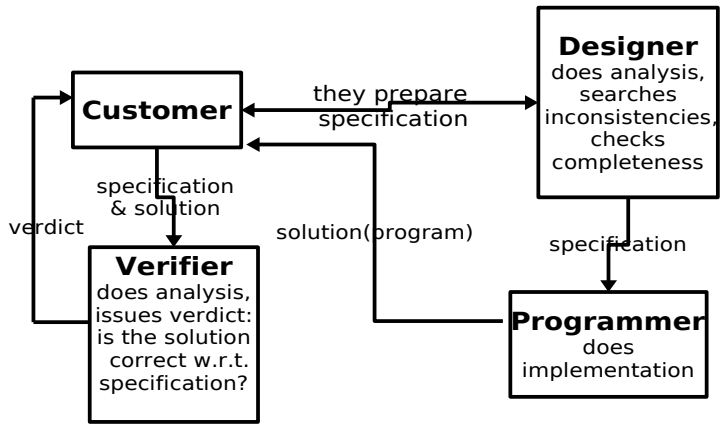
# Algorithmic Logic
i.e.
# Calculus of Programs
for
# specification and verification of software

Only a few think that programmers are creating the error-free programs. However, the majority believes in magic, in some spells... How to explain the belief that a program P will behave, the belief stemming from the fact that the program P has been tested? How to explain the faith in correctness of a program P, as a consequence of testing? Suppose that, the producer of a software executed it 1000 or 10000 times on some data. Suppose that nothing worrisome happened. Are we sure that that the program P will always terminate the computations and that the results are correct? There are no garanties. Is it possible to deliver some software together with a warranty of quality, of correctness. Yes, the warranty can be forwarded together with program. To be able to realize such a task one has to know and apply the calculus of programs.




## What can this calculus serve?

To create *specification* i.e. a set of algorithmic formulas. Such specification is reacher than Java's interface, it contains formulas expressing the requirements.
To record the arguments that arise during *verification.* Process of verification is nothing else but proving. Proofs are founded on axioms and rules of inference of program calculus (=algorithmic logic).

## Where does certainty comes from?

The semantical properties of programs can be expressed by algorithmic formulas. The valid formulas have proofs.
**Note**. The classical logic does not help.

## What do I find in this repository?

- Monographs on algorithmic logic.
- Set of articles.
- Set of examples.
- Introduction to SpecVer project on applications of program calculus.

## Are there better tools?

You can compare calculus of programs AL with: systems of algorithmic algebras(Glushkov 1965), Floyd-Hoare logic(1967-1969), calculus of weakest preconditions (Dijkstra 1975), dynamic logic (V. Pratt 1976).

Hoare logic is entirely contained in AL, calculus of Dijkstra is close to AL, contains some errors, calculus of Gluskov is equational not logical, hence it is not a complete system, dynamic logic is close to algorithmic logic – its language is too liberal in this sense that some expressions far from being algorithms are named programs, its deductive systems are: in one version not complete, in another version it imitate the system proposed by G. Mirkowska.

By comparing the range of applications we get the following information: *Hoare's logic* - several dozen proofs of small programs, for *Dijkstra's calculus* account – ca. 40 theorems on programs, *dynamic logic* - we know one proof of correctness of the algorithm calculating $x^n$. Using AL we can provide proof of a multi-page program. See more examples in the repository.

Moreover, it is algorithmic logic that has been validated as a tool of specifying data structures and classes of programs.

Moreover, all computers and compilers that assure the validity of axioms of AL are implementing the standard semantics of deterministic while programs.:

**Theorem** Let $\mathfrak{A}$ be the a virtual machine that enables exection of while programs. If the machine $\mathfrak{A}$ makes all axioms of program calulus valid and all inference rules sound, then the semantics implemented by $\mathfrak{A}$ is the standard one.

## How much is it worth?

Let's calculate: most of the cost of software comes from having to keep it (maintenance). Surely you can estimate the turnover with software. Say, it is some 50 miliards euro each year. The maintenance is about 40 percent of it. This is the cake to cut something for you. As long as you are able to prove the software's correctness w.r.t. to specifications.

*Program calculus in education.* Should not we discuss the curriculum again?

*Creation of new professions in IT industry.* Look at the diagram. New companies are appearing that limit their activities to creation of specification to be implemented. It is more difficult to find a firm capable to prove correctness of a software with respect to its specification. But time will show that software contracting companies will begin to behave similarly to big building investors. So we will need: specification architects, builders of development systems that meet the specifications and auditors (people checking whether the development product is a correct implementation of the software).

*Your guarantee of validity is important for ... years*? For ever.

## What next?

We have several ideas for further research projects using calculus of programs:

- SpecVer – This project aims at creating tools supporting the work of software engineers in various phases:
  - formulation of specifications,
    Specifications are annexed to civil-law contracts for software development.
    Such an attachment must not contain contradictory requirements.
    In addition, the incomplete specification exposes the software outsourcing party to the (large) costs resulting from the need to start over.
  - Verifying the compliance of the software product with its specifications,
    It is known that this process will never be automated. We propose to elaborate tools to help people in this process. Among others, a language and a proof-checker that checks whether a given text is a proof in program calculus.
- Project (for now without a name) intended to codify knowledge of algorithmics and discrete mathematics in a network of computers.
  Donald Knuth had a similar idea in his "The Art of Programming". Since Knuth began writing his monograph, it has been almost 50 years and science made a great progress. It is difficult today to gather in one work all the knowledge we know about algorithms. In addition, new facts can be included in the proposed project. We deeply believe that the use of AL will enable the development of algorithmic theories of many disciplines of discrete mathematics as algorithmic theories based on AL, instead on the predicate calculus, e.g.
  - Algorithmic theory of natural numbers (this formalized theory contains many proofs that are not doable in Peano first-order theory: halting property of Euclid's algorithm, non-halting of another algorithm as a consequence of last theorem of Fermat, …),
  - algorithmic theory of binary search trees,
  - algorithmic theory of Archimedean, ordered fields (in this theory one can carry proofs of correctness of several algorithms of numerical methods),
  - and many other theories.

---

Join us! Forward this information! Sell your knowledge of program calculus!

mailto: g.mirkowska@uksw.edu.pl , a.salwicki@uksw.edu.pl