Patterns in software design

Peter Vogt

Received: 9 October 2018 / Accepted: 5 March 2019 / Published online: 28 March 2019
© The Author(s) 2019

Knowledge is pattern

In the past 30 years I have worked in various agencies conducting research and developing software in application fields dealing with radiative transfer modelling in the ocean, atmosphere, and vegetation; meteorology; atmospheric chemistry; marine oil spill and fish stock monitoring; cancer risk assessment; satellite sensor design and extraction and assessment of environmental information from remote sensing data. What I have learned from these various assignments is that the process of acquiring knowledge is the result of pattern recognition and pattern analysis. Every time we are introduced to a new application field we are confronted with new terminologies, topics, concepts and ways of thinking. Our first and natural instinct is trying to understand the new system by ‘bringing order into the chaos’, meaning that we try to detect structures driving that system. Pattern recognition can be seen as the cognitive process of delineating the underlying governing rules while pattern analysis is based on the application of these rules in a specific context. In this sense, pattern recognition and pattern analysis are of generic nature, of fundamental importance in any learning process leading to knowledge generation in any domain.

In the context of this special edition on landscape pattern I will outline a more generic conceptual view of pattern exemplified to software design. My goal is to improve the outreach and impact of software in landscape ecology. At some risk of generalization, the readers of this journal can be categorized into three groups:

1. software developers who work in ecology at least sometimes,
2. ecologists who also write software at least sometimes,
3. landscape ecology researchers, planners, and other stakeholders who use software.

Writing from the perspective of a member of group 1, my intended audience is the members of group 3 who aspire to become more involved in group 2. Most landscape ecologists are trained to develop or adapt software to solve their own specific problems, but many may also appreciate some insights and practical advice from a software developer’s point of view.

The purpose of software

End-users usually perceive software simply as a helpful tool to automate workflows. Yet, the role of software in reproducibility of science is a crucial aspect as well. The software Fragstats (McGarigal and Marks 1995), seen by many as a game changer for the

P. Vogt (✉)
European Commission, Joint Research Centre (JRC), Via
E. Fermi 2749, I-21027 Ispra, VA, Italy
e-mail: peter.vogt@ec.europa.eu

advance and acceptance of landscape ecology, is a prime example for the importance of software as a reference, communication and assessment tool. But software is also essential for transparency of research: if not implemented in software, then fewer people can use, test, evaluate, and critically scrutinize the outcomes of a given method or approach. Implementation in a software application allows a method to pass the test of a wider community of people and contexts. Moreover, software leads to increased outreach: writing a paper is rarely enough to make a method useful, because the large majority of the readers do not have the possibility of transferring a formal definition from a paper into a usable workflow. Accompanying any new approach with an implementation for end-users will significantly increase the community of users and the dissemination and popularity of that approach. A tight integration of basic research, development of applications and educational training is considered to be a key issue in landscape ecology (Wu and Hobbs 2002).

The task of a software developer is to program/replicate a system in order to simulate its behavior and allow retrieving key system properties. Software design can be divided into a conceptual phase, which requires a thorough understanding of the rules driving the system, and the technical implementation phase, which requires adequate programming skills to implement these rules in a way that is tailored for the anticipated user community. In addition, the value of a particular software output depends on the context, scale, research question or management needs. For this reason, simply pressing a button in a software application will usually not provide the expected answer because the end-user will always pose the question: what does this software output contribute to *my* specific question? We will always interpret the result and interpretation is subjective by definition.

Patterns of software design

Software design can be summarized in a conic spring model (Fig. 1) following a circular pattern of three components: (a) Abstraction (enhancing features), (b) Provision (enhancing dissemination) and (c) Feedback (enhancing acceptance). While the user wants the software to ‘just work’ this goal may be achieved only after many iterations of development cycles of

abstraction, provision and feedback. The idea of the circular pattern is exemplified by the evolution of my own software GuidosToolbox (GTB, Vogt and Riitters 2017) as follows.

