The Search for Genericity in Graphics Recognition Applications: Design Issues of the Qgar Software System

Jan Rendek, Gérald Masini, Philippe Dosch, and Karl Tombre

LORIA, B.P. 239, 54506 Vandœuvre-lès-Nancy Cedex, France
{Jan.Rendek,Gerald.Masini,Philippe.Dosch,Karl.Tombre}@loria.fr

Abstract. This paper presents the main design and development issues of the Qgar software environment for graphics recognition applications. We aim at providing stable and robust implementations of state-of-the-art methods and algorithms, within an intuitive and user-friendly environment. The resulting software system is open, so that our applications can be easily interfaced with other systems, and, conversely, that third-party applications can be "plugged" into our environment with little effort. The paper also presents a quick tour of the various components of the Qgar environment, and concentrates on the usefulness of this kind of system for testing and evaluation purposes.

1 Introduction

The design of document analysis systems implies mastering a number of complex issues. Firstly, and obviously, the designers have to put together a set of stable, robust implementations of state-of-the-art methods and algorithms, to perform the various processing and recognition tasks needed in document image analysis: Image preprocessing and filtering, binarization, text-graphics separation, vectorization, feature extraction, character and symbol recognition, etc. Best practices in software engineering, as well as the need for performance assessment and evaluation tools, lead to the requirement that each function, class, or task should be associated with data sets for testing and evaluation purposes.

Secondly, the system must be easy to use (and hence provide an intuitive user interface), easy to install, and well documented. The various tools and applications should be as generic as possible, so that the system does not solely work for a tiny subset of document analysis tasks.

The third requirement is probably the most important: The system has to be open, versatile, and interfaceable. Even if the best care is given to all possible aspects of a document analysis problem, and even if the development team has unlimited resources (which is obviously never the case), some applications will inevitably not be foreseen in the system. Had they been foreseen, other teams may still want to use their own algorithm or method for a given problem, and may want to interface their tools with the general system, without being constrained by a rigid framework, non-standard formats, etc. Conversely, the

S. Marinai and A. Dengel (Eds.): DAS 2004, LNCS 3163, pp. 366–377, 2004.

document analysis system, even if it provides by itself a large coverage of document analysis needs, may have to become a component of a larger system. Once again, interfacing and openness are vital requirements.

These design and integration problems are well-known to many researchers in document analysis, and more generally in image processing. One of the most famous initiatives to deal with them was the Image Understanding Environment (IUE) [1, 2], which aimed at providing a complete environment for all kinds of image processing applications. Many useful ideas came out of this project, but the resulting environment is extremely complex and hard to fully master.

Of course, a number of specific, *ad hoc* document analysis systems have been implemented, but the design problems are undoubtedly of another order of magnitude when the system has to be versatile, as the flexible integration of all the system components becomes a crucial problem on its own. Still, there has been clear successes in some subareas. For instance, there is good know-how in building flexible and versatile OCR and page reader systems, taking into account segmentation, feature extraction, classification, and various linguistic post-processing steps [3]. Extending the concept to various business documents, Dengel *et al.* have also developed a very mature system, capable of adaptation to different types of documents [4, 5]. Other specific application domains have their mature systems, such as bank check processing software [6], table recognition [7], or forms processing [8]. Some effort has also been made in building flexible environments for handwriting recognition [9].

However, when it comes to graphics-rich documents, most systems remain very context-dependent and are just able to solve a very specific problem. Few teams made attempts to build generic tools. One of the few early initiatives in this direction comes from Pasternak [10], who proposes a hierarchical and structural description coupled with triggering mechanisms for interpretation. Another example is the DMOS method proposed by a team at IRISA [11], and based on syntactical tools to describe the domain knowledge.

Our team is involved since several years in the development of such an open software environment (called Qgar) according to a software engineering viewpoint [12]. In some aspects, it is similar to the Gamera framework [13], which provides domain experts with a high abstraction-level, user-friendly environment to build, to tune, and to compare document analysis applications. Its plug-in architecture allows a quick integration of new state-of-the-art methods. Qgar also offers a common environment where applications and third-party contributions can easily coexist and interact. However, it mainly aims at providing the community with stable robust and generic document analysis methods, either as a set of off-the-shelf components or as an application builder toolkit [14–16]. Lately, we have successfully integrated Qgar with other tools and systems, as part of a French national project on software for document analysis [17].

