Modular Research Software

Prof. Dr. Wilhelm (Willi) Hasselbring
Software Engineering Group
Kiel University
https://se.informatik.uni-kiel.de

November 25, 2020
What is specific for Software Engineering in Science?
[Johanson and Hasselbring 2018]

• Functional Requirements are not known up front
  • No requirements specification, which could be handed over from scientists to software engineers
  • “Agile” explorative development

• No test oracles available

• Maintainability has low priority
  • Reproducibility valued higher than reusability
  • Implausible to modernize legacy scientific code

• Modularity to the rescue
  • Essential for maintainability, scalability, parallelizability, reusability, reliability, testability, and agility
The Quest for Modular Scientific Code I

From *Software Impacts*: [Calzavarini 2019]

Highlights of a software platform for fluid dynamics:

- “Ch4-project is a fluid dynamics code used in academia for the study of fundamental problems in fluid mechanics.
- It has contributed to the understanding of global scaling laws in non-ideal turbulent thermal convection.
- It has been used for the characterisation of statistical properties of bubbles and particles in developed turbulence.
- It is currently employed for a variety for research projects on inertial particle dynamics and convective melting.
- Its *modular* code structure allows for a low learning threshold and to easily implement new features.”
The Quest for Modular Scientific Code II
From *Software Impacts*: [Calzavarini 2019]

- “A dream for principal investigators in this field is to not have to deal with different (and soon mutually incompatible) code versions for each project and junior researcher in his/her own group.
- In this respect an object-oriented modular code structure would be the ideal one,
  - but this makes the code less prone to modifications by the less experienced users.
- The choice made here is to rely on a systematic use of **C language preprocessing directives** and on a hierarchical naming convention in order to configure the desired simulation setting in a module-like fashion at compiling time.”
Lazy Refactoring to the Rescue I

[Adorf et al. 2019]

Lazy Refactoring, guiding principles:

- Modularity, the core design principle naturally leads to the development of tools with clearly delimited scopes and well-defined Application Programming Interfaces (APIs).
- Public release of software source code.
  - The need to validate scientific results and the need to integrate disparate software tools make the ability to access source code valuable.
  - See also Hasselbring et al. [2020b].

Approach:

- Initial development should always lead to single-use code,
  - but this code is refactored into a reusable solution as soon as two further uses for it are found.
Lazy Refactoring to the Rescue II
[Adorf et al. 2019]

• This method requires that some quality controls be imposed on even initial prototypes to ensure that refactoring remains possible.

• The lazy refactoring approach is designed to balance sustainably improving the scientific software landscape with making immediate scientific progress.

• It does this by advocating for individual researchers to
  • evaluate existing software for its reuse potential prior to any code development,
  • adapt existing code bases for the problem at hand,
  • refactor existing code bases into proper packages whenever there are more than two use cases, and
  • develop rapidly evolving prototype code strictly focused on solving the problem at hand in all other cases.
1 Modules
Modular Design – Information Hiding

[Paras 1972a;b]

Components and modules:

- Old idea of markets for mass-produced software components [McIlroy 1968]
- Modules are the basis for programming-in-the-Large [DeRemer and Kron 1976]
- A module encapsulates a design decision. It can only be accessed by interfaces.
- Interfaces do not give more information than necessary for
  - the user of the module
  - the developer of the module

“Information Hiding, like dieting, is somewhat more easily described than done.”
(David Parnas, The Secret History of Information Hiding, Springer 2002)
• A software system consists of
  1 a collection of modules; and
  2 the relations between the modules.

Modular design defines the decomposition of a system into modules.

• Each module has an interface and a body:
  • **Interface** is the set of all elements in a module available to all users of the module, also called the module’s imported and exported resources.
  • **Body** is what realizes the functionality of a module, also called the implementation.

• A module **imports** resources from other modules.
• A module **exports** resources via its interface to other modules.
Principles of Modular Design

- The software architect designs the software architecture.
- The software architecture represents the decomposition of the system into modules and the inter-module relations.
- A first recommendation to design modular software: **Information Hiding** The interface must be separated from the body. Programs that rely on the module via its interface do not have to be rewritten as the body changes.

This principle implies that the interface of a module contains the right kind of information.

- Example: Users who need to manipulate polynomials should not be required to take into account the internal data structures used to represent the polynomials.
Use Relations should be hierarchical, non-cyclic!

