Rem: Revista Escola de Minas
ISSN: 0370-4467
editor@rem.com.br
Escola de Minas
Brasil

D. Forti, Tiago L.; Longhin, Gustavo C.; da Silva Forti, Nadia Cazarim; V. Requena, João Alberto
Agility based software development for truss design
Rem: Revista Escola de Minas, vol. 67, núm. 3, julio-septiembre, 2014, pp. 259-264
Escola de Minas
Ouro Preto, Brasil

Agility based software development for truss design

Desenvolvimento de programa de computador para projeto de treliças baseado em tecnologias ágeis

Tiago L. D. Forti
Simworx Engenharia,
Pesquisa e Desenvolvimento,
Campinas - São Paulo - Brazil
forti@simworx.com.br

Gustavo C. Longhin
Simworx Engenharia,
Pesquisa e Desenvolvimento,
Campinas - São Paulo - Brazil
longhin@simworx.com.br

Nadia Cazarim da Silva Forti
Professora e Pesquisadora da Pontifícia Universidade Católica da Campinas,
Parque das Universidades,
Campinas-SP, Brasil,
nadiacazarim@yahoo.com.br

João Alberto V. Requena
FEC,Universidade Estadual de Campinas,
Cidade Universitária "Zeferino Vaz",
Campinas-SP, Brasil
requena@fec.unicamp.br

Abstract

This article describes the development of an application for structural analysis of tubular trussed girders. Such structural components are widely used on roofs for which long span components are a necessity, e. g., supermarkets, distribution centers, etc. The application supplies hints to the project engineers as to the most economic solutions for a combination of span, loads, and other characteristics. Software development techniques are also explored. Such techniques can be divided into two large groups. In the first one, which is most widely known and used, software development complies with very rigid planning, where processes are more important than skills. This methodology is known as the Rigorous Development System (RDS). The second group, called Agile Development System (ADS), is conceived as an option for those not aligned with the RDS rules. This text describes the experience of an ADS based software development for truss design. In Section 2, the basis of ADS is presented. Section 3 describes the truss design application and civil engineering related concepts. Section 4 brings an example illustrating the application. Conclusions are in Section 5.

Keywords: steel structures, design automation, agile programming.

Resumo

Este artigo descreve o desenvolvimento de um programa de computador para análise estrutural de treliças tubulares. Esse tipo de estrutura é amplamente empregado em coberturas de grandes vãos, como as de supermercados, centros de distribuição, etc. O programa indica ao projetista as soluções mais econômicas para uma combinação de parâmetros de vão, cargas, entre outros. O artigo também explora técnicas de desenvolvimento de software. Tais técnicas podem ser divididas em dois grandes grupos. No primeiro, mais empregado e conhecido, o desenvolvimento é baseado em um planejamento rígido, em que processos são mais importantes que habilidades. Essas metologias são conhecidas como Rigorous Development System (RDS). O segundo grupo, chamado de Agile Development System (ADS), é concebido como uma alternativa àqueles que não se alinham com as regras do RDS. O texto descreve uma experiência de desenvolvimento, baseado em ADS, de um programa de projeto de treliças. Na Seção 2, as bases da ADS são apresentadas. A Seção 3 descreve o programa e conceitos de engenharia civil relacionados. A Seção 4 traz um exemplo ilustrando o programa. Conclusões são apresentadas na Seção 5.

Palavras-chave: *estruturas metálicas, automação de projeto, programação ágil.*

1. Introduction

The steel construction industry has evolved considerably since the hand designed crafted, assembled structures of the XIX century. It has evolved to a manufacturing process using automated production lines yielding ready to go parts containing the latest technologies of its field.

Along with the last four decades of the steel construction industry evolution, another industry is evolving with astonishing speed and is greatly contributing to all of the industrial evolution, the Information Technology industry (IT).

The pace of development found in IT evolution, combined with the challenges to which IT is exposed, provides scientists with motivation for the development of new technologies, methodologies, rules, and processes in the process of defining the way information is exploited. Among all IT specialties, Software Engineering (SE) is an example of IT related disciplines which evolved because of such motivations.

