Software Engineering

Introduction

Sasha Shapoval

University of Lodz

February 27, 2024

Grading

The grades of the course consist of the grades for the intermediate elements of control and the final exam. The weights are 0.6 and 0.4 respectively. Note that the marks for the group project are individual. The project topic and the contribution of each team member are agreed at the beginning. The formula is

$$Grade = \texttt{round}\,(0.4\times\text{“Exam”}+0.15\times\text{T}+0.1\times\text{P1}+0.15\times\text{P2}+0.20\times\text{P3})$$

where T and P stands for the test and presentations respectively. Rounding is to available grades: 2, 3, 3.5, 4, 4.5, 5.

Dates of the elements of control

- Test: 11.03–15.03 (according to the group schedule)
- Presentation 1: 08.04–12.04
- Presentation 2: 05.05–09.05
- Presentation 3: 26.05–30.05
- Exam: 09.06–13.06

Office Hours

- Wednesday, 16:15–17:45
- Office hours are for questions; NO elements of control are passed during office hours

Elements of control: Test

- Four tasks (university computers; internet off)
- Basic elements of algorithms: operations with integers, lists, loops
- Basic drawing with `turtle`.

Elements of control: Presentation 1

- Introductory presentation of the project with grading for
- Understanding of basic models of software engineering and team building by using your team and the project as examples (in particular you have to specify who does what)
- "Selling" the brief of your project (imitate that you offer a project for investors)
- Explanation of basic ideas, main challenges, and **links** between the different project parts (performed by different team members). In particular, you present the skeleton of your code (single slide)
- Ability to meet the time constrain of 12 minutes for the presentation

Elements of control: Presentation 2

- Presentation introducing a workable code with grading for
- Code itself (main challenge and the way to overcome)
- Explanation of the links between your part of the code and the other parts
- Ability to meet the time constrain of 12 minutes for the presentation
- Note: the speaker who cannot answer questions about the meaning of the lines of their code gets mark 0 for this presentation

Elements of control: Presentation 3

- Final presentation with grading for
- Explanation of the context (the aim of the project, known approach to the problem, system requirements)
- Workability of the project
- Ability to meet the time constrain of 12 minutes for the presentation

Possible Project Components

3 / 4 are mandatory for the team of 3 / 4 individuals

1. simple drawing (`turtle` , `tkinter`)
2. parsing of web-pages (`BeautifulSoup`)
3. processing of numerical data that includes the computation of relevant statistics (`pandas` , `numpy` , `scikit-learn`
4. more advanced previous item that includes the verification of a statistical hypothesis
5. visualization of numerical data (`matplotlib` , `seaborn` , `plotly`
6. backend with a (telegram) bot (`pytelegrambotapi`)
7. testing (`unittest` + test as you go)

Recommended projects

- Educational games (labyrinths, race to 100, Hanoi tower), where computation part is shared with you, and other things like leader board, audio and visual effect, and the backend can be added
- Parsing of the sites with numerical data followed by the data processing to graphical representation of the output and the control of requests through a messenger (you will have the codes that parse a site with the Covid data manipulating the extracted numbers, processing of the Bitcoin data, and drawing the graphs)
- Prediction of time series, where the input is given as a file (you may use the interpolation / extrapolation techniques and examples of visualization discussed in the class); backend with a messenger can complement the project
- Other projects that meet the requirements and agreed with the lecturer

General notes on projects

- I grade for the projects that show your knowledge of the course. All codes used in the course are shared with you
- You can get at least 4.5 out of 5 for a simple workable project based on the codes that I share with you
- You may decide to *improve* a basic project adding the elements not studied within our course
- These additional elements are not graded without the demonstration of knowledge related to the course

Elements of control: Exam

- 5 tasks, 1 question about theory of software engineering, 4 questions that require to write a code to answer
- Lab computers, Internet off, materials can be saved to the laboratory computer in advance
- No new questions

Students' Comments (2023)

- Professor has inbuilt scanner detecting only those student who does not know anything about the project
- Should behave equally with every student: *I spent too much time for bad projects making everybody clear that the work has not been performed properly; will change*
- Some actual coding would be nice
- exam + huge team project is too much *Please note, I do not require huge project; I am asking for a workable (mock) project*

Python

- For most platforms, you can download the required installation files from `https://www.python.org/downloads/` and install them using the appropriate platform-specific method
- You can use cloud resource Colab, which is an online Jupyter Notebook environment from Google: `https://colab.research.google.com/`. It is strongly recommended because the resource simplifies the presentation of your projects in the class
- Jupyter Notebook for your personal computer is recommended within Anaconda (`https://anaconda.org/`)

Course Parts

- Management: team building, the efficiency of the projects
- Software: (presumably efficient) coding
- Algorithms
- Elementary mathematics (mostly, probability and statistics) only to use library functions

Course materials

- Roger S. Pressman Software engineering: a practitioner's approach
- Lecture notes
- Allen B. Downey, Think Stats Probability and Statistics for Programmers `https://greenteapress.com/thinkstats/`
  codes:
  `https://greenteapress.com/thinkstats/thinkstats.code.zip`
  data: `https://greenteapress.com/thinkstats/nsfg.html`
- Jake VanderPlas, Python data science handbook,
  github with everything:
  `https://github.com/jakevdp/PythonDataScienceHandbook`
  handbook only:
  `https://jakevdp.github.io/PythonDataScienceHandbook/`