This paper gives an overview of the main design issues for such a system. In section 2, we describe the general concepts which have guided us throughout the development. We then propose a brief tour of the environment (§ 3), before concentrating on the important issue of testing (§ 4), in a software development perspective, performance evaluation, and benchmarking.

2 Underlying Concepts

When developing the Qgar software platform, we had three main objectives in mind:

- Ease the development of new image processing applications by grouping together implementations of graphics recognition methods and common data structures,
- offer an environment to tune and evaluate the performances of our image processing applications and compare them with other existing ones,
- demonstrate and distribute our know-how in the field of graphics recognition.

The Qgar platform is an ever evolving project, getting enriched by all developments required in the course of our research. In some way, it could be considered as a snapshot of our current know-how about graphics recognition and document image processing. This explains why we did not orient our development effort towards a particular functionality. We paid instead a particular attention to several (software engineering) issues, such as:

- Interoperability: Our platform has to easily interact at various levels with external systems. A high level of interoperability is a great asset to build document analysis chains integrating third-party applications. It is a great help for evaluation purposes, as it allows us to compare our work with others within a common framework. Interoperability is also a key feature for distribution as it makes part or whole of the platform available to other (research) groups with little effort. For example, the integration of Qgar applications into the DocMining platform[1], for heterogeneous document interpretation, raised no major difficulty [17].
- Reusability: As stated above, the software reuse rate must be maximized from one research work to another. This implies that the platform architecture and implementation must be highly modular, as, in particular, only specialized features from a given application are useful to build the next one. The thinner the granularity of the corresponding software components is, the more they may be reused. The architecture should also be flexible enough to allow a quick integration of existing software.
- Extensibility: Since the platform is intended to evolve according to our needs, it must be extensible and easy to maintain. Adding new features must not interfere with already implemented functionalities, and updating or extending existing code to fulfill more precise or slightly different purposes must be simple. A failure to achieve such requirements would lead to the construction of concurrent versions of the platform, specifically designed for different aims. This would contradict the very purpose of our project.

[1] This project is supported by a consortium including four academic partners, PSI Lab (Rouen, France), Project Qgar (LORIA, Nancy, France), L3I Lab (La Rochelle, France), DIUF Lab (Fribourg, Switzerland), and one industrial partner, GRI Lab from France Télécom R&D (Lannion, France). It is partially funded by the French Ministry of Research, under the auspices of RNTL (*Réseau National des Technologies Logicielles*).

We had to conform to several principles to meet these requirements, and first of all enforce software quality procedures. As capitalizing on previous developments is one of our objectives, we try to produce high quality software. Software poorly documented or written without thorough design is not likely to be adapted to purposes different from the initial ones. If rigorous design and implementation are essential to produce high standard code, they must be complemented by good documentation and extensive test suites to result in long lasting code.

Documentation is essential when coming to code reuse or interoperability. Maintaining a component, modifying it, or making it interact with other software artifacts requires information on its design and implementation. To ensure the best documentation coverage, the full source code of Qgar is therefore self-documented. The documentation is embedded in code comments and can be automatically extracted using *Doxygen*[2] to produce online browsable HTML pages or high-quality hardcopy documents. This documentation is complemented by several manuals, including design papers and tutorials.

Testing is another aspect of software quality that is essential in the Qgar context. Regression and performance tests are required to detect errors when integrating, upgrading or refactoring pieces of code. Considering the modularity of the platform architecture, unit testing appeared to be the best methodology to design test suites. Each Qgar module is thus associated with its own independent set of regression and performance tests. They provide great help to detect the propagation of errors when integrating a new module, to evaluate the performance gain when refactoring, and much more...

C++ has been chosen as programming language, so as to combine the object-oriented programming paradigm, which is appropriate to modular and easy-to-maintain software production, with several other advantages of the language, all conforming to our requirements. When efficiency becomes more important than modularity, it allows the use of well-known C programming tricks, cutting across the computational overhead implied by an object-oriented design. C++ is also, more or less, an industrial standard, giving access to a lot of existing software.