Software Engineering was first established as a well defined science, with accurate scope and non ambiguous contexts back in the 80s. One can think of software development evolution as an adaptation of methodologies found in other engineering fields, i.e. Civil Engineering, Mechanical Engineering, etc. Analogies amongst those fields fomented the definitions of man hour specification, budgetary schedules, deliverable schedules, etc. Nevertheless, contrary to such disciplines, SE does not count on such non abstract, totally palpable products such as buildings, engines, bridges etc. SE lacks a well defined deliverable, and this is certainly its main drawback, and still, it is a very strong motivation for all types of evolution.

Since SE defines such an adverse environment, its evolution is addressing all its deficiencies with constant modifications, re-designs, accommodations, adaptations, among others. Its initial methodologies reflected a strong necessity of protecting the contractors against the software developers. Processes were totally dependent on strong bureaucratic tools, schedules, patterns, and measurements, i.e., a very rigid development system. Methodologies compliant to such restrictive requirements are identified as Rigorous Development Systems (RDS). Rational Unified Process (RUP) is an example of RDS methodology, Jacobson *et al.* (1999) and Kroll & Kruchten (2003). RDS methodologies are unquestionably very appropriate to most of the large scale software projects, where hundreds of developers work on the same subject and where final integration without the help of well defined procedures would be humanly impossible.

RDS was then taken as the solution for all software development projects, until a new requirement arouse, speed. Terms like “time to market” became as common as Class Diagrams on any software specification. The intrinsic qualities of RDS methodologies compromised its performance whenever time to market was an issue.

RDS should therefore, evolve to a more flexible, less restrictive methodology, which could address more effectively issues such as time to market, non static specifications, ever changing requirements, change of schedules, competitors actions, etc. Naturally, RDS practitioners themselves evolved to adapted methodologies developed, and in-house procedures were lapidated to the specific contexts. After a few experiences, the community realized they were no longer doing RDS and decided that a new specification should be sketched.

This gap is then fulfilled by the establishment of the Agile Development Systems (ADS), which, according to its authors, better addresses the new challenges of the modern software development environment. Other segments of the Software Engineering community, however, question their arguments. This judgment is not in the scope of this work.

In this state of development for both software engineering and steel construction industry, this work goes towards the direction of sharing an experience the authors had in developing an application for such a highly technological segment, which is the steel construction industry, immersed on the paradigms of ADS and actually sticking to its rules.

The application's main functionality is to process a set of possible configurations of trusses for a given span and load cases. Combinations of all possible designs are automatically projected and subjected to structural analysis and design. Afterwards all the results are sorted according to a specific parameterization, and the best qualified configurations may then be subjected to a deep analysis. The application receives as input the span it must cover as well as its accidental loads according to Brazilian specification / standardization ABNT NBR8800 (2008), and whatever possible configurations to vary, including geometry type, height, planar or 3 dimensional truss. Details of these parameters are presented in Section 3.

The application was written in Delphi and lots of pre-existing classes were used, also contributing to the development process, since most of those classes had already been tested and validated.

Playing by ADS rules allowed the authors to meet the very restrictive schedule of one week. Extensive use of pair programming and Test First techniques were also exploited. Specific topics throughout the document are dedicated to both SE and Steel Industry. Such terms are explained with more detail in upcoming topics.

The application is then validated by an application reference in the field of Steel Structures analysis and its correctness is confirmed.

2. Agile development systems

Agile Development Systems (ADS) is a methodology conceived for the development of software (Highismith (2002)). It is a methodology used as a tool for better developing software. As any methodology, ADS has its own sets of rules, tools, and processes and is based on a few paradigms. What most differentiates the ADS methodology from other Software Engineering methodologies is the nature of its paradigms.

The paradigms upon which ADS is conceived are translated in a few, very important rules. While practicing ADS, these rules are addressed by a set of techniques. A variety of ADS methodologies explores the paradigms in different ways, but agility as its main goal, and it never neglects motivation. All ADS methodologies, as seen in Highismith (2002), advocates: ***Individuals and Interactions*** over processes and tools; ***Working software*** over comprehensive documentation;

Customer collaboration over contract negotiation; ***Responding to change*** over following a plan.

The first statement addresses the fact that, while using RDS, processes and tools overwhelm creativity. RDS states that, as long as you keep your work according to what is planned, chances are that the project will succeed. ADS assumes a position whereby Individuals and the interaction among them are more effective than following the plan.

