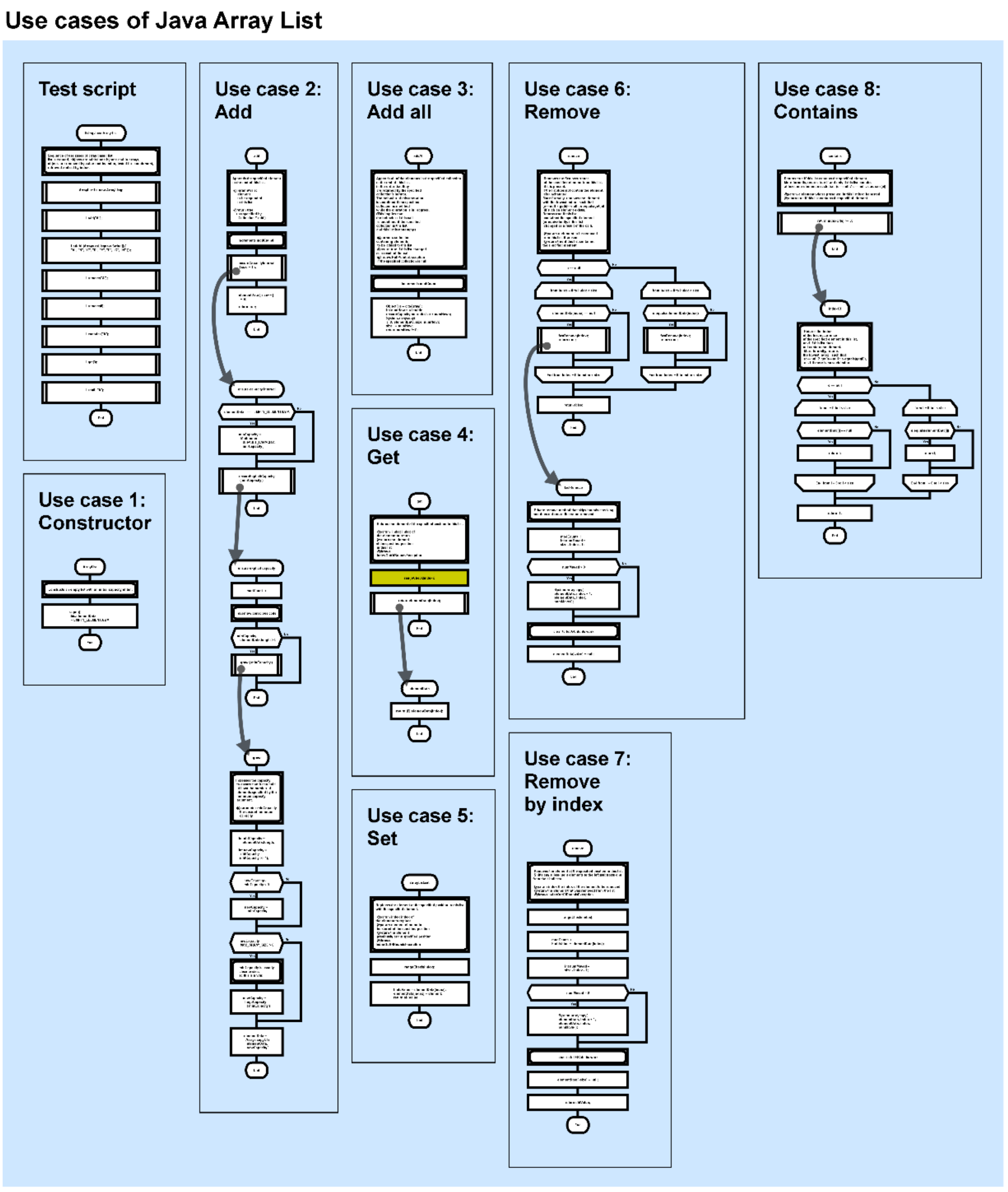


Fig. 2 DRAKON-charts of several use cases of Array List.
This picture was created by Integrator CodeView automatically from a set of breakpoints in Eclipse IDE.

A readable printout of this picture fits, for instance, on a 27 by 37 inch (0.7 by 0.95 meter) precise map of code which eliminates a lot of navigation in IDE. At this book page resolution, code in operators is not readable, but at least DRAKON flows, operator shapes and calls are clear.

What is a “Test script”? This is a small executable application, a unit test. It performs a sequence of 8 actions using Java Array List. Implementation of each of the 8 use cases of Array List is shown with its name and all algorithms called in the process.

Like any other DRAKON-chart, this figure is sufficient to explore, understand and solve issues with the displayed code without any further navigation.

This figure was created in Integrator CodeView from a set of breakpoints in Eclipse IDE.

Use case "Add" in the 2nd column is shown in more detail on Fig. 1.

The readers who want to verify and repeat these results, may download “Integrator CodeView” from [1], then use it on the project with 300 lines of Java Collections Framework code and automatically get the exact drawing on Fig. 2.

TROUBLESHOOTING EXCEPTION STACK TRACE WITH DRAKON

When code runtime execution fails due to internal application failure or network failure, the stack traces of various exceptions often appear in the log files. Manual effort in the IDE will be required to review each line on each stack trace on each server. To see them displayed in DRAKON-chart in CodeView, just copy and paste the text of the traces.

DRAKON helps to troubleshoot exceptions with display of the entire sequence of algorithms and calls between them. Several exceptions are displayed at once to better see possible connections between different parts of failing code.

Fig. 1 and 2 can serve as illustrations of exception stack traces, because every use case may fail just as shown, for example due to hardware-caused JDK failure etc. So each of the call arrow sequences is potentially an exception stack trace.

DYNAMICALLY HIGHLIGHTING DRAKON OPERATORS TO TRACE EXECUTION DURING DEBUGGING

When debugging in IDE, developers step through the code, pausing on one line of code at a time to see the view of a few lines surrounding the current line, and that view keeps changing every step.

This makes trivial movement of a focus within the algorithms look overly complicated.

Instead, when a set of algorithms is shown as a DRAKON-chart, the execution point is highlighted every step by painting the current operator background in a bright color, clearly visible among other operators, which remain with a neutral white background, as shown on Fig. 2.

DRAKON-chart itself does not change during the execution, only the background of operators gets dynamically highlighted. This greatly simplifies the appearance of a debugging session. Now it looks like an elegant dance of a spotlight on a single, unchanging, clear DRAKON-chart, rather than thousands of ever changing views of one chunk of code at a time.

DRAKON FACILITATES CODE REVIEW

To review all changes made in the code in a release containing fixes for a few bugs will take approximately as many clicks as there were code line changes, which might be thousands.

Similarly to big displays of use cases, displaying code changes made by a team of developers in a monthly release is done with a big display of DRAKON algorithms with changes highlighted in operators.

When two or more developers are looking at the same code in IDE, manual effort to navigate through each step in a use case needs to be repeated again and again. The display remains incomplete at any given moment. It is bad.

That is why DRAKON is needed. Reviewing code changes displayed in use cases on big DRAKON-chart displays is efficient, compared to manually clicking on each change. CodeView highlights changes by date or author, as shown in Figure 3. Isn't this approach easier?

Reviewing code logic and flows of calls is facilitated by the power of DRAKON to clearly visualize existing Java code ergonomically.