The Qgar software environment is distributed[3] under the terms of the GNU Lesser General Public License (LGPL) and, for the user-interface only, the Q Public License (QPL). In this way, our work can be freely used, modified and redistributed by others, and we can also reuse existing code published under any LGPL compatible license. In fact, the choice of such a license helps to improve the software quality in the long run, thanks to the feedback provided by the users: Comments, bug reports and corrections, or even requests for new features.

3 System Overview

The Qgar software system is divided into three parts: QgarLib, a library of C++ classes implementing basic image and graphics recognition operators, QgarApps,

[2] http://www.doxygen.org/

[3] The Qgar system may be downloaded from its web site at http://www.qgar.org/.

an applicative layer constructed from QgarLib components and providing graphics recognition applications, such as text-graphics separation or vectorization, and QgarGui, a user interface dedicated to result visualization and application supervision. The whole system represents about 60,000 lines of C++ code. A particular attention has been paid to the support of standard formats (PBM+, DXF, SVG), high-quality documentation, configuration facilities (using `autoconf`/`automake`), and support of Unix/Linux operating systems.

3.1 The QgarLib Library

The object-oriented paradigm is based on data encapsulation, *i.e.* describing abstract data types and their interfaces with the clients. However, image (or graphics) processing generally concerns collections of operators, *i.e.* procedures, to be applied to images. There are two solutions to implement such procedures in an object-oriented way. The first one consists in implementing them as function members of the classes of the objects on which they operate. For example, the convolution of an image by a Gaussian could be defined as the `gaussianConvolution` function member of the `Image` class (that describes images). Such a technique implies ongoing modification of the interfaces of existing classes, whenever new methods are added to the library. Moreover, the interfaces of the corresponding classes grow in proportion and become too large to be efficiently used.

We thus preferred a more pragmatic solution, based on a very simple idea. `Image` becomes the base class of a hierarchy, and an operation that processes an image is implemented by a constructor of a new derived class: The convolution by a Gaussian is thus implemented as the constructor of the `GaussianConvolution` class deriving from the `Image` class. In this way, there is no need to modify the existing hierarchy and classes, and different methods to perform the same conceptual operation can be easily implemented through derived classes of an abstract class. This principle has two main advantages. On the one hand, software parts separately designed may be easily integrated in the Qgar software, even when written in C. On the other hand, designers as well as clients of the library can write compact and easy-to-read code, which meets the understandability requirements [18].

To illustrate the philosophy of our approach, Figure 1 shows excerpts from a text-graphics segmentation application, implementing our own variation of Fletcher and Kasturi's algorithm [19, 20]. In the figure, we did omit the computations themselves – obviously, such an algorithm cannot boil down to merely calling a succession of constructors – but the example should give the general idea of the ease of writing a graphics recognition algorithm using the classes provided by QgarLib. As one can see, the code remains quite understandable. This is an idyllic view as, in the general case, additional parameters are needed to accurately perform the different image processing steps. However, we firmly believe that the general philosophy holds for most low-level and intermediate-level operations in graphics recognition applications.

```
// File containing the initial (binary) image
PbmFile pbf("an_image.pbm");
// Load the image
BinaryImage initImg(pbf);

// Prune small connected components from the image
PrunedConCompBinaryImage prunedImg(initImg, 5);

// Extract connected components from the resulting image
ConnectedComponents components(prunedImg);

// Do some filtering to get the textual components:
// The result is given by variable txtComponents
...
// Create the binary image of the textual components
ComponentBinaryImage txtImg(components, txtComponents);

// Subtract the textual components from the initial image
// to get the graphical components
initImg -= txtImg;

// Save the image of textual components
PbmFile txtf("txt_image.pbm");
txtImg.save(txtf);
// Save the image of graphical components
PbmFile graphxf("graphx_image.pbm");
initImg.save(graphxf);
```

Fig. 1. Excerpts from a text-graphics segmentation application.