- Usage Style: Avoid cyclic use relationships (dependencies)
- This way, we obtain architectural layers in the software.
- Typical example layers for information systems:
  1. User interface
  2. Business logic
  3. Data management
- If the structure is not hierarchical, we may end up with a system in which nothing works until everything works.
Expectations from Reuse:

- Reuse of well-proven components with high quality
- High flexibility for maintenance
- Faster construction with higher productivity
- Cost reduction

Another recommendation to design modular software:

**Design for Change** For instance, use of parameters for options that may later change.

- Example: For modules to manipulate polynomials, we foresee that different coefficient fields could be needed.
- However, reuse of components also introduces (indirect) dependencies among the reusing components.
  - Open sourcing reusable components to the rescue: Hasselbring and Steinacker [2017].
Low Coupling and High Cohesion I

• Low coupling means that modules are largely independent from each other.
  • Low coupling improves flexibility.

• Functions often used together belong to the same module so each module has a high internal cohesion.
  • High cohesion improves independence.

• The functionality of a module must be ready for testing and verification, independent of the rest of the program.

• Example: A module to manipulate polynomials should collect all the operations needed to manipulate polynomials.
Low Coupling and High Cohesion II

Understanding the balancing forces between coupling and cohesion is important when defining module / component / microservice boundaries:

- **Coupling** speaks to how changing one thing requires a change in another;
- **cohesion** talks to how we group related code.

These concepts are directly linked. Constantine’s law articulates this relationship well:

- “A structure is stable if cohesion is high, and coupling is low.” [Endres and Rombach 2003]
Coupling and Cohesion

Please rank the components A – D:
https://menti.com
Coupling and Cohesion

Please rank the components A – D:

https://menti.com
Modules in Python I

- A module is an ordinary Python file that ends with the .py extension. The file name prefix is the module name.
- We can define a group of classes, functions, variables, constants and so on inside a module.
- A client program, or simply client, is a program which uses classes, functions or variables defined in a module without knowing the implementation details.
  - Information hiding principle: Information Hiding in Python can also be achieved by defining private attributes within classes.
    - We can define a private object attribute by starting its name with two underscore characters (__).
  - The import statement searches the module file, parses and execute it’s content and makes it available to the client program.
Modules in Python II

• To reuse the classes or functions defined inside the module, we have to import the module into our program using the import statement. The syntax of import statement is as follows:

```python
import module_name
```

where module_name is the name of the file without the .py extension.

• Example: import math.

Then we can compute $\sqrt{2}$ via `math.sqrt(2)`. If we only need one element of a module:

```python
from math import sqrt
```

Then we can compute $\sqrt{2}$ simply as `sqrt(2)`.

• The `math` module is part of the standard library that is distributed with Python.
• An example of a user-defined module in `circle.py`:

```python
import math

class Circle:
    def __init__(self, radius):
        self.__radius = radius

    # getter method for radius attribute
    def get_radius(self):
        return self.__radius

    # setter method for radius attribute
    def set_radius(self, radius):
        if not isinstance(radius, int):
            print("Error: \"", radius, "\" is not an int")
        return self.__radius = radius

    # compute area
    def area(self):
        return math.pi * self.__radius ** 2
```
Another example of a user-defined module in rectangle.py:

```python
import math

class Rectangle:
    def __init__(self, length, width):
        self.__length = length
        self.__width = width

    def get_length(self):
        return self.__length

    def set_length(self, length):
        self.__length = length

    def get_width(self):
        return self.__width

    def set_width(self, width):
        self.__width = width

    def area(self):
        return self.__length * self.__width
```
Modules in Python V

- Example use of these modules:

```python
import circle
import rectangle

my_circle = circle.Circle(5)
print("Radius of circle:", my_circle.get_radius())
print("Area of circle:", format(my_circle.area(), ".2f"))

my_circle.set_radius(10)
print("Radius of circle:", my_circle.get_radius())
print("Area of circle:", format(my_circle.area(), ".2f"))

my_rectangle = rectangle.Rectangle(5, 10)
print("Length of rectangle:", my_rectangle.get_length())
print("Width of rectangle:", my_rectangle.get_width())
print("Area of rectangle:", format(my_rectangle.area(), ".2f"))
```
Modules in Python VI

Output:

<table>
<thead>
<tr>
<th>Radius of circle: 5</th>
<th>Area of circle: 78.54</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radius of circle: 10</td>
<td>Area of circle: 314.16</td>
</tr>
<tr>
<td>Length of rectangle: 5</td>
<td>Width of rectangle: 10</td>
</tr>
<tr>
<td>Area of rectangle: 50.00</td>
<td></td>
</tr>
</tbody>
</table>