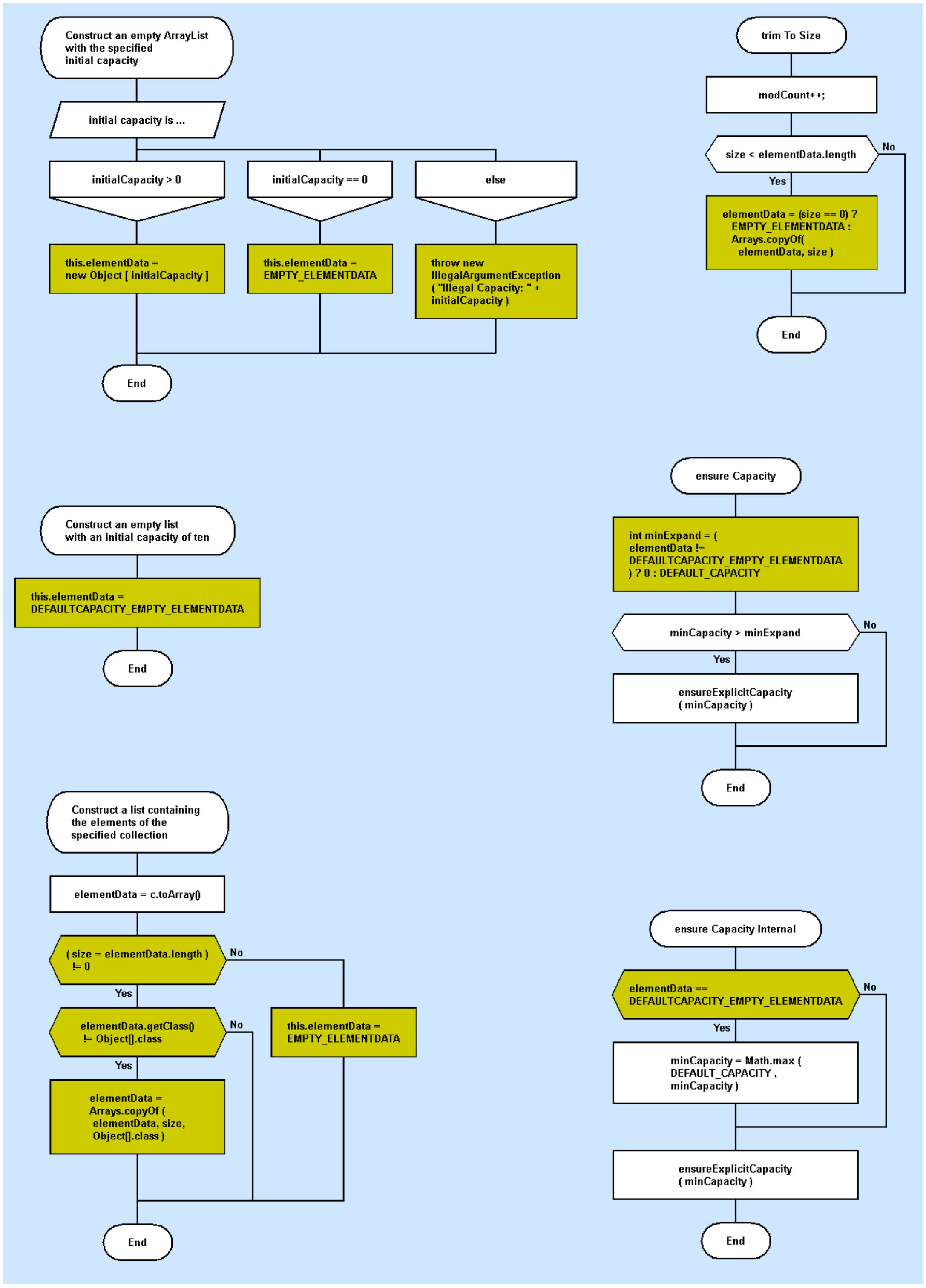

Construct an empty ArrayList with the specified initial capacity
initial capacity is ...
initialCapacity > 0
initialCapacity == 0
else
this.elementData = new Object [initialCapacity]
this.elementData = EMPTY_ELEMENTDATA
throw new IllegalArgumentException ("Illegal Capacity: " + initialCapacity)
End
trim To Size
modCount++;
size < elementData.length
No
Yes
elementData = (size == 0) ? EMPTY_ELEMENTDATA : Arrays.copyOf(elementData, size)
End
ensure Capacity
int minExpand = (elementData != DEFAULTCAPACITY_EMPTY_ELEMENTDATA) ? 0 : DEFAULT_CAPACITY
minCapacity > minExpand
No
Yes
ensureExplicitCapacity (minCapacity)
End
Construct an empty list with an initial capacity of ten
this.elementData = DEFAULTCAPACITY_EMPTY_ELEMENTDATA
End
Construct a list containing the elements of the specified collection
elementData = c.toArray()
(size = elementData.length) != 0
No
Yes
elementData.getClass() != Object[].class
No
Yes
this.elementData = EMPTY_ELEMENTDATA
elementData = Arrays.copyOf (elementData, size, Object[].class)
End
ensure Capacity Internal
elementData == DEFAULTCAPACITY_EMPTY_ELEMENTDATA
No
Yes
minCapacity = Math.max (DEFAULT_CAPACITY , minCapacity)
ensureExplicitCapacity (minCapacity)
End

Fig. 3 Code changes on DRAKON-charts of Array List code in Java version 1.8 as compared to Java 1.7

Let us suppose that we are interested in reviewing all code changes in Array List code in Java version 1.8 as compared to Java version 1.7. Let’s take a glance at Fig. 3. Changed operators are automatically highlighted by Integrator CodeView on DRAKON-charts, the rest remain white. Algorithms that did not change are not shown. This display can be verified by comparing Java sources available in src.zip file in the installed JDKs of versions 1.8 and 1.7.

HIGHLIGHTING SEARCH RESULTS ON DRAKON

Array List implements resizable list on the basis of a non-resizable array. When a new object is added dynamically, array of a bigger size may need to be created to make space for more objects.

Let us imagine that Array List is not the well-tested code it is, but some newly written or legacy code, and as it often happens, it contains a bug and doesn’t work correctly. For example, when the array is re-created, its size is not correct. In this case, developer will need to look at the code and maybe search for the word “size” in the code to see which lines of code change that size.

Examining search results in the IDE requires a manual click on each found instance, one by one. When there are many found instances, it might be a significant effort. It is also hard to see several found lines side-by-side at the same time to double-check or compare them, without scrolling back and forth.

On DRAKON-chart search results in operators are highlighted at once, rather than one by one.

Let us try to solve the hypothetical issue with ‘size’ of the new array not being calculated as expected. Fig. 4 shows results of searching for code with “size” or “SIZE” in it. Found operators with matches are highlighted in yellow, the rest remain white. Algorithms without any matches are not shown.

Now all algorithms working with array size can be examined without further manual effort. If any of the highlighted operators contained a mistake in the code, it would be clearly seen on Fig. 4.

DRAKON HELPS NON-DEVELOPERS UNDERSTAND CODE

Algorithms in the existing code normally look like DRAKON primitives shown on Fig. 1 through 4.

However, non-developers still do not understand code in this depiction. Non-developers speak English, not Java. They also need help identifying important sub-tasks of complicated multi-stage algorithms.

Representing algorithm in the code as DRAKON Silhouette in English enables non-developers to see the actual code, verify the logic of unambiguous DRAKON line flows, identify the sub-tasks, locate the operators performing certain actions – in other words, navigate in that code just like developers. Integrator CodeView includes functionality which helps to do that.

We start with an original primitive algorithm in Java, such as “grow” in the right-most column of Fig. 1. Then long linear sequences of actions on the skewer of the primitive are split into multiple addresses.

Original code display on each operator is overwritten with English, using keyboard or voice recognition input. This does not spoil the original code in anyway, but simply adds an English comment to the operator object implementation in CodeView.

All algorithms in a use case or an issue are translated into English DRAKON Silhouette in this way. The resulting view of code as DRAKON Silhouette makes mysterious code more approachable to non-developers and enables them to provide insight into solving issues with code.

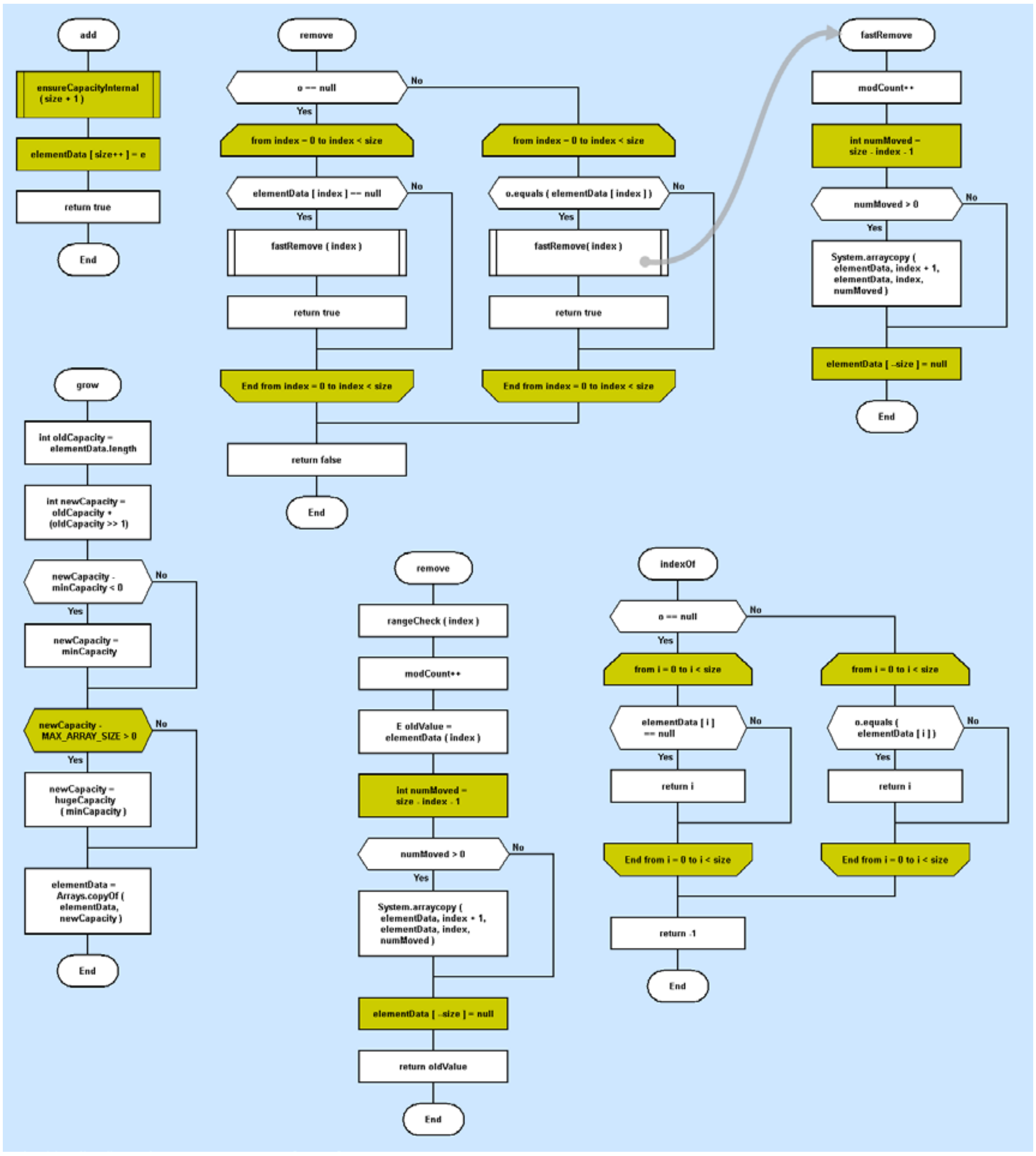

add
ensureCapacityInternal (size + 1)
elementData [size++] = e
return true
End
remove
o == null
No
Yes
from index = 0 to index < size
elementData [index] == null
fastRemove (index)
return true
End from index = 0 to index < size
o.equals (elementData [index])
fastRemove(index)
return false
End
fastRemove
modCount++
int numMoved = size - index - 1
numMoved > 0
System.arraycopy (elementData, index + 1, elementData, index, numMoved)
elementData [--size] = null
grow
int oldCapacity = elementData.length
int newCapacity = oldCapacity + (oldCapacity >> 1)
newCapacity - minCapacity < 0
newCapacity = minCapacity
newCapacity - MAX_ARRAY_SIZE > 0
newCapacity = hugeCapacity (minCapacity)
elementData = Arrays.copyOf (elementData, newCapacity)
rangeCheck (index)
E oldValue = elementData (index)
return oldValue
indexOf
from i = 0 to i < size
elementData [i] == null
return i
o.equals (elementData [i])
End from i = 0 to i < size
return -1