Introducing myself

- Professor Sasha (Alexander) Shapoval
- Researcher with background in mathematics and computer science
- Fields of interests: complex systems, data analysis, prediction of extremes, mathematical economics
- Papers in international refereed journals including *Scientific Reports*, *Physica D*, *Astrophysical Journal Communications in Nonlinear Science and Numerical Simulation*, *Journal of Mathematical Economics*
- Research grants
- Cooperation with industry as an expert in projects

Value one's time

- Send emails, do not chat
- Use office hours for questions
- Use messengers to cooperate, e. g., telegram
- In exceptional cases, online cooperation with me is possible

Value one's time

- Send emails, do not chat
- Use office hours for questions
- Use messengers to cooperate, e. g., telegram
- In exceptional cases, online cooperation with me is possible

Value one's time

- Send emails, do not chat
- Use office hours for questions
- Use messengers to cooperate, e. g., telegram
- In exceptional cases, online cooperation with me is possible

Value one's time

- Send emails, do not chat
- Use office hours for questions
- Use messengers to cooperate, e. g., telegram
- In exceptional cases, online cooperation with me is possible

Why do we study this course

- This is the (part of the) profession
- Open linkedin and look for data analyst, software developes, quantitative analysis and you see our topics as the requirements

Why do we study this course

- This is the (part of the) profession
- Open linkedin and look for data analyst, software developes, quantitative analysis and you see our topics as the requirements

Following the book focused on management

Roger S. Pressman, Software engineering: a practitioner's approach

Summary / Topics / Questions

- What is my name?
- What is the title of the course
- How are the final grades calculated? What are the elements of control?
- What are the dates of the presentations, test, and exam?
- What are the requirements to the project?
- What is the attitude to cheating?

Summary / Topics / Questions

- What is my name?
- What is the title of the course
- How are the final grades calculated? What are the elements of control?
- What are the dates of the presentations, test, and exam?
- What are the requirements to the project?
- What is the attitude to cheating?

Summary / Topics / Questions

- What is my name?
- What is the title of the course
- How are the final grades calculated? What are the elements of control?
- What are the dates of the presentations, test, and exam?
- What are the requirements to the project?
- What is the attitude to cheating?

Summary / Topics / Questions

- What is my name?
- What is the title of the course
- How are the final grades calculated? What are the elements of control?
- What are the dates of the presentations, test, and exam?
- What are the requirements to the project?
- What is the attitude to cheating?

Summary / Topics / Questions

- What is my name?
- What is the title of the course
- How are the final grades calculated? What are the elements of control?
- What are the dates of the presentations, test, and exam?
- What are the requirements to the project?
- What is the attitude to cheating?

Summary / Topics / Questions

- What is my name?
- What is the title of the course
- How are the final grades calculated? What are the elements of control?
- What are the dates of the presentations, test, and exam?
- What are the requirements to the project?
- What is the attitude to cheating?

Software engineering is

Fritz Bauer (1969)

the establishment and use of sound engineering principles in order to obtain economically sounding software that is reliable and works efficiently on real machines

- The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software
- The study of approaches as in the previous item

Software engineering is

Fritz Bauer (1969)

the establishment and use of sound engineering principles in order to obtain economically sounding software that is reliable and works efficiently on real machines

- The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software
- The study of approaches as in the previous item

Software engineering is

Fritz Bauer (1969)

the establishment and use of sound engineering principles in order to obtain economically sounding software that is reliable and works efficiently on real machines

- The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software
- The study of approaches as in the previous item