Abstraction

Abstraction deals with the definition and implementation of the functional features and capabilities of the software. The purpose of abstraction is to increase the application potential through generalization.

- *Taking advantage of additional analytical frameworks:* The inclusion of a new analytical framework provides access to additional analysis tools. For example, the moving window methodology was used extensively to measure pattern on binary maps resulting in four classes: core, edge, perforated, patch (Riitters et al. 2000). By definition, this methodology is constrained to a fixed window-level perspective and cannot account for features outside the window, which may lead to classification errors at pixel level. In contrast, basic principles of mathematical morphology—the scientific foundation of pattern recognition in digital images—(Soille 2004) can be applied to investigate patches and pixels with respect to the roles they play relative to all other patches and pixels in the image. The first implementations of the approach in landscape ecology were to partition non-core pixels into different components (edge, islet, branch) in terms of their relationship to core (Vogt et al. 2007a), and to detect connecting structural and functional pathways between core patches (Vogt et al. 2007b, 2009) using the same analysis scheme. Therefore, mathematical morphology can be seen as a technological knowledge transfer providing a conceptualization of a patch as a collection of elements that are important to landscape ecological analysis.
- *Increasing flexibility through parameterization:* Most pattern analysis software starts by implementing a new measure with a tailored set of instructions. Setting up these instructions in a generic way allows their application in a variety of scenarios. For example, convolution filters are a generic form of the moving window technology, which can be used to assess proportion, density, or contagion, for edge detection, or for image editing

Fig. 1 Horizontal pattern recognition following the sequence Detect—Formulate—Discuss. The vertical pattern recognition is an iterative process of the same scheme generating knowledge through Abstraction. The software design column indicates the increase in software features starting from a simple script to a

full-fledged software suite, for example a GIS or any other professional software framework. The right column lists potential measures in landscape ecology contributing to the three components Abstraction, Provision and Feedback

such as smoothing, noise removal or contrast enhancement. Within GTB, a custom sequence of morphological operators was set up in a generic way via parameterization of processing parameters. The final version, Morphological Spatial Pattern Analysis (MSPA, Soille and Vogt 2009), allows fine-tuning the spatial pattern analysis with user-selectable edge-width, connectivity rule, class detail and different types of perforation. Because of the importance of scale in both pattern recognition and landscape ecology, there should always be at least one ‘scale parameter’ in any software implementation.

- **Thematic containers:** As the software becomes more complex it becomes useful to include image analysis modules for different purposes. These modules should be grouped into thematic containers to facilitate the user’s ability to find appropriate analysis tools for a selected thematic application field. In the same way that GIS software has thematic containers structured as submenus on a wide variety of tasks, Fragstats metrics are grouped into patch, cell, surface and structural metrics, and

the connectivity indices in Conefor (Saura and Torné 2009) are grouped into binary and probabilistic indices. The same process occurred in the case of my software, where additional image analysis modules for different purposes were developed and grouped into thematic application fields of landscape ecology, such as pattern, fragmentation, connectivity, distance, and cost/restoration. This included the addition of a concurrent multi-scale analysis of fragmentation and landscape mosaic (Wickham and Norton 1994; Riitters et al. 2000) to investigate their behavior across a series of scales.

- **Facilitate use and processing:** The application potential can be enlarged further by adding sections facilitating the use of the software. Examples include a detailed help section with program documentation and links to online resources, and batch-processing options to permit automated processing of many images without user interaction. As a result of early user-feedback to my software, I added a dedicated container of pre- and post-processing tools to facilitate preparing

input data for selected analysis schemes, modifying data or converting data into other formats including Google Earth image overlays.

Provision

Provision concerns all aspects of software setup, layout and dissemination. The purpose of provision is to optimize the usage of the software and to maximize the outreach into the user community.