Fig. 4 Looking for cause of bug with array resizing: results of searching for "size" ignoring case in the Array List code highlighted on DRAKON-charts

INTEGRATOR SOFTWARE

Developing and maintaining code is difficult and takes a lot of time and resources.

Is there a way to simplify those complexities and make it easier to achieve every day development goals? Integrator Software was founded to provide such a tool. It combines the help of several graphic languages to assist in code development.

Integrator Software is a startup with a suite of products which help companies and developers work with code.

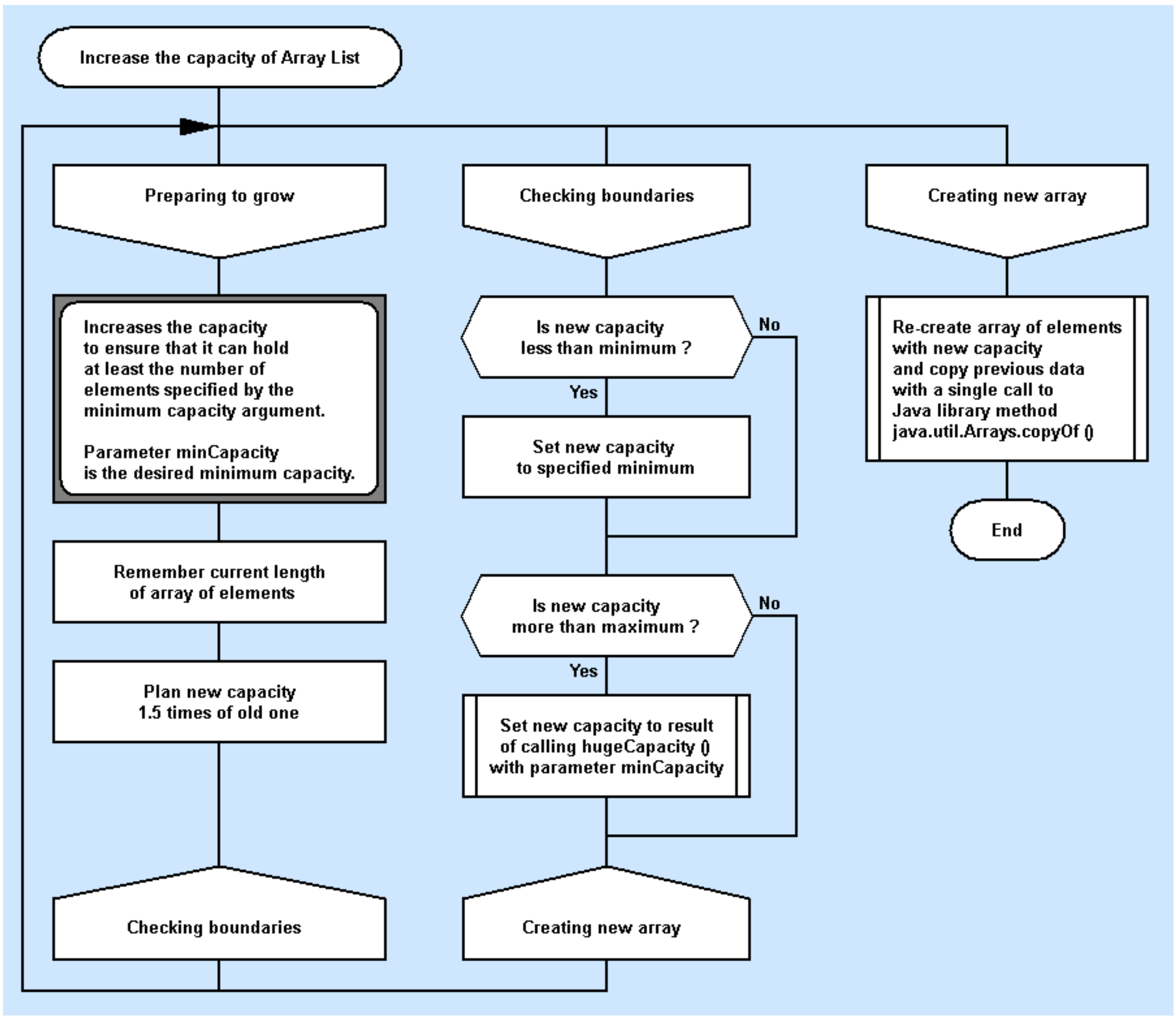

Increase the capacity of Array List
Preparing to grow
Checking boundaries
Creating new array
Increases the capacity to ensure that it can hold at least the number of elements specified by the minimum capacity argument.
Parameter minCapacity is the desired minimum capacity.
Is new capacity less than minimum ?
No
Yes
Set new capacity to specified minimum
Re-create array of elements with new capacity and copy previous data with a single call to Java library method java.util.Arrays.copyOf ()
End
Remember current length of array of elements
Plan new capacity 1.5 times of old one
Is new capacity more than maximum ?
No
Yes
Set new capacity to result of calling hugeCapacity () with parameter minCapacity
Checking boundaries
Creating new array

Fig. 5 English DRAKON Silhouette of a method in Java Array List, which creates a bigger underlying array

INTEGRATOR CODEVIEW

The product called Integrator CodeView automatically represents algorithms in existing code as DRAKON charts.

DRAKON view of code is automatically cleaned up from visual noise, such as repetitive structural punctuation.

CodeView can handle big amounts of modern code, with a single picture containing thousands of lines of code and hundreds of DRAKON charts.

Integrator CodeView can currently visualize code in Java, JavaScript, C#, C, Python.

The code can be taken from a file, folder, IDE project, GitHub or SVN repository.

Code debugging features visualize the sequence of calls between DRAKON algorithms as arrows.

CodeView understands Exception stack traces, grabs them from application log files and shows as highlights of corresponding DRAKON operators with background color.

When debugging several breakpoints in IDEs, users can bring together, observe and analyze all related code as DRAKON charts with call connections on a diagram.

The history of changes in code repository is shown by highlighting corresponding DRAKON operators.

Any given troublesome line seen in the IDE can be highlighted in the corresponding DRAKON operator on the big picture.

EVALUATION VERSION AND WEBSITE

A demo version is available for evaluation as a plugin to Eclipse IDE on http://integratorsoft.com/codeview .

For more information, examples and videos of code in DRAKON visit our website http://integratorsoft.com

CONCLUSION

Learning and working with complicated code can be simplified if the code is automatically shown as DRAKON charts.

Ergonomic benefits of DRAKON visual language help users to clearly see the flows in the complicated code.

The Integrator software helps programmers to learn a complicated project in a few hours by looking at a big picture of DRAKON algorithms, literally without moving a finger.

Compare that to the whole days of clicking and typing in modern IDEs which are typically required for a developer to get "on board".

DRAKON is a powerful tool to tame the inherent complexity of software development.

REFERENCES

1. Integrator Software website http://integratorsoft.com
2. Integrator CodeView, DRAKON code visualizer http://integratorsoft.com/codeview
3. Integrator Software Patent “Integrated Software Development Environments, Systems, Methods, and Memory Models” International Publication Number: WO 2017/031459 A1. International publication date: 23 February 2017. International application number: PCT/US2016/047871. International Filing Date: 19 August 2016 https://www.google.com/patents/WO2017031459A1

ABOUT DRAKON EDITORS BY STEPAN MITKIN

By Basil Valevich, software developer, Belarus
basil.valevich@gmail.com

DRAKON helps people understand programs. DRAKON Editors of Stepan Mitkin are part of existing DRAKON Tools. In addition to Stepan's editors, there are also IS Dragon of Gennady Tyshov, Fabula of Eduard Ilchenko, and DRAKON-Assembler of Artem Brazovsky. DRAKON Editors of Stepan Mitkin and IS Dragon of Gennady Tyshov are the most advanced tools of those that are in the public domain.

USING OF DRAKON EDITOR

DRAKON Editor is a desktop program for editing algorithms in the DRAKON language. DRAKON Editor allows you to program with the help of DRAKON schemes. Programmers can, with the help of DRAKON Editor, generate source code from DRAKON schemas.