QgarLib classes implement most data structures and algorithms to be handled when designing a document analysis system (graphical objects such as points, segments, arcs of circle... image processing operations such as convolutions, mathematical morphology... and so on) as well as different kinds of utilities, especially to store graphics or image data in files using various formats. The introduction of XML in the library has become inevitable, as it is now a standard format which is accepted by an increasing number of tools. Moreover, import and export of SVG data cannot work without it. However, to avoid compatibility problems between various software licenses, an external XML library cannot be directly integrated. We have therefore adopted a solution allowing the use of any SAX XML parser. It is based on a set of interfaces providing an indirection level between the Qgar system and the external parser. These interfaces have to be implemented by concrete classes acting as wrappers between an XML API and the system. Wrappers for the Xerces and Qt parsers have already been implemented.

3.2 The QgarApps Applications

QgarApps is a set of stand-alone applications representing the applicative layer of the Qgar system. Some of them are simple wrappings of QgarLib classes implementing the construction of useful objects, essentially intended to serve generic purposes, whereas others are more sophisticated and specific programs. About ten applications are available, such as binarization, thin-thick segmentation, text-graphic segmentation, vectorization, etc.

An application is run from a command line and works as a black box, to which clients have to supply arguments (input/output images, thresholds values, etc.) suited to the result they want to obtain. Communication between applications is file-based, in order to simplify the implementation. The interoperability and integration capability of the Qgar system relies on these applications, which can be sequentially invoked to build processing chains performing document analysis tasks. Chains performing the same task according to different methods can be quickly elaborated and can be conveniently compared, for performance evaluation for instance.

However, an application is practically useless if clients are not supplied with a description of the application itself (the tasks which are performed, the corresponding methods, etc.), of its parameters, and of the syntax of the command line. This information is given by a XML file associated with the application. In this way, an "external" application, like the DocMining platform [17], can integrate any Qgar application, provided that it is able to parse the corresponding XML description to "understand" the behavior of the Qgar application and to get the kind of data the Qgar application must be fed with. Conversely, a Qgar application can be interfaced with any application designed according to the same principle like, for example, the graphical user interface QgarGui (see next section), or the Qgar web site, that proposes dynamic demos of Qgar applications. In the latter case, a HTML form is automatically generated from the description file to get parameter values supplied by clients to run selected applications (Fig. 2).

3.3 The QgarGui User Interface

QgarGui provides users with a friendly environment to visualize and interact with every step of a document processing chain. It is completely independent of the other parts of the platform, though it may be used to control any Qgar application. As previously mentioned, an application is integrated within the interface with the only help of its description file. This description is used to automatically generate dialogs to tune the application parameters (for a specific issue) and to make online help directly available thanks to the embedded application documentation. For example, Figure 3 shows the activation dialog created from the description of an application which performs a binarization. Even when designed apart from the Qgar platform, an application can thus be very easily integrated into the platform, without recompiling any part of the platform itself, as long as a description is provided. In this way, the interface