For ADS, some processes and tools are certainly necessary, but after a point processes and tools start inhibiting creativity, which by no means should be inhibited.

The second one states that one cannot find a software developer who has never been involved in a project where the effort for complying with documentation requirements surpassed the necessary effort for delivering the first executable for that project. It is very common that one spends more time documenting what the software does than actually coding its functionalities.

The third statement is also very reasonable, since arguments on validating the projects deliverable based on contracts happen very often. In this approach, the protection the contractor has is the contract, which makes its negotiation mostly an adverse event, where developer plays against contractors; this is certainly not useful for the development process. A more symbiotic customer/developer relationship must always be the main objective.

For ADS, this can be accomplished with a more collaborative arrangement involving both developers and contractors, in a sense that both parts share the responsibility for the resulting schedule, planning, budget etc. Having the contract as the sole entity for validation of the project execution is definitely not a good option.

The last statement addresses the discipline of playing according to what was previously planned.

RDS states that you have to plan the work very carefully and then work on the plan. Nevertheless, ADS understands that such a strict rule of playing according to the plan prevents the development process from adapting to changes, which, by the ADS perspective is more important than following a plan.

At the end, if a project does not adapt to possible changes, at best, one will have an application that works correctly but is worthless, since the context in which it was conceived, is not the same as that which is delivered.

Further discussions on the subject of Software Development technologies are not in the scope of this work. The message here goes in the direction of sharing a very effective experience the authors had on working with a specific set of ADS rules on the development of a scientific code.

2.1 - ADS Practices

A few ADS practices, which are actually defined by eXtreme Programming (Astels et. al. (2002)), or XP, address the motivations of team work. Some of the paradigms found for XP concern the sharing of knowledge which should be as widespread as possible in any technical environment.

The more information a team shares, the more chances are that this team will be a winning one. Therefore, XP defines the Pair Programming practice.

The XP community understands that, for a wide variety of scenarios, two engineers (or programmers) working on the same subject, each one in its own workstation, are less effective than the work both engineers perform while working at the same workstation. In our experience of developing the truss design application, we could observe that our work was more effective because of this approach.

Another proposal XP brings concerns the coding methodology. XP proposes the Test First philosophy, i.e. the first actions the programmer takes aims on testing, if the planned model actually delivers what it was first conceived to accomplish.

These two approaches are described in the following topics.

2.1.1 - Pair Programming

Pair programming is not only the configuration of two programmers sharing the same workstation. While sharing the same code, not only programming is performed in pairs but plenty of other things are shared by those two programmers. Certainly, one aspect where Pair Programming effectively adds up to the quality of the code being developed, lies on the fact that two different persons are solving the same problem, and, what is more important, both tend to agree on the adopted solution. With this, both persons are exposed to different approaches while solving a specific problem. The sharing of experiences is enormous (Astels *et. al.* (2002)).

Another aspect which deserves some attention concerns error generation. Two persons reading the same code may better identify errors, from simple misspelling to misunderstanding of the algorithms.

XP also mentions the “double fun effect”, whenever a code is being developed concurrently by two persons. If one of them enters a low productivity mood, there are chances that the other is warming up since, for instance, the upcoming subject is its preferred one.

In the development of the truss design application, the authors experienced these aspects. The authors experience while practicing pair programming was very interesting.

All expectations driven from XP's pair programming explanations were in fact observed.

2.1.2 - Test First

According to XP perspectives, the traditional testing plans to which all software developers are used to, those based on final definitions of the class model according to well written use cases, somehow lack an effective way of contributing to the quality of the software developed. It solely verifies that the model is correct, and it behaves according to that which was documented and implemented on the classes that comprise the model.

In the XP proposal for testing, the chronological order of task execution is switched, contrary to mainstream testing.

One first writes the tests, and then implements the code, different from the

former, where one first implements the code and then the testing.

XP proposes the Test First philosophy, where the first thing a programmer should implement is the tests which verify the functionalities of its to-be implemented code. By doing so, programmers, while elaborating test cases, also validate their expectations for the elaborated solutions. While revisiting those solutions, some missing points can be clarified and better addressed.

A process with such iterative characteristics effectively adds up to the quality of the solution.