- *OS/platform*: Which operating system and platform should be supported? Windows, Linux, Mac (PC), Android/IOS (tablet/phone)? This question is directly related to the cost/effort of development, choice of programming language, the potential reuse of already existing modules, and the development of software installer packages and software maintenance.
- *Installation/maintenance*: The development of software installers is a software project in itself. It is different for each operating system, maintenance cycle (installation-upgrading-uninstalling) and especially the installation type: systemwide versus standalone installation. Systemwide installation is most efficient because the system administrator deploys the software which can then be used by all system users, for example in corporate or government computing environments. In contrast, a standalone setup is designed to provide the entire software package in a single autonomous container, which does not interfere with the operating system. The standalone mode provides maximum portability because it allows use of the software from an external device (USB-key) or on systems where users do not have administrator rights.
- *Licensing*: A Closed Source license, usually outlined in an End User License Agreement, must be applied when using a proprietary programming language or the developer wishes to maintain full and exclusive control over future developments. On the other hand, Open Source Software (OSS) licenses provide the rights: use, copy, modify, and distribute the software; it uses copyright to enable a wide dissemination of software code and incentivize further developments. An OSS license scheme can contribute to a wider adoption and dissemination of a software. Depending on the

software setup and distribution purpose, care must be taken to choose the appropriate license which requires meticulous reading and understanding of the respective license terms.

- *Various end products*: There is a wide range of options for software provision and the software developer needs to understand which option is most suited for the targeted user group. A simple script (R, Python, etc.) can provide the easiest and quickest solution for a landscape ecologist who is familiar with the respective programming language. Further outreach is achieved to setup (in increasing order of workload) a web-application (without software installation), GIS-plugin, server-application for automated mass-processing, up to a self-contained desktop application driven by a user-friendly graphical user interface (GUI).
- *Propagation*: Besides various end products additional outreach can be achieved via pro-active dissemination including conducting workshops, disseminating software in public repositories, providing and distributing flyers, product sheets, offering online training courses or recorded YouTube videos, etc. Propagation is a very important part of software design because it forms the main source of feedback.

Feedback

User-feedback is fundamental for software developers. It serves as a measure of user satisfaction, product adoption and overall success. It is important for bug-fixing, quality control and as a source of new feature requests leading to an enriched user experience as well as further adoption and acknowledgement of the software package by the user community. In the circular layout of software development, Feedback will close the loop from Abstraction and Provision. The potential next round of software development can then start with implementing the new feedback into abstraction and provision measures.

- *Feedback form*: The developer can implement a template within the software product, including automatic collection of hardware and operating system details to facilitate bug-reporting and feature requests by the end-users.
- *Forum*: The developer can setup a dedicated forum website allowing end-users to discuss and report

any software-related issues between themselves as well as with the developer.

- *Social media*: The developer and/or the end-user can use applicable social media websites to provide an online portal for discussion and suggestions.
- *Direct contact*: The end-user may contact the developer directly and provide information on program bugs, installation and/or program feature requests.
- *User-group*: End-users can organize themselves in user-groups where they physically meet and share experiences and ideas to improve their understanding and use of a software package.
- *Citations*: Citations and application examples in journal articles can be an insight for software developers and a source of new ideas for further development.
- *Workshops*: Conducting workshops is the ideal occasion for a software developer to get into direct contact with end-users, learn about their needs, see which part of the software is of most interest or how it should be modified to be more applicable. For example, 60% of the analysis schemes and program functionality of GTB is based upon the feedback from GTB workshops.

People often equate software development to abstraction only and undervalue the importance of provision and especially feedback. While abstraction and provision are under the control of the developers, feedback is in the hand of the end-users. It can be seen as the entrance door for the end-user to pro-actively contribute to the overall scope and quality of the software. The value of provision is often underrated as well. For example, throughout the development process of the software GuidosToolbox over the past ten years, the development and implementation of the various image analysis tools (abstraction) amounts to only 20% of the total effort. The remaining 80% was dedicated to provision, meaning the majority of the workload was attributed to tasks considered to be critical for the communication, adoption and dissemination of the software to the user-community, and ultimately to trigger feedback.