- Module files are searched in a list of directories given by the variable `sys.path`.
- There is a variant to import all names that a module defines via `*`, but this is frowned upon, since it often causes poorly readable code.
Packages in Python I

• Modules are a simple way to share code between different scripts or notebooks in the same project.
• Module files must reside in the same directory as any script which imports them!
  • This is a big limitation; it means you can’t share modules between different projects.
• Once you have a piece of code that is general-purpose enough to share between projects, you should create a package.
• Packages are python’s way of encapsulating reusable code elements for sharing with others.
• Any directory that contains an __init__.py file is recognized by python as a package.
• Both, packages and modules define their own namespaces.
Packages in Python II

• Example folder and file structure for packages, sub-packages and modules:

```
sound/
   __init__.py          Top-level package
   effects/            Subpackage for sound effects
      __init__.py      Initialize the effects sub-package
      echo.py          module

Users of the package can import individual modules from the package, for example:

    import sound.effects.echo
```

• The Python Package Index pypi.org is a repository of software for the Python programming language.

• To make your package installable and accessible, you need some setup (not further discussed in this lecture).
Testing with Packages in Python

- You should add automated unit tests to your package,
  - for instance, via pytest,
- and configure automatic testing of your package via continuous integration,
  - for instance, by integrating with Travis-CI.
Modules with Jupyter Notebooks I

- Jupyter is a free, open-source, interactive web tool for literate programming [Knuth 1984], which you can use to combine
  - software code,
  - explanatory text,
  - computational output, and
  - multimedia resources in a single document.

- The result is a “computational narrative,” a document that allows you to supplement your code and data with analysis, hypotheses and conjecture.

- You can also use notebooks to create tutorials or interactive manuals for their software.

- Various cloud services such as BinderHub and CodeOcean emerge to publish notebooks as a service.
Modules with Jupyter Notebooks II

• However, Jupyter notebooks also encourage poor coding practice, by making it difficult to organize code into reusable modules and develop tests to ensure the code is working properly [Perkel 2018].

• Nevertheless, it is common that people want to import code from Jupyter Notebooks.

• This is made difficult by the fact that Notebooks are not plain Python files, and thus cannot be imported by the regular Python machinery.

• You could export extracts of notebooks as executable scripts whenever you edit your codes, but that should be too cumbersome and error-prone.

• Fortunately, we can import notebooks via import-ipylnb.
Modules with Jupyter Notebooks

- [https://github.com/fastai/nbdev](https://github.com/fastai/nbdev) provides two-way sync between notebooks and source code, which allow you to use your IDE for code navigation or quick edits if desired.

- Note that since every cell in an imported notebook is executed when you import it,
  - the imported notebook should only contain classes and function definitions.
  - Otherwise you’ll end up loading all variables and data into memory.

- Specify dependencies via `jupyter-nbrequierements`

- A large-scale study about quality and reproducibility of Jupyter Notebooks with recommendations for best practices presented by Pimentel et al. [2019].
Provided and required interfaces

UML: Unified Modeling Language [UML 2.5.1]

Connections between provided and required interfaces have to match, where applicable type substitutable (via polymorphism).
Provided and required interfaces

UML: Unified Modeling Language [UML 2.5.1]

Connections between provided and required interfaces have to match, where applicable type substitutable (via polymorphism).
Provided and required interfaces in Programming Languages

- Python emphasizes required interfaces
  - In particular via imports
- Java emphasizes provided interfaces
  - In particular via Java Interfaces
- With Python (also with C++), you may use abstract base classes to mimic interfaces.
- Java 9 (since 2017) and C++20 include modules with provided and required interfaces.
Module 1

  --

module body code

requires module2;

---

Module 2

exports com.timeserver;

module body code

---
C++20 Module Interface & Implementation Units

Module Interface Unit (mathInterfaceUnit.ixx)

```cpp
module;

import std.core;

export module math;

export namespace math {
    int add(int fir, int sec);
}
```

Module Implementation Unit (mathImplementationUnit.cpp)

```cpp
module math;

import std.core;

int add(int fir, int sec) {
    return fir + sec;
}
```

Thanks to the Module Interface Unit and the Module Implementation Unit, the module can be partitioned into its interface and its implementation.
2 Interfaces
Interface Design