The reference Astels et al. (2002) brings a more in depth information on the XP subject.

3. The truss design application

The truss design application developed has a graphical user interface as shown in Figure 1.

The application is developed in Delphi language running on Windows operating system.

The application automatically generates the truss geometry, applies the external loads, computes the efforts acting on each element and designs them. These steps are described in the next topics.

3.1 - Truss geometry

The trusses analyzed are girders for long span roofs. Some types of geometry are implemented. Figure 2 illustrates the available geometry types: Warren, Mounted Warren, Pratt, and Inverted Pratt. The trusses may also be planar or multiplanar trusses (Figure 3).

For defining the truss geometry, the following data is required:

- the span to be covered by the truss; the truss height;
- the truss width of multiplanar trusses. It also indicates the distance between neighbor trusses, both planar and multiplanar;
- the modulation of the truss which is defined by the angle of the diagonal elements.

The application adopts 5 different modulations within the admissible diagonal angle values;

- the geometry type: Warren, Mounted Warren, Pratt, or Inverted Pratt; planar or multiplanar.

Actually, the user informs a range for the truss height and a range for the diagonal angle.

The application then generates all the possible geometries and designs all of them. In fact, that is the main goal of the application: to determine the optimal geometry for a given set of span, truss width, and external applied loads.

3.2 - External loads

The user must provide the accidental loads and their coefficient in combination cases.

In general, for the purpose of roof systems, two major external load cases are considered, based on wind loads. The first is for the pressure wind case, and the second is for the suction wind case. More load cases could certainly be adopted. Once these values are provided, the application automatically applies these loads to the superior chord of the truss. The self weight loads are, of course, computed by the application itself during the design process.

3.3 - Computing internal stresses

The stresses acting on each element of the truss are computed in linear elasticity.

Employed is the direct stiffness method (DSM), also known as the displacement method or matrix stiffness method (Gere & Weaver (1965)).

The main assumptions of this method are:

- the truss is a framed structure;
- materials are elastic and have linear behavior: small displacements and small deformations;
- validity of the Bernoulli principle, i.e. a planar cross section remains planar after loading;
- validity of the effect superposition principle;
- elements are modeled as spatial beams with axial, torsional, and bending rigidity.

The DSM computes the displacement and rotation of each node in the structure and the internal forces of each element: normal force, shear forces, bending moments and torsional moment.

For solving the DSM system of equations, a Cholesky direct decomposition solver is used (Burden & Faires (2010)).

A band storage matrix is utilized (Burden & Faires (2010)).

The equations are sorted to reduce the band matrix and increase the speed of the Cholesky solver.

3.4 - Design of elements

The elements must be designed to support the forces acting on it.

In the developed application, the elements are designed and checked against the Brazilian Specification ABNT NBR8800 (2008), which is concerned with steel structure analysis, design and construction.

For the design, all the forces are taken into consideration: normal force, bending moments, shear forces, and torsional moment.

Connections are not designed or checked in the current version of the application.

3.5 - Validation

Several tests were applied to verify the correctness of each part of the developed application for both graphical user interface and kernel functionalities.

Moreover, some classes used in the development of this application had been previously validated for their use in other applications. Finally, the application was validated comparing its results with the commercial structural analysis and design application SAP (2000).

The comparison of several examples confirmed the correctness of the application. Details of the validation task will not be explored in this text.