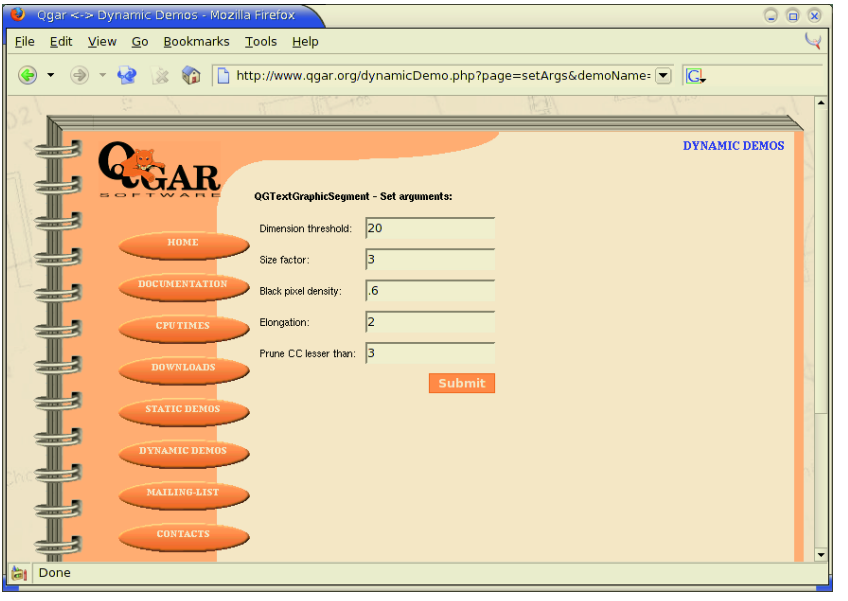


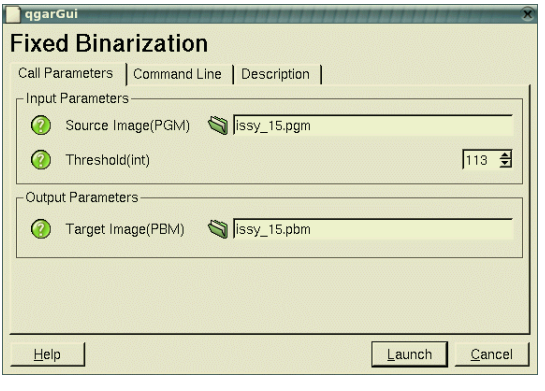
Fig. 2. HTML form automatically generated by the Qgar web site for the application performing text-graphics segmentation.

```
<application>
  <descr>
    <name>Fixed Binarization</name>
  </descr>
  <paramlist>
    <param name="Source Image"
           flag="in"
           type="grayscale" />

    <param name="Target Image"
           flag="out"
           type="binary"
           default="_.pbm" />

    <param name="Threshold"
           flag="thr"
           type="numeric"
           default="127"
           min="0"
           max="255" />
    </paramlist>
</application>
```

(a)



(b)

Fig. 3. Automatic integration of an application of binarization: From the XML description of the application (a), QgarGui generates a dialog box (b) to tune its parameters and then to run it.

may also be appropriately used as a flexible tool to compare the efficiency of different applications/methods delivering the same kind of result.

QgarGui supports bitmap and vectorial image formats, such as subsets of DXF or SVG. It offers features to apply and control image processing tools, to

display results (including zooming and image superimposing facilities), and to manually correct them (*i.e.* add missing results, delete or alter erroneous ones...) if needed. In particular, special operations to edit for both bitmap and vectorial images are supported, such as copy and paste of bitmap images, creation and modification of vectorization components, etc. The interface is implemented in C++ using the Qt framework[4] and is distributed under the QPL license. This makes it easy to be upgraded or customized by anyone interested in its features.

4 Testing

As mentioned in § 2, testing is an important part of the Qgar platform development and it takes place on two levels, with different purposes.

The first level is related to the developer viewpoint. It is based on a white box approach with thin granularity, each test focusing on several code lines, up to a full C++ class. These tests are implemented according to the unit testing approach, as defined by the extreme programming methodology [21], and are directly integrated in our software development process. For each C++ class, a "twin" test-class implements as many test-function members as features of the original class. A test-function is a set of assertions which automatically check if the code of the corresponding feature runs correctly. Once written, these tests are handled by a dedicated framework and can be run individually or together, using a separate API, *CPPUnit*[5]. Every time new code is introduced in the platform, or old code is updated, all tests are run.

Such a systematic test policy offers several immediate advantages. It provides developers with an instant feedback on the design of any new code. If the tests are hard to write, it generally means that the proposed design is either improper or too complex. It also improves the modularity of the platform: Each new class is implemented along with a direct client class, the test-class, that constrains developers to focus on the services offered by this class, rather than on their implementation.

There are also long term benefits. The set of all the test-classes provides a solid regression test suite. This eases the process of refactoring [22], as tests can be run every time a code change occurs, ensuring that no other feature has been broken. The code quality is improved on the long run: Every time a glitch is found, a new test is introduced to ensure that it will not appear again.