The conic spring model (Fig. 1) provides a graphical summary of the development cycles in software design. In the initial phase, a single feature is selected for implementation in a simple script (Detect). Next,

this feature is programed in a computer language (Formulate). The dissemination of this script triggers user-feedback (Discuss). Feedback from the initial implementation and further suggestions can then lead to the next development cycle, where similar related features are detected and included forming a new, more generic rule set for implementation. Further cycles may follow in a similar fashion, enhancing the number of features, scope and application potential of the software. Moving upward in the conic spring leads to knowledge generation through abstraction. Adequate provision measures aim to trigger more feedback, which in turn leads to more generic analytical frameworks, meaning further abstraction. Pattern analysis can be assigned to horizontal levels where a given set of features is analyzed. Pattern recognition is associated with a step in the vertical dimension where an increase in abstraction entails the addition of similarly structured features or new conceptual approaches.

The conic spring model also applies to the highly interdisciplinary field of landscape ecology itself. Here, the inclusion of various concepts from other scientific fields (abstraction) has led to knowledge generation and advances in the understanding of ecological processes. For example, concepts of graph theory to assess connectivity were introduced by Keith and Urban, later implemented in the software package Conefor. McRae introduced concepts from electronic circuit theory to predict connectivity in heterogeneous landscapes resulting in the software Circuitscape (Shah and McRae 2008). In general, many new concepts from humanistic, natural and social sciences were added to and have enriched the understanding and application fields in landscape ecology.

Provision aims to effectively distribute knowledge and methods of landscape ecology to the actual user community but also to increase the outreach to a wider audience, i.e., policy and decision makers. This journal, IALE conferences, field research activities and projects dealing with risk assessment and landscape management all apply and disseminate principles of landscape ecology.

The discussions, issues found, and conclusions of these meetings and projects lead to feedback, resulting in new ideas, suggestions or approaches for feature enhancement, or problem solving. This feedback then leads to further insights and research, triggering a new cycle for the evolution of landscape ecology through

knowledge increase and a more holistic interdisciplinary vision.

Summary and outlook

This essay outlined a view on pattern recognition and pattern analysis within the context of software design. Software design follows three distinct rules aimed at enhancing functionality, disseminating existing and new analysis schemes, and trigger feedback to improve the user experience. Feedback is the key aspect to maximize the outreach, content and usability of software. It is clear that the conic model can be adapted to any other software project. I briefly outlined how the same pattern of knowledge generation model can be transferred (abstraction) into the field of landscape ecology, and it would be straightforward to apply it to another scientific field, e.g., mathematics or physics. Ultimately, one could argue that human evolution is driven by concepts of pattern recognition and pattern analysis because we use them with our cognitive system to interact with and understand the environment we live in. For example, reading a book or driving a car require pattern recognition. Artificial vision, robotics, fingerprint/face recognition are technical implementations of applied pattern analysis. Pattern recognition can be found in all fields, in acoustics [resonance/nodal patterns (Chladni 1787)], social-, psychological- and language patterns (Mirowsky and Ross 1986; Kuhl 2000; Pinker 2006), meteorological- and geographical patterns (Enfield and Mestas-Núñez 1999; Grotjahn et al. 2016), and even in a nested setup, such as patterns of patterns (fractal geometry) and different patterns at different scales (Wu et al. 2002; Wu 2004).

Returning to landscape ecology, and acknowledging that the capability for pattern recognition and pattern analysis is in each of us, the take home message of this discussion could be to listen to others and be open to new ideas. If you can look outside of your field and reflect about different perspectives, you may discover a new pattern which may be an undiscovered missing link that you were looking for.

Acknowledgements I am indebted to Jim Wickham, Kurt Riitters, Jennifer Costanza, Tim Wade, Bruce Jones, Jacek Kozak, Santiago Saura and Pierre Soille.