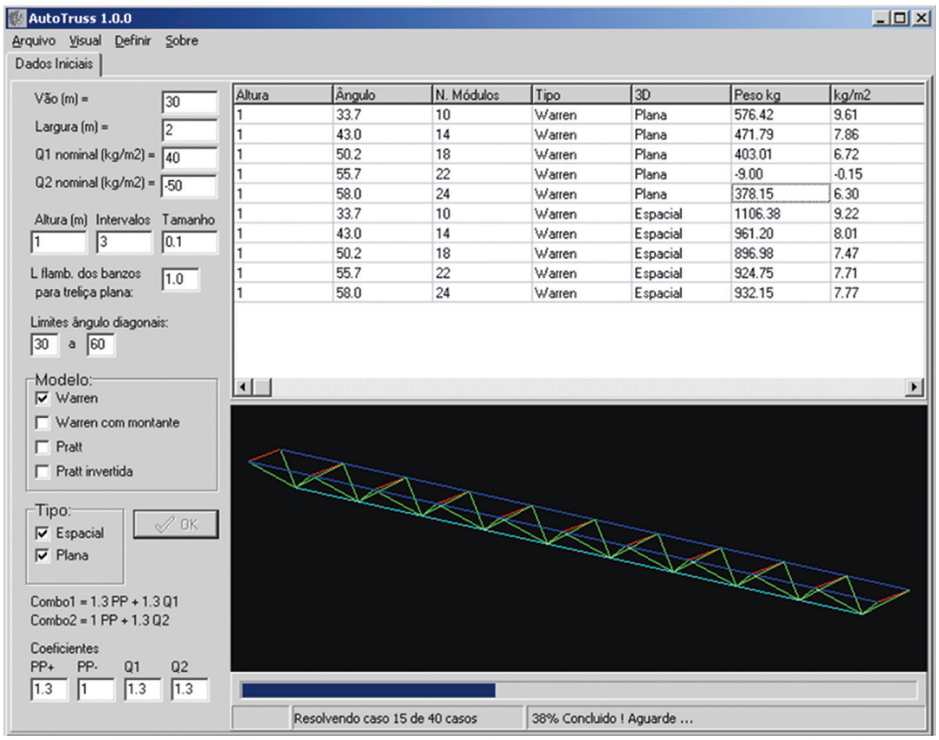

Figure 1
Graphical user interface of the truss design application

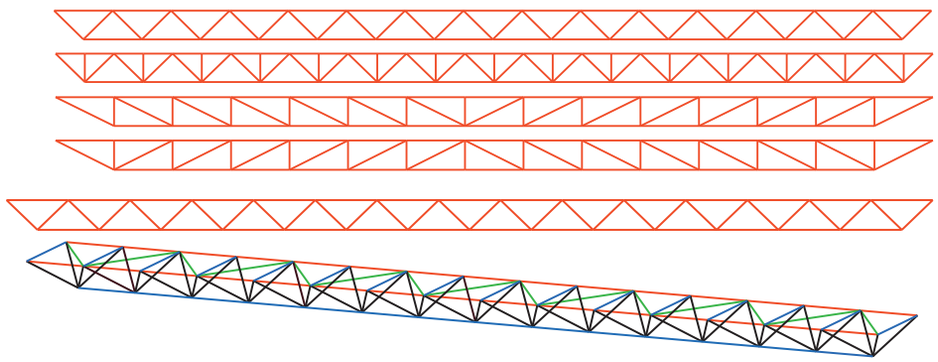

Figure 2
Geometry types: Warren, Mounted Warren, Pratt, and Inverted Pratt

Figure 3
Planar and multiplanar truss geometry

4. Numerical example

A numerical example is presented to illustrate the functionalities of the developed application. Let us take the following input data:

- Span length = 40 m;
- Geometry:
  - o Truss width = 2.5 m;
  - o Truss height from 2 m to 4 m varying of 10 cm, i.e. 2, 2.1, 2.2, … , 3.8, 3.9, 4 m
  - o Admissible diagonal angles between 30º and 60º;
  - o The four available geometry types: Warren, Mounted Warren, Pratt, or Inverted Pratt
  - o Planar and multiplanar geometries
- Load cases:
  - o Q1 = 40 kgf/m² (vertical downward);
  - o Q2 = -50 kgf/m² (vertical upward);
- Combination cases:
  - o Combo 1=1.4 Self weight + 1.4 Q1;
  - o Combo 2 = 1.0 Self weight + 1.4 Q2;
- Design of elements using circular hollow steel section.

The available possibilities for the truss geometry lead to a number of 840 different geometries to be analyzed. The application runs on multithreading, processing two cases at the same time. Running on a personal laptop computer, the application took 16 minutes to process all the cases. The computer has an Intel Core 2Duo P8600 2.4 GHz processor and 4 Gb of RAM memory. The operating system is a Windows 7 professional 64 bits.

After the cases are run, the application sorts them based on their weight, from the lightest to the heaviest truss. The application has also the possibility of sorting the trusses based on their prices. In order to use this option one needs to provide the price of each circular hollow steel section. It was not the case in this example. The lightest truss obtained is presented in Figures 4 and 5. Its geometry is given by: truss width = 2.5 m; truss height = 2.5 m; diagonal angle = 56.3º which gives 12 modules; mounted Warren geometry type; multiplanar geometry.