The second testing level, called functional testing, is related to the client viewpoint and focuses on the ready-to-use services offered by the software, that is to say Qgar applications, in our case. Functional tests are written once an application is ready to be run. Their are designed to validate that the application fulfills the tasks it is supposed to perform and to evaluate its performances. The tests are completely independent of the application implementation. They see the application as a black box and proceed only by comparing its inputs and outputs. This approach is really interesting when combined with the method of

[4] The Qt framework is available at http://www.trolltech.com/.
[5] http://cppunit.sourceforge.net/

integration of third-party applications in the platform (see § 3.2). Indeed, the tests defined for an application can be reused for any application designed for the same purpose. This makes Qgar a good tool to compare different methods or different implementation of methods.

Unfortunately, unlike unit tests, functional tests are complex to set up, and a lot of work remains to be done to define a general, complete framework to perform such tests. However, their importance is crucial to the field, as they are at the basis of numerous performance evaluation schemes. We have therefore started work on adding a number of evaluation methods and tools to the Qgar platform, providing, among other things, support for the VEC format used by various evaluation methods [23] and for the document image degradation model proposed by Kanungo [24]. These tools were used for our participation in the organization of the first symbol recognition contest, held in Barcelona during GREC'03 [25]. We also plan to integrate other evaluation methods designed by our group, notably the vector-to-ground-truth matching method developed by Xavier Hilaire within our group [26].

5 Conclusion

In our work to develop the Qgar software environment, we do our best to aim at genericity, ease of use, and interoperability with other software systems. This led us to the adoption of rigorous design principles, which have been detailed in this paper. The result is a reference platform, available to any person interested in building a document analysis system. The environment can be used as such, obviously with its limitations and constraints. It can also be easily enriched by "plugging in" document analysis tools separately developed, on the unique condition that a description is provided for each new tool to be added, as explained in section 3.2. Conversely, the whole environment can itself become a component of a larger project, using the same principles, as we have proved with the DocMining project.

In addition to the interest it may be for the community to freely download state of the art document analysis tools and methods, to use them, and to eventually integrate them into one's own application, such a software environment provides an efficient platform for running benchmarks, comparisons and performance evaluations. While software-oriented test facilities, as described in section 4, have been introduced in the platform, we have started adding support for testing and benchmarking at the functional level. This effort will be emphasized in the coming time, as the Qgar environment provides a good basis for conducting thorough evaluation campaigns on various document analysis methods.

References

1. Kohl, C., Mundy, J.: The development of the Image Understanding Environment. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, Seattle, WA (USA). (1994) 443–447

2. Lerner, R.A.: The Image Understanding Environment progress since IUW'97. In: Proceedings of the 1998 DARPA Image Understanding Workshop, Monterey, CA (USA). (1998)
3. Baird, H.S.: Anatomy of a versatile page reader. Proceedings of the IEEE 80 (1992) 1059–1065
4. Dengel, A.R., Klein, B.: smartFIX: A requirements-driven system for document analysis and understanding. In Lopresti, D., Hu, J., Kashi, R., eds.: Proceedings of the 5th IAPR International Workshop on Document Analysis Systems, Princeton, NJ (USA). Volume 2423 of Lecture Notes in Computer Science., Berlin, Springer-Verlag (2002) 433–444
5. Dengel, A.R.: Making Documents Work: Challenges for Document Understanding. In: Proceedings of 7th International Conference on Document Analysis and Recognition, Edinburgh (Scotland, UK). (2003) 1026–1035
6. Gorski, N., Anisimov, V., Augustin, E., Baret, O., Maximov, S.: Industrial bank check processing: the A2iA CheckReader. International Journal on Document Analysis and Recognition 3 (2001) 196–206
7. Shamilian, J.H., Baird, H.S., Wood, T.L.: A retargetable table reader. In: Proceedings of 4th International Conference on Document Analysis and Recognition, Ulm (Germany). (1997) 158–163
8. Niyogi, D., Srihari, S.N., Govindaraju, V.: Analysis of printed forms. In Bunke, H., Wang, P.S.P., eds.: Handbook of Character Recognition and Document Image Analysis. World Scienti'c (1997) 485–502
9. Cracknell, C., Downton, A.C.: A Handwriting Understanding Environment (HUE) for rapid prototyping in handwriting and document analysis research. In: Proceedings of 5th International Conference on Document Analysis and Recognition, Bangalore (India). (1999) 362–365
10. Pasternak, B.: Adaptierbares Kernsystem zur Interpretation von Zeichnungen. Dissertation zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften (Dr. rer. nat.), Universität Hamburg (1996)
11. Coüasnon, B.: DMOS: A generic document recognition method. Application to an automatic generator of musical scores, mathematical formulae and table structures recognition systems. In: Proceedings of the 6th International Conference on Document Analysis and Recognition, Seattle, WA (USA). (2001) 215–220
12. Dosch, P., Ah-Soon, C., Masini, G., Sánchez, G., Tombre, K.: Design of an integrated environment for the automated analysis of architectural drawings. In Lee, S.W., Nakano, Y., eds.: Document Analysis Systems: Theory and Practice. Selected papers from the 3rd IAPR Workshop, DAS'98, Nagano (Japan), November 4–6, 1998, in revised version. Lecture Notes in Computer Science 1655. Springer-Verlag, Berlin (1999) 295–309
13. Droettboom, M., MacMillan, K., Fujinaga, I.: The Gamera framework for building custom recognition systems. In: Proceedings of the Symposium on Document Image Understanding Technologies, Greenbelt, Maryland (USA). (2003) 275–286
14. Dosch, P., Tombre, K., Ah-Soon, C., Masini, G.: A complete system for analysis of architectural drawings. International Journal on Document Analysis and Recognition 3 (2000) 102–116
15. Tombre, K., Ah-Soon, C., Dosch, P., Habed, A., Masini, G.: Stable, robust and off-the-shelf methods for graphics recognition. In: Proceedings of the 14th International Conference on Pattern Recognition, Brisbane (Australia). (1998) 406–408

