Divide (along interfaces) and Conquer

Dirk Engel, info@engel-internet.de, August 22, 2023
https://github.com/dirkengel/

Introduction

"*A divide-and-conquer algorithm recursively breaks down a problem into two or more sub-problems of the same or related type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem.*" This is the definition we can find in [Wikipedia]. In my opinion, the terms *algorithm* and *recursive* in this definition reduce the true meaning of this paradigm due to a framing effect. Divide-and-conquer plays an important role in the entire field of SW engineering and is not so limited to the field of algorithmic techniques as the definition might suggest. In this article, I want to give some examples and examine the key success factor of its use.

Algorithm and Processor

When divide-and-conquer occurs along with the words *algorithm* and *recursive*, it is easy to get caught up in the context of programs running on the CPU. However, the human brain is also a pretty powerful processing unit and there are some basic but crucial SW engineering activities (algorithms) that work iteratively and also recursively.

Breakdown of Problem and Tasks

Having extended the scope of algorithms designed to run on computer HW to human-performed activities as well, we can take a look at the duties of a SW engineer who usually starts his/her career in kindergarten, followed by elementary school, high school, and university. Each of the passed modules is divided into semesters, lessons and subjects.
Divide-and-conquer right from the beginning. Finally, when working as a SW engineer, the following recurring divide-and-conquer tactics can be identified:

- **Requirements analysis**: The requirements for the desired functionality of a new product are divided into different groups such as functional requirements and quality attributes, starting from the high level down to atomic, independent and verifiable elements. The fulfillment of each individual requirement can be addressed and tracked.
- **System and Software design**: Find the right design to achieve the desired functionality and quality. To do this, we break down the system into subsystems, components, subcomponents and interfaces and try to bring order to the set of identified elements through structuring or grouping using layers and hierarchies. The design is often described using UML. A language with a formal grammar containing an alphabet and a set of rules that define how larger elements are assembled from the indivisible symbols. Decomposition or formal language, both are different manifestations of the divide-and-conquer principle.
- **SW development**: Choose the right programming language and translate your design into packages, classes, methods, functions, lambdas, and instructions. Reuse of experiences and knowledge from older projects, which are preserved and made accessible with the help of a pattern language and corresponding catalog. Depending on the programming language, search for the right idioms and choose

powerful algorithms. Eventually the algorithms again, but there is a lot more divide-and-conquer on the long road to getting here.

- **Planning, delivery, and maintenance**: Specify what features or fixes will be implemented when, what packages they will be included in, and how they will be bundled into an update. Maybe there are parts like writing test code that could be done with AI support or completely outsourced to an AI.
- **Communication**: All previous steps do not work without communication between the stakeholders. These stakeholders are part of companies, departments, teams and may be located in different countries, states, cities. To communicate successfully, you must know the right contact for different topics and how to convey information is an effective way.

Find the Right Interfaces

We review the previous examples again, this time focusing on the interfaces and the relationships between them:

- **Requirements analysis**: Requirements are usually divided into sections by topic and can be identified by a unique key. The relationship between the different requirements should follow rules such as consistency, non-redundancy, and completeness [InformIT], which form a kind of interface between the individual elements.
- **System and SW design**: External interfaces are explicitly defined as ports to communicate with the environment and to put the system in the right outer context. Likewise, internal abstract interfaces and relationships between the decomposed parts are specified.
- **SW development**: The abstract interfaces defined by the design are implemented. This can be done directly by realizing the abstract interface in a specific programming language, or indirectly by using an IDL as an intermediate step that serves as input to a code generator that can produce different implementations or bindings.