The weight of this truss is 1603 kg which gives a weight per square meter of roof of 8.01 kg/m². For instance, the worst solution obtained produces 40.4 kg/m².

It is important to note that the best solution obtained is for the given load and its combination cases, span length and truss width. For other data, different solutions would be obtained. However, the example illustrates how a great number of possibilities (840 cases) may be analyzed in a few minutes when design automation is employed.

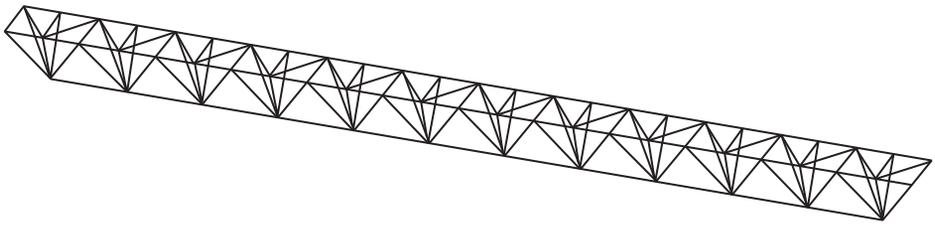

Figure 4
The best geometry found for the example – perspective view

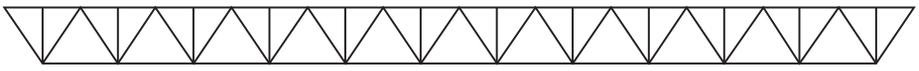

Figure 5
The best geometry found for the example – 2D view

5. Conclusions

This work describes an experience of a scientific code development under the methodology of Agile Development Systems (ADS). A review of software engineering techniques such as ADS and Rigorous Developments Systems was presented. The text describes the rules of ADS employed in the development of a truss design application.

The authors recognize that the use of the ADS methodology was very important for the success of this project which had a very tight schedule of only one week to be accomplished. Techniques like pair programming played a very effective role in the development process.

Along with pair programming, test first techniques contributed a lot to the development process. One can say that test first techniques sharpen the engineers focus on the problem being solved. It enforces just-in-time development, i.e. implement just what you need now, and do not implement anything for the future. Such practice certainly yields a code which is simpler when compared to generic implementations.

ADS techniques are certainly very suitable for the development of scientific code. The paradigms upon which ADS is built are very aligned to the paradigms dictating the way knowledge should be shared in the academic environment. At least, in the academic environment, if one is playing by the ADS rules, it will probably benefit from it.

The application was implemented from some already validated structural analysis classes that were extended to this project. A graphical user interface was also developed for helping the input of data and the analysis of results. A numerical example is presented.

Finally, the article contributes with a witnessed experience of how different disciplines like computer science and civil engineering can be combined to produce an effective solution.

6. Acknowledgments

The authors would like to thank the V&M do Brasil company for the support and funding and the professors of the School of Civil Engineering (FEC) in State University of Campinas (UNICAMP) for their provided help and availability.

7. References

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. ABNT NBR8800:2008. Projeto de estruturas de aço e de estruturas mistas de aço e concreto de edifícios. Brasil, 2008.

ASTELS, D., MILLER, G., NOVAK, M. Practical guide to eXtreme programming. Computer Software - Development. Prentice Hall PTR, first edition, 2002.

BURDEN, R. L. FAIRES, J. D. Numerical analysis, ed. Brookes-Cole, 2010.

COMPUTER AND STRUCTURES INC. SAP2000 Users guide. Computers & Structures, 2000.

GERE, J. M. WEAVER, W. Jr. Analysis of framed structures. D. Van Nostrand Company, 1965.

HIGHSMITH, J. Agile software development ecosystems. Computer software - Development. Addison Wesley, 2002.

JACOBSON, I., BOOCH, G., RUMBAUGH, J. The unified software development process. Addison-Wesley Professional, 1999.

KROLL, P. and KRUCHTEN, P. The rational unified process made easy. Addison-Wesley Professional, 2003.

Received: 25 July 2013 - Accepted: 25 July 2014.