Several programming languages are supported: C/C ++, Qt, Java, Processing.org, C#, Python, Tcl, Javascript, Erlang, Lua. DRAKON Editor supports all icons from the graphic alphabet of the DRAKON language.

The desktop app runs on Windows, Mac, and Linux operating systems. To run the DRAKON Editor application, you must first install the Tcl/Tk library. The program is supplied under a free license (public domain). DRAKON Editor is distributed in Russian and English.

ADVANTAGES

Code generation. The ability to generate code for various algorithmic languages is one of the main advantages of the DRAKON Editor from the point of view of a programmer. Before generating the code, it automatically checks if the file diagrams match the rules of the DRAKON language.

Sources. DRAKON Editor comes with open-source. You can expand the list of supported programming languages if necessary. Most developers will be able to adapt the DRAKON Editor application to their specific tasks with the help of a debugger. Some important examples and recommendations are in the accompanying folders of DRAKON Editor.

Snippets. If DRAKON Editor is used for code analysis, the programmer can use the code snippets as separate visual objects. The block of code added as a snippet to the DRAKON Editor diagram can be removed from the method and replaced with another block of code. Approximately how in a personal computer one module is replaced by another module. This is a convenient opportunity for recombining and for refactoring of code. Using of code snippets as separate visual objects allows you to get a two-sided growth of productivity when using DRAKON Editor. On the one side, the strict rules of the DRAKON language lead the development straight to the goal. On the other side, code snippets allow you to restrict the effect of strict rules of the DRAKON language wherever the developer does not need it based on performance considerations. For example, in DRAKON Editor you can create a silhouette in which there are many If blocks. In this case, you can insert all If blocks in the form of a single block of code if there is no time for detailed drawing and if so everything is clear. This combination of code snippets and DRAKON rules is viable option. Ultimately, DRAKON Editor provides a time benefit. The ability to operate with code snippets in the DRAKON Editor is a surprisingly simple and powerful solution for productivity growth.

Independence. One of the advantages of DRAKON Editor as a desktop application is its independence from the Internet connection and network bandwidth.

DISADVANTAGES

Principle of free drawing. DRAKON Editor uses the principle of free drawing. You can draw not only DRAKON schemes, but also other schemes. However, some users believe that such universality leads to the loss of benefits. The Verify button which checks compliance with the rules does not solve the problem. In the discussion about the principle of free drawing, users expressed their arguments about the need to limit the freedom of "drawing creativity", the need to programmatically prohibit violation of the rules of DRAKON and the need to programmatically force the user to strictly comply with the rules of DRAKON.

Generating diagrams from code. Also, users in the discussion noted that there is no automatic conversion of the program code back to the diagram.

TARGET AUDIENCE AND PRICES

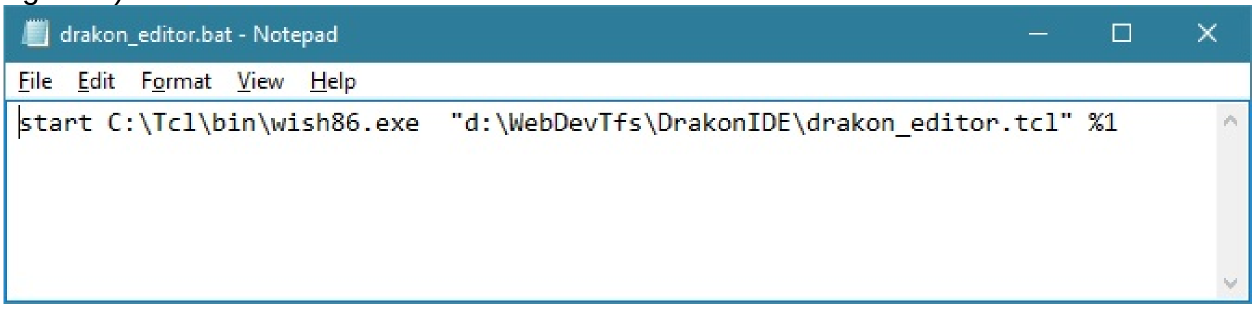
Developers, Team Leaders, System Architects, Project Managers, Analysts, Quality Assurance Managers, System Administrators, Chief Technology Officers.

Let us add that offline DRAKON EDITOR is free of charge.

EXAMPLE OF USING

Working on tasks in startups, I used the DRAKON Editor to analyze processes, formalize requirements, and generate the code. In one of such tasks, I faced the need to develop a long algorithm for sampling and manipulating data on energy and power consumption. At some point, the development stalled, the situation just seemed to be deadlocked.

Then I decided to rewrite the problematic classes in DRAKON. I integrated the DRAKON Editor with Visual Studio 2017 (Figure 1, Figure 2, and Figure 3). I added drn-files to the project and code repository and started to regenerate source files from the drn-files (Figure 4).

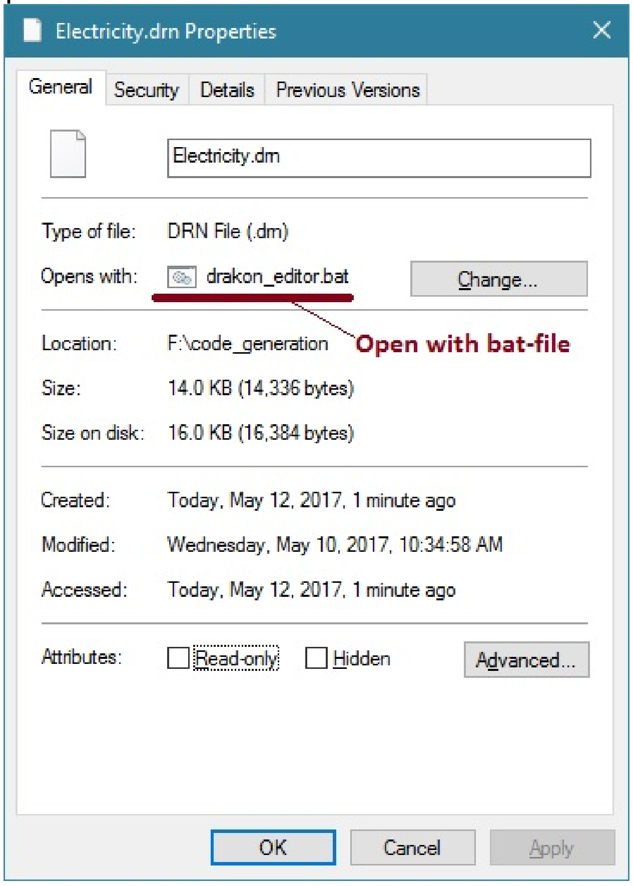


```
start C:\Tcl\bin\wish86.exe  "d:\WebDevTfs\DrakonIDE\drakon_editor.tcl" %1
```

Fig. 1 Example of a bat-file for running drn-files

TFS (Team Foundation Server) is a Microsoft product that provides source code management capabilities. Subversion (often abbreviated SVN) is a software versioning and revision control system. I have a daily statistic which is called Changeset in TFS. Changeset in TFS that's basically the same thing Commit in SVN.

On the graph of statistic, you can see that the using of DRAKON Editor significantly improved the situation (Figure 5). The issue was resolved right in a matter of hours. The code began to grow gradually, in the normal mode. A couple of days after I started using the DRAKON Editor, the statistics increased about 10-fold and maintained at this high level. The task was successfully completed.



Electricity.drn Properties

General | Security | Details | Previous Versions

Electricity.drn

Type of file: DRN File (.drn)

Opens with: drakon_editor.bat

Location: F:\code_generation

Size: 14.0 KB (14,336 bytes)

Size on disk: 16.0 KB (16,384 bytes)

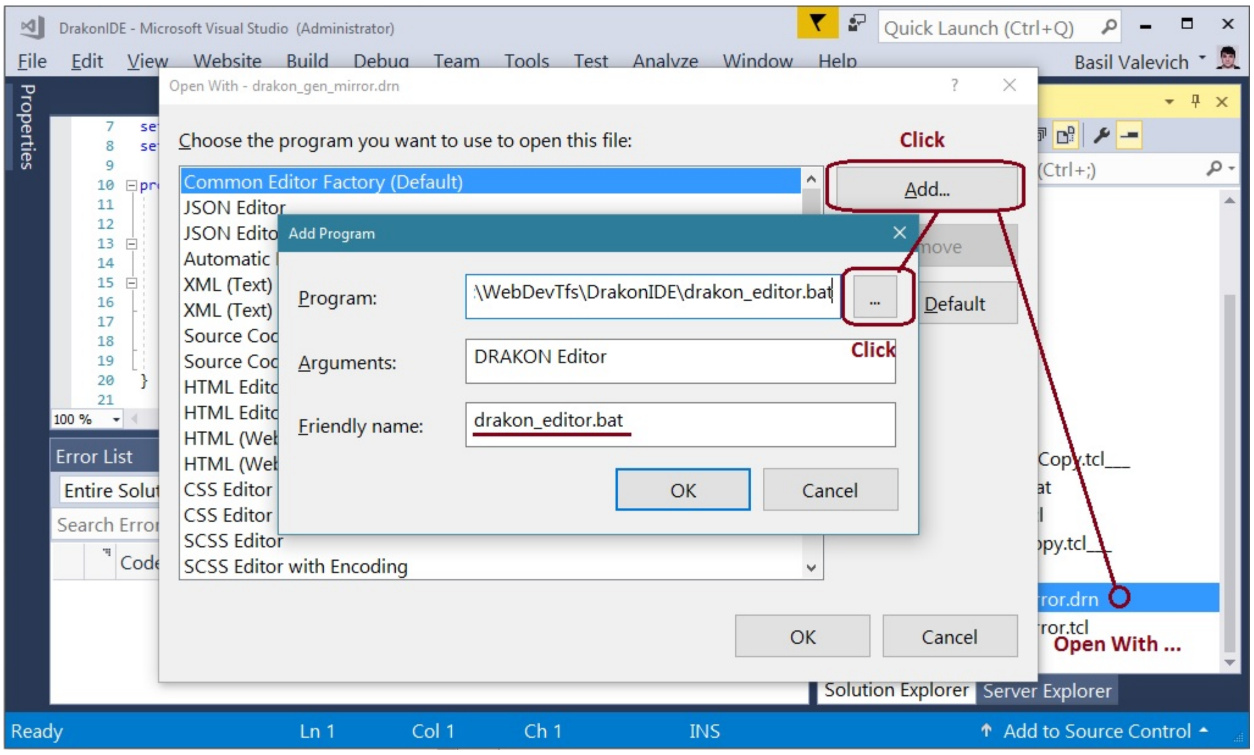
Created: Today, May 12, 2017, 1 minute ago