- At a high level, an interface acts as a blueprint for designing classes.
- Like classes, interfaces define methods.
- Unlike classes, these methods are abstract.
  - An abstract method is one that the interface simply defines.
  - It doesn't implement the methods.
  - This is done by classes and methods/functions, which then implement the interface and give concrete meaning to the interface’s abstract methods.
Application Programming Interfaces I

- An application programming interface (API) is a computing interface which defines interactions between multiple software components,
  - in particular relevant for distributed components and services.
- It defines the kinds of calls or requests that can be made, how to make them, the data formats that should be used, the conventions to follow, etc.
- Through information hiding, APIs enable modular programming, which allows users to use the interface independently of the implementation.
- The separation of the API from its implementation can allow programs written in one language to use a library written in another.
Application Programming Interfaces II

• An API can specify the interface between an application and the operating system,
  • including the hardware that is managed by the operating system.

• Remote APIs allow developers to manipulate remote resources through protocols, regardless of language or platform.
  • For example, the Java remote method invocation API uses the Java Remote Method Protocol to allow invocation of functions that operate remotely, but appear local to the developer (proxy pattern).
  • CORBA extends this to heterogeneous programming languages. However, CORBA is a predecessor to Java RMI.
APIs for Web Development

- When used in the context of web development, an API is typically defined as a set of specifications,
  - such as Hypertext Transfer Protocol (HTTP) request messages,
  - along with a definition of the structure of response messages, usually in an Extensible Markup Language (XML) or JavaScript Object Notation (JSON) format.

- While “web API” historically has been virtually synonymous with web service, the recent trend has been moving away from Simple Object Access Protocol (SOAP) based web services and service-oriented architecture (SOA) towards more direct representational state transfer (REST) style web resources.
REST APIs

- REST APIs are designed around resources, which are any kind of object, data, or service that can be accessed by the client.
- A resource has an identifier, which is a URI that uniquely identifies that resource.
- REST APIs use a uniform interface, which helps to decouple the client and service implementations.
  - For REST APIs built on HTTP, the uniform interface includes using standard HTTP verbs to perform operations on resources.
  - The most common operations are GET, POST, PUT, PATCH, and DELETE.
- REST APIs use a stateless request model.
  - HTTP requests should be independent and may occur in any order, so keeping transient state information between requests is not feasible.
  - The only place where information is stored is in the resources themselves, and each request should be an atomic operation.
  - This constraint enables web services to be highly scalable.
APIs with Python

```python
response = requests.get('https://google.com/
print(response.status_code)
200
```

- Using `response`, you can examine the headers and contents of the response, get a dictionary with data from JSON in the response, and also determine how successful our access to the server was by the response code from it.
  - In our example, the response code was 200, which means that the request was successful.

- The requests module has a function called `get` that does an HTTP GET.

- The response object has a method called `json`. This takes the response body from the server and transforms it into a Python list of dictionaries.
  - Other popular data formats besides JSON are YAML and XML.
NASA Open API with Python
Astronomy picture of the day

```python
import requests
from pprint import PrettyPrinter  # Data printing

pp = PrettyPrinter()
api_key = 'YourAPIKey'  # Get your own

def fetchAPD():
    URL_APOD = "https://api.nasa.gov/planetary/apod"
date = '2020-11-11'
params = {
    'api_key': api_key,
    'date': date,
    'hd': 'True'
}
response = requests.get(URL_APOD, params=params)
pp.pprint(response.json())

fetchAPD()
```
Astronomy picture of 2020-11-11

```
{
    'copyright': 'Marcella Giulia Pace',
    'date': '2020-11-11',
    'explanation': 'cropped...',
    'media_type': 'image',
    'service_version': 'v1',
    'title': 'Colors of the Moon',
    'url': 'https://apod.nasa.gov/apod/image/2011/MoonColors_Pace_960.jpg'
}
```

Take a look: https://apod.nasa.gov/apod/image/2011/MoonColors_Pace_960.jpg
OceanTEA leverages modern web technology to enable the interactive exploration and analysis of high-dimensional oceanographic datasets.
Another Microservice Example: ExplorViz

[Hasselbring et al. 2020c],
https://www.explorviz.net/
API Design

- The design of an API has significant impact on its usage.
- The principle of information hiding describes the role of programming interfaces as enabling modular programming by hiding the implementation details of the modules so that users of modules need not understand the complexities inside the modules.
- The design of programming interfaces represents an important part of software architecture.
- There are two kinds of communication participants (or participants for short) that communicate via an API:
  - An API provider exposes any number of APIs;
  - an API client uses any number of APIs.
- One participant can also play both roles:
  - In an API Gateway in which the communication participant offers services as the provider of the gateway and is client to the services shielded by the gateway.
API Gateway

Mobile App

Single Page Web Application

API Gateway

Users

Images

Comments
API Gateway: Solution

• Implement an API gateway that is the single entry point for all clients.

