Software development process and testing legislative acts

Jakub Kierzkowski[1]

[1]National Information Processing Institute, Warsaw, Poland, Jakub.Kierzkowski@opi.org.pl

Keywords
Regulatory Impact Analysis, testing, development model

1. ABSTRACT

The purpose of this paper is to describe legislative process using the terms of software engineering and to propose some new manners of detecting and avoiding defects in new law acts. The idea of software tester's approach to lawmaking comes from the observation that software development and legislative process are similar. Paragraph 2 contains an explanation of that similarity, definition of some basic software testing terms and a basic discussion on the topic. Paragraph 3 presents some basic software development models. In paragraph 4, one can find a brief description of the legislative process in the Republic of Poland. Paragraph 5 contains a discussion on concepts concerning making better law. I give two new ideas: dividing legislative bills into two documents and setting off the Law Testing Committee, that would lead to better validation and verification of legislative acts.

2. INTRODUCTION

A general process of creating a statutory act (or other law document given by legislative or executive authority) and a software development process, apart from obvious differences, have some key similarities. Both, statute and computer system, start as an idea (that solves a problem, meets a need, a must, etc.) which is later put into life. More generally, a legislative can be seen as a 'factory of law', and described as any other factory – how it is organized, how does its production process look like etc. More particularly, a group of people write a sequence of characters (signs) which can be discussed, modified, and in the end is approved and comes into life. What legislators write is a natural language, and what software developers write is a programming language. A document created using a programming language is called a code. More generally, any (natural) language is a code itself.

A legislator (lawmaker, rule-maker) and a software developer (programmer, software engineer) constitute a pair showing similarities or things referring to both law making an computer system making. A legislator can be called a 'law (bill) developer' or 'law engineer', a software developer give rules, according to which a computer will behave. Also a computer programming language and a natural language (especially formal, used by lawyers or authorities) constitute such a pair. Obviously, neither programmers' nor lawmakers' work consists of writing only. Both, law making (law development) and software development, processes include decision making, problem solving, verification and validation. Differences between the law development and the software development show in the way those processes are organized. The reason of the differences come not only from the difference between software and law itself, but also from different approaches to their development models. Those models are discussed in paragraphs 3 and 4.

Verification and validation are terms related to software testing, a sub-discipline of software engineering. The International Software Testing Qualifications Board (ISTQB©), international software testing qualification certification organization, in its glossary – *Standard Glossary of Terms used in Software Testing (Version 3.1)* – give the following definitions of those terms.

Definition 1. Verification – confirmation by examination and through provision of objective evidence that specified requirements have been fulfilled.

Definition 2. Validation – confirmation by examination and through provision of objective evidence that the requirements for a specific intended use or application have been fulfilled.

In other words, verification is the process of ensuring if the software is made the proper way, while validation is ensuring the proper software is being made.

Both law and software, as made by humans, can be done incorrectly. As discussed in paragraphs 3 and 4, in general there is no particular moment when a final version of a bill is verified and validated, while software testing teams play an important role in software development life cycle. The aforementioned software testers' glossary gives definitions of three basic terms referring to incorrect human actions.

Definition 3. Error (mistake) – a human action that produces an incorrect result.

Definition 4. Defect (fault, bug) – a flaw in a component or system that can cause the component or system to fail to perform its required function, e.g. an incorrect statement or data definition.

Definition 5. Failure – deviation of the component or system from its expected delivery, service or result.

The definition of an error can be applied to bill development without any further comment. To help apply 'defect' and 'failure' to statutes, we need two more definitions, cited after (Frith, 2012).

Definition 6. Loophole – an ambiguity that can be used to circumvent or otherwise avoid the express or implied intent of the law. It is a technicality that allows a person or entity to avoid the scope of a law or restriction, to get around it in a way that was not intended by the legislators who put the law or restriction into place, without directly or technically breaking the law.

Definition 7. Unintended (unanticipated, unforeseen, unexpected) consequences (results) – outcomes that are not the ones that were intended by a purposeful action.

A loophole in an legislative act is an example of a defect and every usage of the loophole is a failure. Also, every unintended result is a failure of the act (one can observe that the definitions of unintended consequence and failure actually treat about the same thing: unexpected result). Note that a defect, which is a result of a developer's error, does not have to lead to a failure, e.g. when the developer specifies what is to happen in an impossible case – such situation would simply not occur and thus will have no (unexpected) result.