Modified: Wednesday, May 10, 2017, 10:34:58 AM

Accessed: Today, May 12, 2017, 1 minute ago

Attributes: ☐ Read-only ☐ Hidden

Fig. 2 Example of the drn-file properties for Windows



DrakonIDE - Microsoft Visual Studio (Administrator)

File Edit View Website Build Debug Team Tools Test Analyze Window Help

Open With - drakon_gen_mirror.drn

Choose the program you want to use to open this file:

Common Editor Factory (Default)
JSON Editor
JSON Editor
Automatic
XML (Text)
XML (Text)
Source Code
Source Code
HTML Editor
HTML Editor
HTML (Web
HTML (Web
CSS Editor
CSS Editor
SCSS Editor
SCSS Editor with Encoding

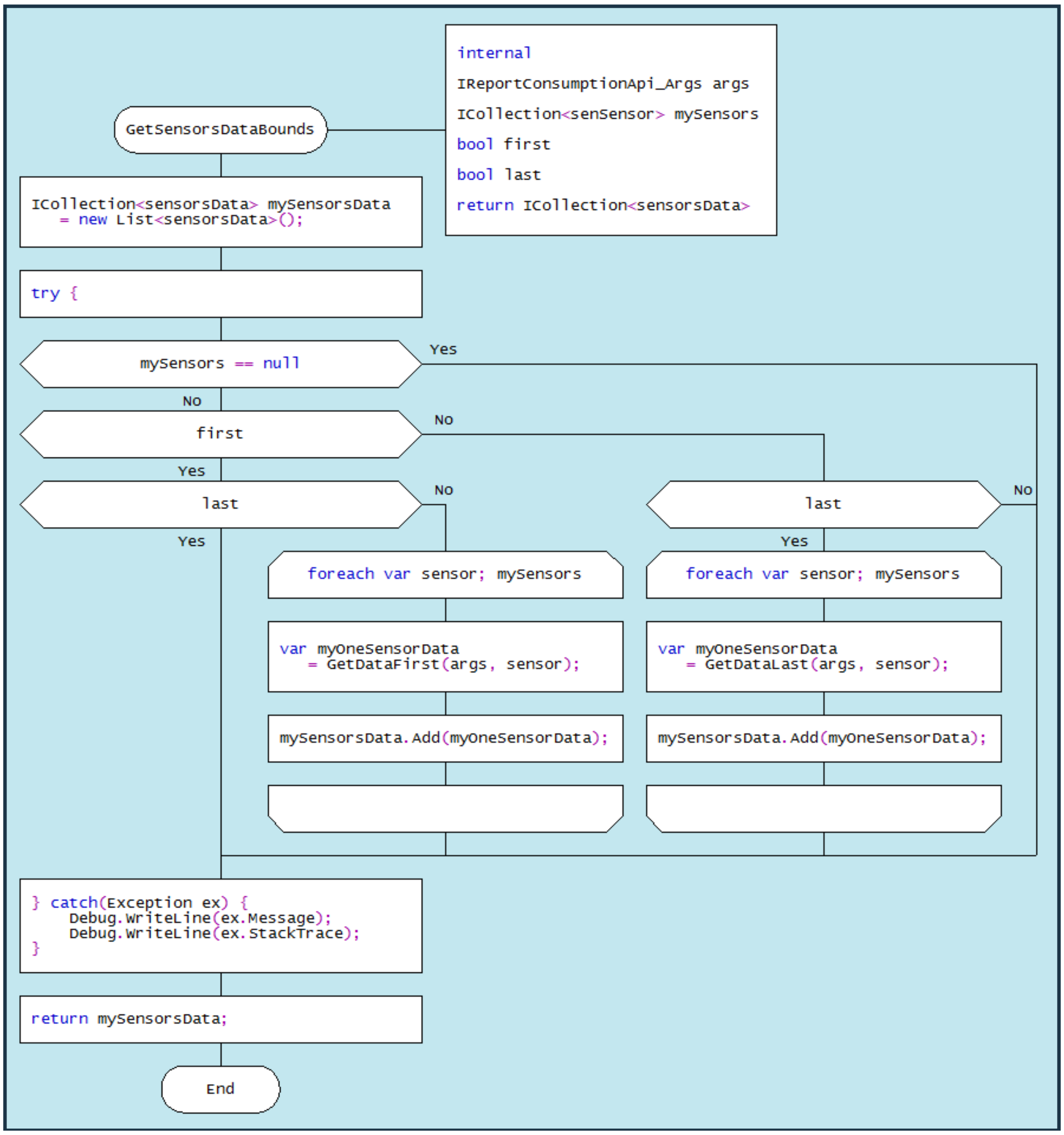
Add Program

Program: \WebDevTfs\DrakonIDE\drakon_editor.bat

Arguments: DRAKON Editor

Friendly name: drakon_editor.bat

Fig. 3 Example of Integration DRAKON Editor and Visual Studio 2017



```
internal
IReportConsumptionApi_Args args
ICollection<senSensor> mySensors
bool first
bool last
return ICollection<sensorsData>
```

```
ICollection<sensorsData> mySensorsData
   = new List<sensorsData>();
```

```
try {
```

```
mySensors == null
```

```
first
```

```
last
```

```
foreach var sensor; mySensors
```

```
var myOneSensorData
   = GetDataFirst(args, sensor);
```

```
mySensorsData.Add(myOneSensorData);
```

```
last
```

```
foreach var sensor; mySensors
```

```
var myOneSensorData
   = GetDataLast(args, sensor);
```

```
mySensorsData.Add(myOneSensorData);
```

```
} catch(Exception ex) {
     Debug.WriteLine(ex.Message);
     Debug.WriteLine(ex.StackTrace);
}
```

```
return mySensorsData;
```

Fig. 4 The skewer-method of DRAKON and code snippets

SUMMARY

DRAKON works well with the code if SOLID principles are used. SOLID (single responsibility, open-closed, Liskov substitution, interface segregation and dependency inversion) is an abbreviation for five basic principles of object-oriented programming and design.

The DRAKON Editor is a handy tool. Thanks to it, I successfully completed this task much faster. The DRAKON Editor really improves the quality and speed of coding. It is interesting that the code is fully under control, it compiles immediately without errors.

DRAKON Editor improves code development processes.