16. Tombre, K., Ah-Soon, C., Dosch, P., Masini, G., Tabbone, S.: Stable and robust vectorization: How to make the right choices. In Chhabra, A.K., Dori, D., eds.: Graphics recognition – Recent advances. Volume 1941 of Lecture Notes in Computer Science. Springer-Verlag, Berlin (2000) 3–18
17. Clavier, E., Masini, G., Delalandre, M., Rigamonti, M., Tombre, K., Gardes, J.: DocMining: A cooperative platform for heterogeneous document interpretation according to user-defined scenarios. In: Proceedings of 5th IAPR International Workshop on Graphics Recognition, Barcelona (Spain). (2003) 21–32
18. Meyer, B.: Object-oriented software construction, second edition. The Object-Oriented Series. Prentice-Hall, Englewood Cliffs, NJ (USA) (1997)
19. Fletcher, L.A., Kasturi, R.: A robust algorithm for text string separation from mixed text/graphics images. IEEE Transactions on PAMI 10 (1988) 910–918
20. Tombre, K., Tabbone, S., Pélissier, L., Lamiroy, B., Dosch, P.: Text/graphics separation revisited. In Lopresti, D., Hu, J., Kashi, R., eds.: Proceedings of the 5th IAPR International Workshop on Document Analysis Systems, Princeton, NJ (USA). Volume 2423 of Lecture Notes in Computer Science., Springer-Verlag (2002) 200–211
21. Beck, K.: Extreme programming explained: Embrace change. Addison-Wesley Professional, Reading, MA (USA) (2000)
22. Fowler, M.: Refactoring – Improving the design of existing code. Addison-Wesley, Reading, MA (USA) (2000)
23. Phillips, I.T., Chhabra, A.K.: Empirical performance evaluation of graphics recognition systems. IEEE Transactions on PAMI 21 (1999) 849–870
24. Kanungo, T., Haralick, R.M., Baird, H.S., Stuezle, W., Madigan, D.: A statistical, nonparametric methodology for document degradation model validation. IEEE Transactions on PAMI 22 (2000) 1209–1223
25. Valveny, E., Dosch, P.: Symbol recognition contest: A synthesis. In: Selected Papers from the 5th International Workshop on Graphics Recognition (GREC'03, Barcelona, Spain). Lecture Notes in Computer Science (2004) to appear.
26. Hilaire, X.: A matching scheme to enhance performance evaluation of raster-to-vector conversion systems. In: Proceedings of 7th International Conference on Document Analysis and Recognition, Edinburgh (Scotland, UK). (2003) 629–633