Although general rules of developing a bill can be similar for all or most developed countries, each country has its own particular model. Differences between those models can result in differences in amounts of defects and failures in law of the countries. Comparison of the quality of law in different countries is not an interest of this paper. Paragraph 4 describes the situation in the Republic of Poland. Paragraph 5 contains a discussion on how to organize a legislative acts development model so that the acts would be given proper validation and verification, similarly to software being tested.

3. SOFTWARE DEVELOPMENT MODELS

Many different approaches to software development have been used since the early years of information technology. This paragraph mentions only few of them. Those will form a background for the discussion in the next paragraphs. The models are not described in full, as this is not the aim of this paper, and one can find rich literature about this topic. A stage of testing and its role in the process is discussed.

The first important software development model is the waterfall. It consists of several steps, that occur one after another, always in the same sequence (like water falling on rock steps of a cascade waterfall, which is the explanation of the model name). Typically, the waterfall model steps are:

1. Defining software requirements
2. Analysis resulting in models and business rules
3. Designing of the system resulting in software architecture
4. Implementation (programming)
5. Testing

In this model, the whole computer system, with all its complexity, is tested at once, when it is ready. Any defects are repaired after the testing is finished (the process steps back to step 4), and another testing process is proceeded after all defects are repaired. The processes of testing, repairing and re-testing are repeated several times. In this approach, bugs (especially caused by mistakes made in documents created in steps 1-3) are found relatively late (implementation can take months or years, depending on the system), which makes repairing them difficult and expensive, related to other development models. A way of coping with this problems of the outdated model is the V-model.

The V-model can be considered as an extension of the waterfall model. A common type of V-model uses four test levels, corresponding to the four development levels. The name of the model comes from the letter V shape: the development stages are the left side and the test stages are the right side of it. The steps are:

1. Business requirements
2. System requirements
3. High level design
4. Low level design
5. Coding
6. Unit testing
7. Integration testing
8. System testing
9. Acceptance testing

When V-model is the one applied, acceptance tests can be designed or prepared when its corresponding phase – business requirements – just ended and the system requirement step is in progress. Similarly, system testing is designed after system requirements are documented, integration tests are designed after high level software design is ready. Low level design, coding and unit testing can be made by the same person – a developer, who designs, implements and test their own code. Early preparation of test levels can result in finding mistakes (especially mistakes in design) relatively early. Still, the testing process takes place after the software is finished.

A much different approach show agile software development models, a modern example of an iterative-incremental development models. Iterative-incremental models use a series of short development cycles (iterations), that last one or few weeks. In each cycle a small amount of work is made, using a small waterfall, V-model or similar model for designing, implementing and testing a small part of the developed computer system. Each cycle ends up with a working software delivery. The aim is to make the developed computer system evolve every cycle. Testing takes place when a new part of computer system is ready – during the same iteration it was designed and implemented. Defects found just few days after they were produced, have small influence on the rest of the system, which makes them easy and cheap to repair.

Testing within a software life cycle model

Another document created by ISTQB©, *Certified Tester. Foundation Level Syllabus (Version 2011)*, gives some rules related to software testing, that are independent from a software development model. In any model, there are several characteristics of good testing:

1. For every development activity there is a corresponding testing activity.
2. Each test level has test objectives specific to that level.
3. The analysis and design of tests for a given test level should begin during the corresponding development activity.

One more rule can be added: testing does not end when the developed system is deployed – e.g. validation of a deployed system can show, that the users' needs have changed, or the users simply found bugs, that had not been found by the software testers.

4. LEGISLATION IN POLAND

The legislature in Poland is a bicameral parliament consisting of a lower house (Sejm) and a Senate. Article 118 of the 'Constitution of the Republic of Poland' states:

1. The right to introduce legislation shall belong to Deputies, to the Senate, to the President of the Republic and to the Council of Ministers.

2. The right to introduce legislation shall also belong to a group of at least 100,000 citizens having the right to vote in elections to the Sejm. The procedure in such matter shall be specified by statute.
3. Sponsors, when introducing a bill to the Sejm, shall indicate the financial consequences of its implementation.