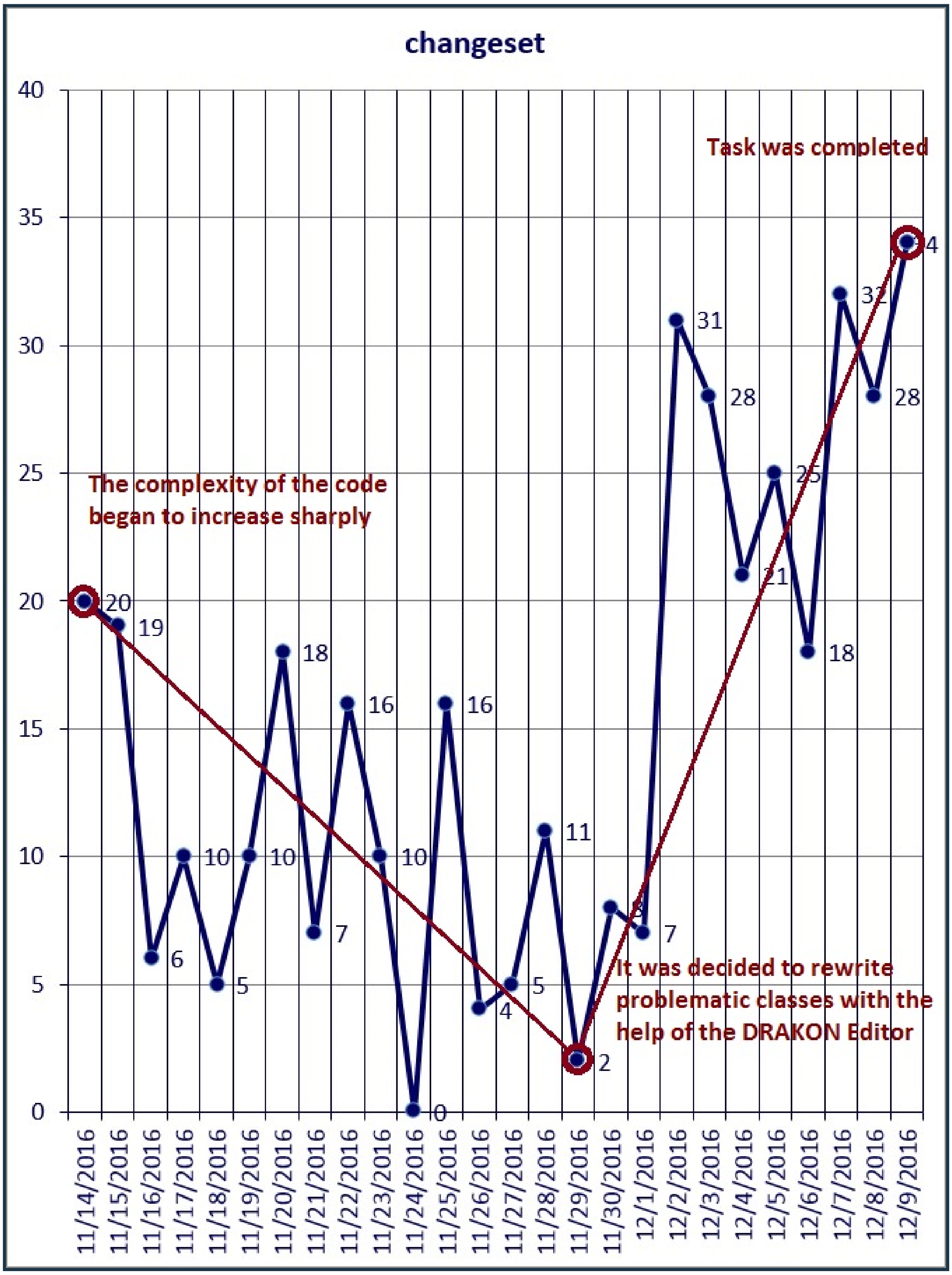


Fig. 5 Statistics of code creation before and after applying the DRAKON Editor

USING OF DRAKON EDITOR WEB

DRAKON Editor Web is a web-based tool for creating and quickly changing the DRAKON diagrams. This is an online flowchart editor for business procedures. The application is accessible via the web interface through a browser.

The server part of DRAKON Editor Web is rewritten in Lua programming language. Previously, the server part was written in Erlang programming language. DRAKON Editor Web works on the basis of Tarantool. Tarantool is an open-source NoSQL database management system and Lua application server.

DRAKON Editor Web is available in Russian and English. The DRAKON Editor and the DRAKON Editor Web are different applications. Each of them is attractive in its own way. They do not replace but rather complement each other.

ADVANTAGES

Resources. DRAKON Editor Web does not impose any special requirements on resources and hardware platform. It only requires a browser and Internet access. Everything else for working with DRAKON Editor Web is of secondary importance.

Mobility. DRAKON Editor Web allows users to be truly mobile. Users can use a computer, tablet, netbook or smartphone. DRAKON Editor Web is displayed in the browser using browser capabilities, such as Javascript and CSS processing.

Diagrams. DRAKON Editor Web allows you to create projects, folders and diagrams. It is possible to create two types of diagrams - DRAKON diagram and Free-form diagram. Thus for the first type of diagrams (DRAKON diagram) more functionality is realized.

Navigation. DRAKON Editor Web has a convenient navigation system. Implemented the possibility of decomposition through the transitions from the icon to its detailed diagram.

Cookies. Login and password entered in DRAKON Editor Web are saved for a long time thanks to the use of cookies. This is a convenient option for users.

Service. Another advantage of DRAKON Editor Web is the minimization of maintenance problems. The developers of DRAKON Editor Web themselves who have sufficient competencies act as administrators of DRAKON Editor Web.

Installation. One of the advantages of DRAKON Editor Web is that the software does not need to be installed. The DRAKON Editor Web software is already installed, configured and constantly updated in the cloud.

Exchange. DRAKON Editor Web can be configured for both individual and shared access. Users of Linux, Windows or Mac can freely share diagrams among themselves. Several DRAKON Editor Web users can view and edit the same diagram from different devices at the same time. All the changes that have been made by one of the team members will immediately be able to see the entire team that is working on the project. When working in the DRAKON Editor Web, there is no need to regularly send each other new versions of diagrams, because they are always available in the current state.

DISADVANTAGES

Internet. To access DRAKON Editor Web, you need a permanent Internet connection. Network problems can result in the user not being able to work with the DRAKON Editor Web.

Diagrams. One of the types of diagrams implemented in DRAKON Editor Web (specifically the Free-form diagram) looks, at the moment, slightly underdeveloped.

Code generation. The lack of code generation is one of the disadvantage of DRAKON Editor Web.

API. Another disadvantage of DRAKON Editor Web is that it does not support API (Application Programming Interface).

Confidentiality. The problem of data confidentiality in DRAKON Editor Web requires more detailed consideration. Some users of DRAKON Editor Web may be concerned about the fact that their data will be processed somewhere and stored on a foreign server.

TARGET AUDIENCE AND PRICES

Developers, Team Leaders, System Architects, Project Managers, Analysts, Quality Assurance Managers, System Administrators, Chief Technology Officers, Chief Executive Officers, Company Founders.

Basic subscription plan = Free. Extended subscription plan = EUR 5 / per month. Team subscription plan = from EUR 12 / per month.

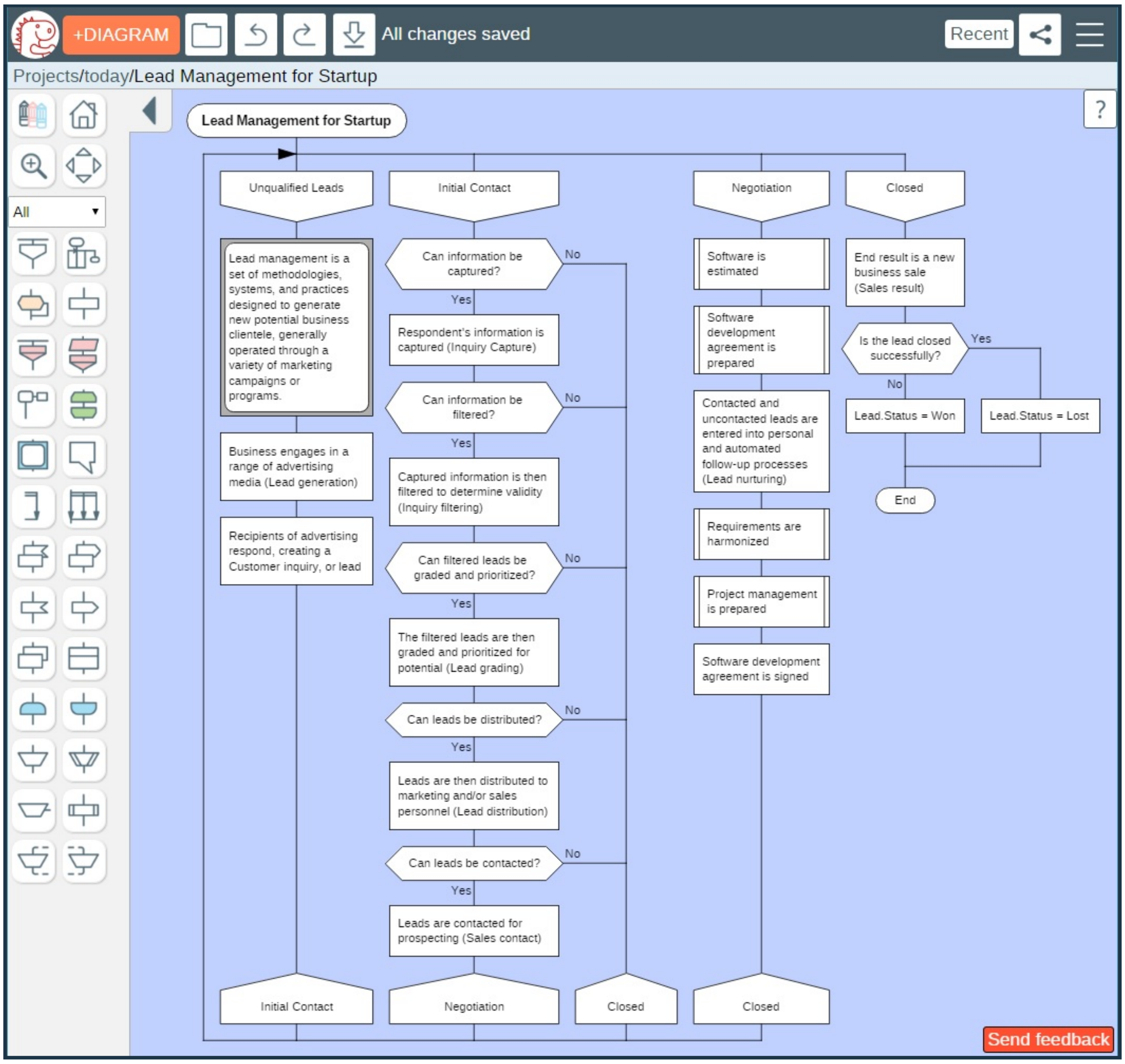

+DIAGRAM
All changes saved
Recent
Projects/today/Lead Management for Startup
All
?
Lead Management for Startup
Unqualified Leads
Initial Contact
Negotiation
Closed
Lead management is a set of methodologies, systems, and practices designed to generate new potential business clientele, generally operated through a variety of marketing campaigns or programs.
Business engages in a range of advertising media (Lead generation)
Recipients of advertising respond, creating a Customer inquiry, or lead
Can information be captured?
No
Yes
Respondent's information is captured (Inquiry Capture)
Can information be filtered?
No
Yes
Captured information is then filtered to determine validity (Inquiry filtering)
Can filtered leads be graded and prioritized?
No
Yes
The filtered leads are then graded and prioritized for potential (Lead grading)
Can leads be distributed?
No
Yes
Leads are then distributed to marketing and/or sales personnel (Lead distribution)
Can leads be contacted?
No
Yes
Leads are contacted for prospecting (Sales contact)
Software is estimated
Software development agreement is prepared
Contacted and uncontacted leads are entered into personal and automated follow-up processes (Lead nurturing)
Requirements are harmonized
Project management is prepared
Software development agreement is signed
End result is a new business sale (Sales result)
Is the lead closed successfully?
Yes
No
Lead.Status = Won
Lead.Status = Lost
End
Initial Contact
Negotiation
Closed
Closed
Send feedback

Fig. 6 Example of a business process designed in DRAKON Editor Web

EXAMPLE OF USING

The Theory of Constraints (TOC) is a management paradigm that views any manageable system as being limited in achieving more of its goals by a very small number of constraints. Theory of Constraints also is pioneering business process improvement methodology.