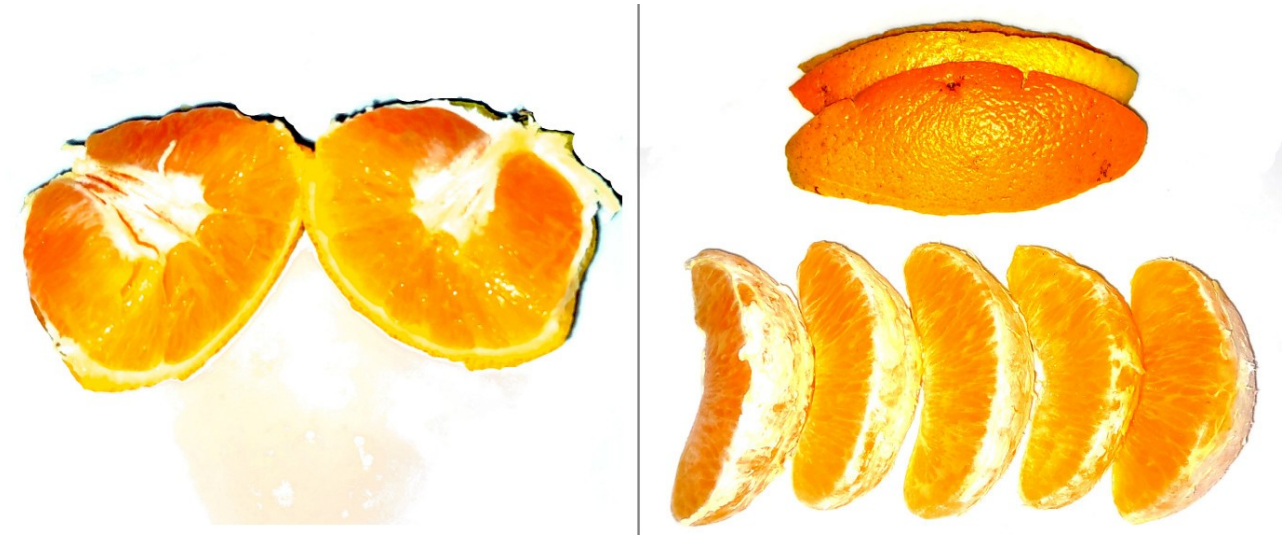

Fig. 1: Without considering the dividing lines, the division into parts leads to leaky results (left). If used successfully, however, the system can be broken down into subsystems, whereby the modularity achieved in this way can be used, for example, to replace individual faulty elements without affecting the rest of the system (right).

- **Planning, delivery, and maintenance**: Clarify the dependencies between different packages and specify which versions of other packages are required for a new or updated package and where to find them.
- **Communication**: With whom to talk about which topic and how. The how is not only about the right language/terminology, but also about the appropriate interpersonal protocol.

These activities may sound easier than they actually are. Breaking down something somehow is not hard, but making it successfully can be an art. The reason for this is that you cannot conquer a problem or task simply by breaking it down into random parts (Fig. 1). It is important to find the right dividing lines and suitable interfaces. Divide-and-conquer only really makes sense with the right interfaces, so putting the elements together is more (better) than just the sum of the parts. In my opinion, the search for the right dividing lines is more research than mere definition work, since the interfaces are usually already there, hidden somewhere in the problem space, but have to be discovered and mapped in the solution space.
It certainly makes sense to invest time and effort in interfaces since they form the backbone of your system. If you are not sure how important interfaces are, imagine having to choose between replacing half the components or half the interfaces in a given system. Which challenge would you rather accept? I argue that it is much easier to provide alternative implementations for the components than it is to find alternative definitions for the interfaces.
It is therefore important to understand the entirety of a system's interfaces as a central part of the system. The interfaces of an ecosystem thus form their own subsystem and are not just appendages to individual components. In this way, when applying interface segregation, you ensure consistency and completeness across a set of interfaces and are also able to achieve systematic usability in general. That concept is inspired by how divide-and-conquer is used to organize our human body with its inner organs responsible for separated functional concerns and the nervous or cardiovascular system for interconnection [LibreTexts].

Conclusion

Of course, it always helps to break down problems and tasks into parts until the new elements are manageable. But finding the right place for the cuts and suitable interfaces is not easy. And once the candidates are found, do not forget that not only the segments, i.e. the component implementations, but also and especially the interfaces and their relationships are part of the system that needs to be managed and continuously checked, maintained and possibly revised: Even good implementations go away, but interfaces stay.

References

Wikipedia	https://en.wikipedia.org/wiki/Divide-and-conquer_algorithm, last access 2023-07-31.
InformIT	https://www.informit.com/articles/article.aspx?p=1152528&seqNum=4, last access 2023-08-19.
LibreTexts	https://med.libretexts.org/Bookshelves/Anatomy_and_Physiology/Human_Anatomy_and_Physiology_Preparatory_Course_(Liachovitzky)/05%3A_Higher_Levels_of_Complexity-_Organs_and_Systems/5.01%3A_Organs_and_Systems_of_the_Human_Organism, last access 2023-08-21.