The proposition of a new law is considered in three readings in the Sejm. It starts form the Marshal of the Sejm, who decide whether the first reading is to be proceeded by the whole house or by its committee. The bill is then considered by committee(s) between first and second reading. During the committee's consideration, the bill can be amended. The second reading is held before the Sejm. After that, the bill can be considered by committees again – in that case, amendments are debated by the committees and the bill together with the committee's report is sent back to the Sejm for the third reading. If the Sejm does not decide to send the bill to a committee, the suggested amendments are discussed and voted in the third reading. After the third reading the bill is sent to the Senate.

The reading in the Senate can end up in three ways. One is accepting the bill without amendments. In that case, the Marshal of the Senate sends back the bill to the Marshal of the Sejm, who sends it to the President of the Republic of Poland.
The second possibility is amendment of the bill by the Senate. Then, the Sejm accepts or rejects the amendment and the Marshal of the Sejm sends the bill to the President.
The Senate can also request for cancellation of the bill. The request can be accepted or rejected by the Sejm. In the latter case, the Marshal of the Sejm sends the bill to the President.

The President of the Republic of Poland can sign the bill (new act) without further consideration, can ask the Constitutional Tribunal whether the bill violates the Constitution or send the bill back to the Sejm.

If the Constitutional Tribunal states the bill is unconstitutional, the bill is canceled. If it states the bill is partially constitutional, the President signs the bill with some parts canceled. If the tribunal states the bill does not violate the Constitution, the President signs the bill.

If the President decide to send the bill back to the Sejm, the lower house can amend the bill or send it to the President back without amendments – the President has to sign the act then, or cancel the bill.

All this process (described in a simplified form) is more complicated than the software development models. Trying to compare this process to the software development, one can observe the following stages:

1. Defining requirements, analysis and implementation (before the actual legislative initiative)
2. Discussions and amendments
3. Testing and decision making

Testing here means searching for loopholes and anticipate the results of the new law. The first and second steps require good will from the persons involved in the process for the bill to be tested. In step 3, the President (one person) can test the bill or ask the Constitutional Tribunal (15 judges) to test it. Defects found by the President on this stage (the acceptance tests) are expensive to repair and defects found by the tribunal require setting off a new process, with a new bill draft.

Amendment are, in theory, made to improve or repair a bill. But they cannot be treated as testing, because they can cause new defects – in fact, amendments are a part of the implementation process, similarly to bug fixes in the software development process.

However, one element of testing (anticipating the results) is required by the Constitution. As cited earlier, sponsor of the bill have to indicate the financial consequences of its implementation.

The Council of Ministers' bills

The discussion above omits one important situation: legislative initiative of the cabinet. In practice, most of the bills are prepared by the Ministers or the Cabinet. The Statute of the Council of Ministers, see *Regulamin pracy Rady Ministrów* in Polish, specifies the cabinet bills development process. Its key features are public consultations and obligatory Regulatory Impact Analysis (throughout this article called 'RIA'). RIA in general is a document, that should be created before introducing a new regulatory act in any OECD member country. Its role is to provide an anticipation of outcomes and potential impact of the new law. RIA in Poland refers to cabinet bills only, as it is mentioned only in the aforementioned Statute. The document shows expected results of the proposed bill, concerning social and economic affairs. In particular, according to the Statute, RIA in Poland contains:

1. who the act is going to affect,
2. information about the public consultation results,
3. results of analysis of the bill influence, especially on the budget, employment, competitiveness of the economy and condition of companies,
4. sources of funding the cost of the proposed act,
5. data used in the analysis.

In practice, many Minister's bills do not become the Cabinet's bills, but to make the bill be processed faster, it is proposed as a Members' of Parliament bill. On the on hand, RIA (together with public consultation) can be a good early-stage testing tool, on the other, it is easy to by-pass. What is more, many amendments are made after RIA is completed which makes RIA outdated during the parliament's readings.

Another disadvantage of the Polish implementation of RIA is the fact, that RIA is prepared by the same persons as the bill itself, and no other RIA can be prepared (by third party, someone distanced to the bill). This stands in opposition to software engineering approach, where software testing (except low-level testing done by the programmer) of a part of the computer system is proceeded by other than its author team members. Moreover, legislators writing a draft bill and RIA tend to see only the advantages of their draft. This results in RIA being a form of and advertisement of the proposed bill.