DRAKON Editor Web well corresponds to the paradigm of constraints. In my opinion, DRAKON is one of the best and most practical world tool for theory of constraints. Example of a business process designed in DRAKON Editor Web is shown in Figure 6.

In summary, DRAKON Editor Web improves business processes.

CONCLUSION

Based on the results of the performed analysis, we can conclude that DRAKON Editor and DRAKON Editor Web can be recommended for general use as tools that improve algorithmic processes.

REFERENCES

1. Parondzhanov, V. D. How to improve the workings of the mind. Algorithms without programmers – this is very simple! – (Only in Russian). – Паронджанов, В. Д. Как улучшить работу ума. Алгоритмы без программистов – это очень просто! – М.: Дело, 2001. – 360 с. – Ил.: 154. – http://drakon.su/_media/biblioteka_1/parondzhanov_v.d._kak_uluchshit_rabotu_uma_.pdf

2. Parondzhanov, V. D. The DRAKON Language. Short Description. – (Only in Russian). – Паронджанов, В. Д. Язык ДРАКОН. Краткое описание. – 2009. – 124 с. – http://drakon.su/_media/biblioteka/drakondescription.pdf

3. Parondzhanov, V. D. Friendly algorithms, understandable to everyone.(How to improve the workings of the mind, without extra trouble). – (Only in Russian). – Паронджанов, В. Д. Дружелюбные алгоритмы, понятные каждому. (Как улучшить работу ума без лишних хлопот). – М.: ДМК-пресс, 2010. – 464 с. – Ил.: 233. – http://drakon.su/_media/biblioteka_1/03._2010_druzheljubnye_algoritmy_1.pdf

4. Mitkin, S. I want to program in DRAKON. Is that possible? / By Stepan Mitkin, software developer, Norway, 2017 (PDF File)

5. DRAKON Editor for Desktop // Slashdot Media. – http://drakon-editor.sourceforge.net

6. Mitkin, S. DRAKON. The Human Revolution in Understanding Programs // Slashdot Media. – October 2011. – http://drakon-editor.sourceforge.net/DRAKON.pdf

7. DRAKON Editor Web // DRAKON Labs. – 2015-2017. – https://drakon-editor.com

8. Mitkin, S. Smart editing mode in DRAKON Editor // Slashdot Media. – 20.05.2012. – http://drakon-editor.sourceforge.net/smart.html

9. Mitkin, S. DRAKON-Erlang: Visual Functional Programming // Slashdot Media. – http://drakon-editor.sourceforge.net/drakon-erlang/intro.html

10. Mitkin, S. State Machines in DRAKON Editor // Slashdot Media. – 29.12.2014. – http://drakon-editor.sourceforge.net/auto.html

11. Mitkin, S. DRAKON, Actors and Message Passing // Slashdot Media. – http://drakon-editor.sourceforge.net/actors.html

12. Mitkin, S. State Machines in DRAKON-Erlang // Slashdot Media. – http://drakon-editor.sourceforge.net/erlang_auto.html

13. Mitkin, S. ERIL: a Visual Language for Data Modelling. ERIL is an attempt to apply the principles of DRAKON to entity-relationship and class diagrams // Slashdot Media. – http://drakon-editor.sourceforge.net/eril.html

14. Mitkin, S. A DRAKON-C# application that shows off DRAKON state machines for concurrency // GitHub. – https://github.com/stepan-mitkin/actor-http

15. DRAKON. Flowcharts that have order. Part 1 // DRAKONlanguage. – 02.02.2017. – https://www.youtube.com/watch?v=j-M_z_7_Nkw

16. DRAKON. Flowcharts that have order. Part 2 // DRAKONlanguage. – 24.11.2016. – https://www.youtube.com/watch?v=Q0YBdX9xazw

17. Mitkin, S. Visual functional programming with DRAKON-Erlang. Erlang User Conference 2015 // Erlang Solutions. – 30.07.2015. – https://www.youtube.com/watch?v=yZLedcnFA94

18. Валевич, В. Способы организации бизнеса // SciTecLibrary (Scientific and

Technical Library) – 08.09.2003. – http://www.sciteclibrary.ru/rus/catalog/pages/6015.

19. Valevich, V. Ways of Business Organization // SciTecLibrary (Scientific and Technical Library) – 08.09.2003. – http://www.sciteclibrary.ru/texsts/eng/stat/st1121eng.htm

I WANT TO PROGRAM IN DRAKON. IS THAT POSSIBLE?

By Stepan Mitkin, software developer, Norway.
stipan.mitkin@gmail.com

MEETING DRAKON

In summer 2011, I decided to investigate the enigmatic DRAKON programming language that had been created for the Russian Buran space programme. One of the first hits that the search engine showed was an introductory article on DRAKON. Having read the article, I got a strong feeling: this is what I had wanted for a long time! I had been looking for a way to efficiently represent algorithms and finally found it.

First, I tried to use Microsoft Visio to draw DRAKON diagrams. At that time, I got a pretty tough programming assignment at work. I had to write a reporting module for a complicated access system in some enormous piece of accounting software. The access system was so complex that no one knew exactly how it was working. Different members of the development team had different *opinions* on various parts of the code.

I carefully gathered this scattered knowledge and then had to capture it somehow as software requirements. DRAKON turned out to work well for that. A bunch of DRAKON diagrams clearly showed **all** possible use cases. The myth about a mystical and magically complex system disappeared.

The next challenge was to approve these requirements with Matthias, the business analyst who was responsible for that module. But how to explain the behavior of that tricky system to another person? The biggest problem with natural languages is that the listener forgets what was in the beginning long before the middle of the conversation. To make it worse, Matthias and I came from different countries and spoke different languages.

The situation seemed hopeless, so I simply showed the DRAKON diagrams to Matthias. There was no time to explain what DRAKON was and where it came from. But it worked again! It took Matthias only a few minutes to figure out the whole thing.

That assignment gave me confidence: my hunch did not let me down. DRAKON had real power.

The weak spot was Visio. Drawing DRAKON in Visio took too much time. So I decided to put together a small program that would specialize in drawing DRAKON flowcharts. This is how the DRAKON Editor project started.

WHAT IS DRAKON EDITOR?

DRAKON Editor is a desktop application for making DRAKON diagrams. It is a rather simple vector graphics editor with built-in DRAKON shapes.

In DRAKON Editor, the user builds diagrams manually of lines and DRAKON icons. This user experience is straightforward. No training is required for that. The disadvantage is that the rules of the DRAKON language are not enforced during drawing. The editor does have a function that checks correctness of the diagram, but the user has to click a button to run it.

DRAKON Editor offers a few more features on top of its basic editing engine:

- advanced editing mode;
- automatic icon size;
- navigation between diagrams.

ADVANCED EDITING MODE

In this mode, the disjoint primitives that comprise the diagram become temporarily connected. For example, if one drags a vertical line, all icons that sit on that line will move too. If that line hits other lines or icons, it will push them. This creates an illusion of physics that makes diagram elements feel tangible.

An important property of this advanced editing mode is that it does not alter the topology of the diagram. In other words, the user can arrange icons to make the picture look better, but he cannot damage the underlying algorithm. The advanced editing mode also makes sure that empty space between neighboring icons has the same thickness.

AUTOMATIC ICON SIZE

According to the rules of the DRAKON visual language, icons that are on the same vertical should have the same width. DRAKON Editor automatically adjusts the size of icons to their text content. The task of keeping the same icon width is automated by the editor, too.

NAVIGATION

Only a limited amount of information can fit in a single diagram. In order to cope with the complexity of real-world problems, we inevitably need to split them into parts of manageable size. Therefore, working with just one isolated diagram is rare. More often, we deal with a set of tightly related diagrams. This is why navigation between diagrams is so important.

DRAKON Editor offers the following means of navigation:

- Quick jump to a diagram by name.
- Browser-style go back and forward.
- Find places in other diagrams where a specific diagram is referenced.
- Go to the diagram from a place where it is referenced.

Good navigation creates a miracle. It turns a pile of unrelated diagrams into a well-connected system.

PROGRAMMING WITH DRAKON

Once DRAKON Editor was up and running, one natural question came to my head. Is it possible to build real programs with DRAKON? People already generate source code from flowcharts, why not try that with DRAKON?