• The API gateway handles requests in one of two ways.
  • Some requests are simply proxied/routed to the appropriate service.
  • It handles other requests by fanning out to multiple services.

• Rather than provide a one-size-fits-all style API, the API gateway can expose a different API for each client.
  • For example, the Netflix API gateway runs client-specific adapter code that provides each client with an API that’s best suited to its requirements.

• The API gateway might also implement security, e.g. verify that the client is authorized to perform the request.
Interface Representation and Interface Quality Patterns

[Zimmermann et al. 2017, Stocker et al. 2018]
API Key

- An API provider needs to identify the communication participant it receives a message from to decide if that message actually originates from a registered, valid customer or some unknown client.
- A unique, provider-allocated API Key per client to be included in each request allows the provider to identify and authenticate its clients.
- An API Key is a lightweight alternative to a full-fledged authentication protocol and balances basic security requirements with communication overhead.
  - Because the API Key is small, it can be included in each request without impacting performance much.
  - The API Key is a shared secret, and because it is transported with each request, it should only be used over a secure connection such as HTTPS.
- This pattern is concerned with the quality attribute security.
Rate Limit

- Having identified its clients, an authenticated client could use excessively many resources, thus negatively impacting the service for other clients.
- To limit such abuse, a Rate Limit can be employed to restrain certain clients.
  - Formulate this limit as a certain number of requests that are allowed per period of time.
  - If the client exceeds this limit, further requests can either be declined, be processed in a later period or be serviced by allocating a smaller amount of resources or by providing only best-effort guarantees.
- A Rate Limit gives the provider control over the client’s API consumption, but deciding on the right limits is not easy.
Rate Limit (cont.)

- The client can stick to its Rate Limit by avoiding unnecessary calls to the API.
  - Using a Wish List has a positive influence on a Rate Limit, as less data is transferred when the pattern is used.
  - The Pagination pattern also reduces response messages sizes, but in a different way (by splitting large repetitive responses into parts).
- Example:
  - The GitHub API v342 has a 5000 requests per hour per user limit for authenticated requests.
  - Clients can also make unauthenticated requests but these are limited to just 60 requests per hour.
  - In the GraphQL-based GitHub v4 API43, the Rate Limit has become more sophisticated and takes into account the number of queried nodes.
- Many API Gateways allow to introduce Rate Limits.
- This pattern is concerned with the quality attributes of reliability and performance.
If the service is paid for or follows a freemium model, the provider needs to come up with one or more pricing schemes.

The most common variations are a simple flat-rate subscription or a more elaborate consumption-based pricing scheme. Example:

<table>
<thead>
<tr>
<th>Storage (up to)</th>
<th>Pricing per Month</th>
</tr>
</thead>
<tbody>
<tr>
<td>First 5GB</td>
<td>Free</td>
</tr>
<tr>
<td>Next 95GB</td>
<td>$0.15 per GB</td>
</tr>
<tr>
<td>Over 100GB</td>
<td>$0.14 per GB</td>
</tr>
</tbody>
</table>

Rate Limits can be used to enforce different billing levels of the Rate Plan.

Define and monitor metrics for measuring API usage, such as API usage statistics on a per-operation level.

API Gateways can be used to implement metering and can thus also be used as enforcement points.
Interface Responsibility Patterns

[Zimmermann et al. 2020a]
Data-Oriented Interface Responsibility Patterns

[Zimmermann et al. 2020b]
Interface Evolution Patterns

[Lübke et al. 2019]
3 Summary
Summary

- Modularity is essential for maintainability, scalability and agility
  - but also for reusability
- Proper application programming interface design is key.
- Lazy refactoring may help [Adorf et al. 2019].
- Research software publishing practices in computer science and in computational science show significant differences [Hasselbring et al. 2020b]:
  - computational science emphasizes reproducibility, while
  - computer science emphasizes reuse.
- Reproducibility is essential for good scientific practice.
  - Follow the FAIR principles [Hasselbring et al. 2020a].


References II


References III