Public consultation of the Cabinet's bill play a similar role to user acceptance tests in software engineering. The difference is that a draft bill is put under public consultation, while user acceptance concerns a ready or almost ready to use computer system.

5. CONCEPTS AND CONCLUSIONS

Testing in law development process in Poland includes public consultation and Regulatory Impact Analysis. Parliament readings of bills and the Presidents decision should also include validation and verification. This not much enough to state that the new law in Poland is tested as thoroughly as needed, and that is tested in a similar way a software is. Ease to avoid public consultation and RIA (as mentioned in the previous paragraph) makes testing law only an option, a sign of good will. From a software tester's point of view, law testing should be a formal part of law development process. There are few ideas on how to achieve this state.

One of them is constituting the State Council, proposed by Janusz Kochanowski (Kochanowski, 2005). The Council would prepare RIA for all bills and check any formal criteria a bill should meet. The Council was supposed to be an independent body, similar to the Constitutional Tribunal, but working on bills *ex ante*, not on acts *ex post*.

Another idea, also suggested by Kochanowski, is the usage of the sunset provisions, *'according to which a passed legal act becomes automatically null and void unless a political will of its retention is expressed within a specified period of time'*. A political will of retention of a faulty act makes an occasion to amend the act, fixing the defect. A lack of such will would result in vanishing the faulty law.

The Kochanowski's State Council would opinion bills before the Sejm readings. From a software tester's point of view, that is a right stage for testing the general aim and idea of the bill, but RIA should be made also in the end of the parliament's process. I suggest two major changes in the law development process.

One is a change in the artifacts. When a software is developed, analytic documents are developed, describing both the business and technical requirements, and then there is the implementation stage (no matter if it is the implementation of the whole system in the waterfall model or implementation of one particular use case in an agile model). Similar division may be made in bill-making – first an analysis, the requirements, second the bill itself. Discussion and voting during the parliament readings would refer separately to the requirements, and separately to the bill, first validating the purpose of the bill, second – verifying whether the bill meets the requirements. Both the analysis document and the bill would be amended by the parliament.

The second change is constituting the Law Testing Committee in the Sejm, which role would be to update RIA after each stage of amendments of both the bill and the bill analysis. Such up-to-date RIA document would be an equivalent of integration and system tests of a computer system.

Reformulation of the law development process to make it more similar to software development processes can make detecting and avoiding defects in acts easier. Furthermore, it can make the whole process easier to maintain and manage, making it also more transparent. Transparency of the legislative process is a must in a democratic state ruled by law.

6. REFERENCES

Frith, R. (2012, October 11). *Preventing and avoiding loopholes and unintended consequences in legislation*. Retrieved from: http://www.ncsl.org/documents/lsss/spotloophole1.pdf.

International Software Testing Qualifications Board (2016). *Standard Glossary of Terms used in Software Testing (Version 3.1)*: ISTQB© Glossary Working Group Judy McKay (Chair), Matthias Hamburg (Vice-Chair). Available at: http://www.istqb.org/downloads/send/20-istqb-glossary/186-glossary-all-terms.html.

International Software Testing Qualifications Board (2011). *Certified Tester. Foundation Level Syllabus (Version 2011)*: Thomas Müller (chair), Debra Friedenberg, and the ISTQB WG Foundation Level. Available at: http://www.istqb.org/downloads/send/2-foundation-level-documents/3-foundation-level-syllabus-2011.html.

Kochanowski, J. (2005). Deregulacja jako pierwszy etap reformy systemu tworzenia prawa. Ius et Lex, 1/2005(III). Retrieved from http://www.kochanowski.pl/pub_iusetlex_1_2005_deregulacja.pdf.

Regulamin pracy Rady Ministrów. Monitor Polski. M.P. 2016 poz. 1006. Available in Polish at: http://isap.sejm.gov.pl/Download?id=WMP20160001006\&type=2.

7. AUTHOR'S BIOGRAPHY

Jakub Kierzkowski (Ph.D.) – software tester and researcher at the Laboratory of Intelligent Systems, a part of the National Information Processing Institute, since 2013. He received his MSc (2009) and PhD (2016) degrees in mathematics at the Faculty of Mathematics and Computer Science at the Warsaw University of Technology. He has several years of experience in software testing. His areas of interest include software and law development models comparison, software testing, linear algebra and numerical methods.

https://www.linkedin.com/in/jakubkierzkowski