The programmer would draw a DRAKON diagram and put snippets of code into icons. Those snippets would contain only simple sequential code, like calling functions. The logic of the algorithm, i. e. branching and looping would be controlled by the visual structure of the diagram.

That sounded like a solid plan and soon was implemented. The first programming language to produce source code for was chosen to be Tcl.

The code generator works like this:

1. Extract the graph of the algorithm from the vector image.
2. Check the rules of the DRAKON language on the way.
3. Build an abstract syntax tree from the graph.
4. Print out the tree using the concrete syntax of the chosen programming language.

Building the tree was the trickiest step. It felt like writing a disassembler. But after some time, the code generator was ready.

The next mission was to test it. Having unit tests was definitely not enough. There were too many possible combinations in the input data. A good test coverage meant nothing. There was a clear need for more diagrams to run the editor with.

The solution invited itself. From that point on, the rest of the editor was written in DRAKON-Tcl. So a great deal of DRAKON Editor itself is built with DRAKON Editor. Working with a lot of real-world diagrams provided enough material to strengthen the code generator. Slowly and painfully, the generator became reasonably stable.

And stability was not the only win. The proof of concept proved the concept. Not only programming in DRAKON was possible, but it was also fun!

After Tcl, more programming languages got support in DRAKON Editor: Python, C, C++, C# and Java. It was not sufficient, though. The community was demanding more languages than I had command of. Luckily, some bright developers joined the DRAKON Editor team and wrote plugins for even more languages.

These newly added languages brought with themselves an interesting observation. All of them are very, very similar!

There seems to be no conceptual difference between JavaScript, C++ and Python. Some developers may consider this blasphemy, but as long as the code generator is concerned, the only real way the mainstream programming languages differ is error messages.

DRAKON AND FUNCTIONAL PROGRAMMING

Functional programming has its infamous halo of elitism. Only those few who possess the secret knowledge of functional programming can exercise its power. Why is this power so unavailable to common people? Some blame for it the unusual syntax. Some claim that the very concept of pure functions belongs to a totally different universe.

Dutch erlanger Maas-Maarten Zeeman wrote the Erlang code generator for DRAKON Editor and proved otherwise. Yes, functional programming *is* different, but not much. DRAKON-Erlang quite visually showed two things:

1. Functions are good old algorithms. The only thing which is missing in them is loops. Functional programming relies on recursion instead of loops. So functional DRAKON is just a subset of the normal DRAKON. Take away DRAKON's loop facilities and you get functional DRAKON. All other features of DRAKON including silhouette work just as great.
2. DRAKON is a good friend to functional programming. DRAKON adds visual clarity to the strictness of functional programs. They retain all their advantages and gain readability.

Functional DRAKON is a discovery that has not gotten due attention yet.

DRAKON, FINITE AUTOMATA AND ACTORS

It is unclear why state machines (aka finite automata) did not make it to mainstream software engineering. They are rarely used outside of embedded programming and it's a shame. A state machine models behavior and is thus good for building interactive programs. (Programs that interact with humans or other programs, for example on the network.) Given their simplicity and obvious practical value, state machines should be everywhere. And yet they are not.

Here is how a typical state machine works. By default, it is sitting quietly and doing nothing. When a message comes, the state machine reacts to it. The reaction consists of two parts. First, the machine performs some useful work, such as calculation or signaling to the outside world. Then, the machine switches to its next *state*. The machine can choose between a finite number of states. For example, a light switch has two states: on and off. When the next message comes, the machine reacts again. What exactly the machine does depend on two things:

- the incoming message;
- the state which the machine is in.

Since the state was selected by the previous action, it is an explicit link to the past. That is why state machines are a good fit for programming behavior.

With a working code generator for DRAKON at hand, putting together DRAKON and state machines was enticing. I could not resist the temptation and extended the code generator to produce working state machines from DRAKON diagrams.

One DRAKON silhouette diagram is transformed into one state machine. Each branch of the silhouette becomes a state. The name of the branch is the name of the state. A Choice icon right under the branch header defines which messages can be accepted when the machine is in this state. Each Address icon at the bottom points at the next state.

The advantage of DRAKON here is that it shows both the algorithmic logic **and** state transitions on one visual scene! In other words, the useful work that the automaton performs and the way it selects the next state are drawn on the same diagram.

One of practical applications of state machines is actors. Actors is a way to model concurrency. Thanks to the ability to represent behavior, state machines make exceptionally good actors.

An application was written in DRAKON-C# to test the concept. In that application, DRAKON actors were able to handle parallel computation, input/output and the graphical user interface.

DRAKON EDITOR WEB

Making small apps that prove concepts is fun. Most of the time, at least. However, almost any programming idea looks good in a simple hello-world program. Only building a real-world software product entirely in DRAKON could tell if programming in DRAKON was a viable idea. Even though DRAKON Editor itself was partially written in DRAKON, it was not enough.

DRAKON Editor Web became the first widely available commercial software fully written in DRAKON. Despite the similar name, DRAKON Editor Web has no common source code with its desktop predecessor. The editing experience has been re-worked from scratch.

Productivity is the first strong point of DRAKON Editor Web. The user needs to make far fewer mouse clicks in order to build a diagram, because many routine editing tasks are automated. Modifying existing diagrams is also much easier. Another benefit of the new editing engine is that it does not allow to make an invalid DRAKON diagram. The rules of the DRAKON language are built into DRAKON Editor Web. They are automatically enforced. One can even start drawing DRAKON diagrams without learning the DRAKON language first.

These features have brought in significant complexity into the source code of the editor. And DRAKON seems to cope with that complexity just fine.

The server side of DRAKON Editor Web is implemented in DRAKON-Lua on the Tarantool platform. The front-end is written in DRAKON-JavaScript and runs in the browser. User input events are handled by a pipeline of DRAKON state machines.

CONCLUSION

So, programming in DRAKON is definitely possible:

- DRAKON diagrams can be turned into programs in various programming languages, static and dynamic, new and classical.
- DRAKON gives a boost to functional programming.
- DRAKON is good for interactive and concurrent programs based on state machines;
- DRAKON has been used to build a complex cloud-based application.

Does DRAKON offer a significant advantage in programming? Are we talking about a quantum leap in the software industry? I do not think so. DRAKON does not change the nature of programming. The programmer still has to give the computer an order for every little thing. Take a number from that variable. Multiply it by two. Is the result greater than some other variable? If yes, call procedure X. And so on. Even after DRAKON, programming remains manual labor. The programmer puts every brick in the castle of the software product by hand. DRAKON alone does not bring the bright future of computing where people give machines goals and machines figure out how to reach those goals.

And yet, DRAKON makes a difference. If an algorithm has neither loops nor branching, DRAKON does not add any value. But if the algorithm gets at least one "if" statement, the benefits of DRAKON start to appear. For example, the happy path through the program can be visualized with DRAKON's *skewer*. Another DRAKON's feature which is difficult to underestimate is *common fate*. Common fate shines even in simple algorithms.

When a procedure contains not one, but *several* "if" constructs, DRAKON's superiority becomes obvious. Don't even bother me with text source code when there are many conditions in one procedure. Interestingly, switching to DRAKON felt for me like no big deal. It is going back to text that is painful.

The worst thing about programming in DRAKON is a lack of tool support. DRAKON Editor cannot compete with full-blown IDEs like Visual Studio or Eclipse. Programming without a visual debugger and code completion is just wrong in the 21st century. Another missing feature is version control. The popular version control systems are designed to handle text. They are not efficient at working with structured data such as diagrams. In order to be useful to teams of developers, DRAKON needs a proper version control system and a tool for diff visualization and merging.

Programming in DRAKON works, and it works well. DRAKON is not a miracle, it is just a better way to build programs. But for that way to work, an ecosystem of tools needs to be created.

LINKS

1. DRAKON Editor for desktop. http://drakon-editor.sourceforge.net/
2. DRAKON. The Human Revolution in Understanding Programs. http://drakon-editor.sourceforge.net/DRAKON.pdf
3. DRAKON Editor Web. https://drakon-editor.com/
4. Smart editing mode in DRAKON Editor. http://drakon-editor.sourceforge.net/smart.html
5. DRAKON-Erlang: Visual Functional Programming. http://drakon-editor.sourceforge.net/drakon-erlang/intro.html
6. State Machines in DRAKON Editor. http://drakon-editor.sourceforge.net/auto.html
7. DRAKON, Actors and Message Passing. http://drakon-editor.sourceforge.net/actors.html

8. State Machines in DRAKON-Erlang. http://drakon-editor.sourceforge.net/erlang_auto.html
9. ERIL: a Visual Language for Data Modelling. ERIL is an attempt to apply the principles of DRAKON to entity-relationship and class diagrams. http://drakon-editor.sourceforge.net/eril.html
10. A DRAKON-C# application that shows off DRAKON state machines for concurrency. https://github.com/stepan-mitkin/actor-http

VIDEOS

1. **DRAKON. Flowcharts that have order. Part 1.** This video tutorial presents the basics of DRAKON with help of DRAKON Editor Web. https://www.youtube.com/watch?v=j-M_z_7_Nkw
2. **DRAKON. Flowcharts that have order. Part 2.** A video tutorial that explains silhouette. https://www.youtube.com/watch?v=Q0YBdX9xazw
3. **Visual functional programming with DRAKON-Erlang.** This is a presentation of DRAKON at Erlang User Conference 2015. https://www.youtube.com/watch?v=yZLedcnFA94