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

References

- Chladni EFF (1787) Entdeckungen über die Theorie des Klanges. Weidmanns Erben und Reich, Max Planck Institute for the History of Science, Library. <http://echo.mpiwg-berlin.mpg.de/MPIWG:EKGK1SP1>, <https://www.youtube.com/watch?v=wwJAgUBF4w>
- Enfield DB, Mestas-Núñez AM (1999) Multiscale variabilities in global sea surface temperatures and their relationships with tropospheric climate patterns. *J Clim* 12:2719–2733
- Grotjahn R, Black R, Leung R, Wehner MF, Barlow M, Bosilovich M, Gershunov A, Gutowski WJ, Gyakum JR, Katz RW, Lee YY (2016) North American extreme temperature events and related large scale meteorological patterns: a review of statistical methods, dynamics, modeling, and trends. *Clim Dyn* 46:1151
- Kuhl PK (2000) A new view of language acquisition. *PNAS* 97(22):11850–11857
- McGarigal K, Marks BJ (1995) FRAGSTATS: spatial pattern analysis program for quantifying landscape structure. Gen. Tech. Rep. PNW-GTR-351. U.S. Department of Agriculture, Forest Service, Pacific Northwest Research Station, Portland, OR. pp 122
- Mirowsky J, Ross CE (1986) Social patterns of distress. *Ann Rev Sociol* 12:23–45
- Pinker S (2006) How the mind works. *Ann N Y Acad Sci* 882:119–127
- Riitters KH, Wickham JD, O'Neill RV, Jones KB, Smith ER (2000) Global-scale patterns of forest fragmentation. *Ecol Soc (formerly Cons Ecol)* 4(2):3
- Saura S, Torné J (2009) Conefor Sensinode 2.2: a software package for quantifying the importance of habitat patches for landscape connectivity. *Environ Model Softw* 24:135–139
- Shah VB, McRae BH (2008) Circuitscape: a tool for landscape ecology. In: Varoquaux G, Vaught T, Millman J (eds) *Proceedings of the 7th Python in Science Conference (SciPy 2008)*, pp 62–66
- Soille P (2004) *Morphological image analysis: principles and applications*, 2nd edn. Springer, Berlin
- Soille P, Vogt P (2009) Morphological segmentation of binary patterns. *Pattern Recogn Lett* 30:456–459
- Vogt P, Riitters KH, Estreguil C, Kozak J, Wade TG, Wickham JD (2007a) Mapping spatial patterns with morphological image processing. *Landscape Ecol* 22:171–177
- Vogt P, Riitters KH, Iwanowski M, Estreguil C, Kozak J, Soille P (2007b) Mapping landscape corridors. *Ecol Ind* 7:481–488

- Vogt P, Ferrari JR, Lookingbill TR, Gardner RH, Riitters KH, Ostapowicz K (2009) Mapping functional connectivity. *Ecol Ind* 9:64–71
- Vogt P, Riitters KH (2017) GuidosToolbox: universal digital image object analysis. *Eur J Remote Sens* 50(1):352–361
- Wickham JD, Norton DJ (1994) Mapping and analyzing landscape patterns. *Landscape Ecol* 9:7–23
- Wu J, Hobbs R (2002) Key issues and research priorities in landscape ecology: an idiosyncratic synthesis. *Landscape Ecol* 17:355–365
- Wu J, Shen W, Sun W, Tueller PT (2002) Empirical patterns of the effects of changing scale on landscape metrics. *Landscape Ecol* 17:761
- Wu J (2004) Effects of changing scale on landscape pattern analysis: scaling relations. *Landscape Ecol* 19:125–138

Online resources

Circuitscape: <https://circuitscape.org/>

Conefor: <http://www.conefor.org/>
Fragstats: <https://www.umass.edu/landeco/research/fragstats/fragstats.html>
GDAL: <https://gdal.org/>
Google Earth: <https://www.google.com/earth/>
GuidosToolbox (GTB & GWB): <https://forest.jrc.ec.europa.eu/en/activities/lpa/gtb/>
MSPA: <https://forest.jrc.ec.europa.eu/en/activities/lpa/mspa/>
OSS licensing: <https://opensource.org/licenses>
https://en.wikipedia.org/wiki/Comparison_of_free_and_open-source_software_licenses
Python: <https://www.python.org/>
QGIS: <https://www.qgis.org/>
R-Project: <https://www.r-project.org/>

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.