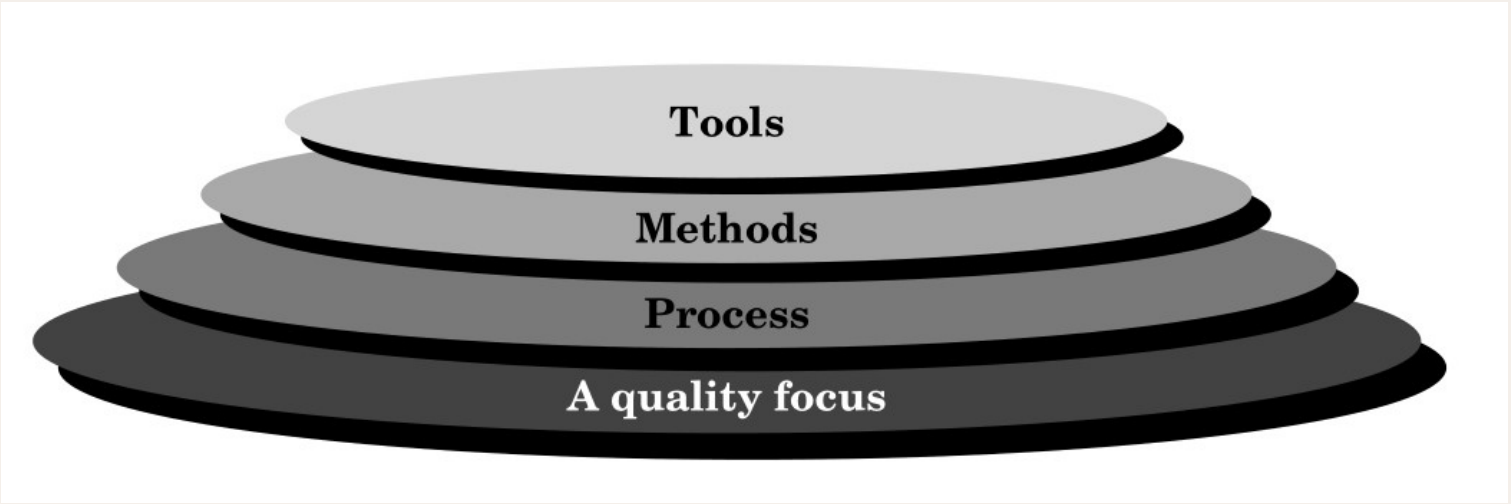
Software engineering is

Fritz Bauer (1969)

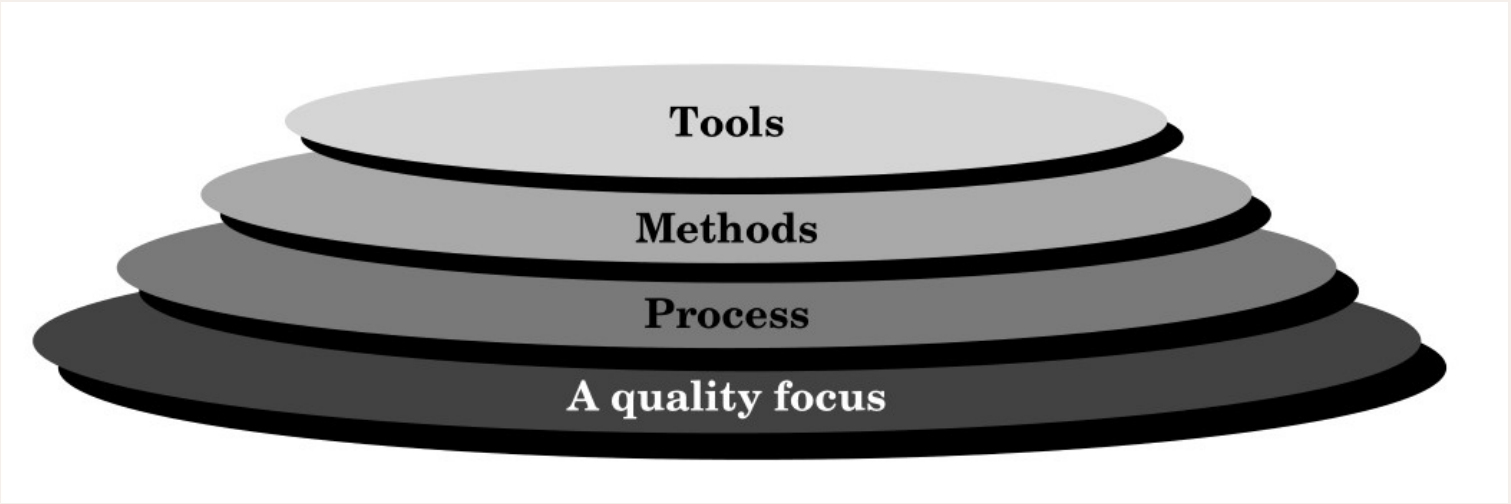
the establishment and use of sound engineering principles in order to obtain economically sounding software that is reliable and works efficiently on real machines

- The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software
- The study of approaches as in the previous item

Components of Software Engineering



Components of Software Engineering



Components of Software Engineering

- What are these components in the creation of USOS-system of the University of Lodz
- of Visual Studio Code?
- in the definition of a class in `C++`?

Parts of Software Engineering

- definition phase: what must be done
- development phase: how it can be done
- support phase: correction of errors

Parts of Software Engineering

- definition phase: what must be done
- development phase: how it can be done
- support phase: correction of errors

Parts of Software Engineering

- definition phase: what must be done
- development phase: how it can be done
- support phase: correction of errors

Definition phase

- During definition, the software engineer attempts to identify what information is to be processed, what function and performance are desired, what system behavior can be expected, what interfaces are to be established, what design constraints exist, and what validation criteria are required to define a successful system.
- The key requirements of the system and the software are identified. Although the methods applied during the definition phase will vary depending on the software engineering paradigm (or combination of paradigms) that is applied, three major tasks will occur in some form: system or information engineering, software project planning, and requirements analysis.

Developmet phase

during development a software engineer attempts to define how data are to be structured, how function is to be implemented within a software architecture, how procedural details are to be implemented, how interfaces are to be characterized, how the design will be translated into a programming language (or nonprocedural language), and how testing will be performed. The methods applied during the development phase will vary, but three specific technical tasks should always occur: software design, code generation, and software testing.

Developmet phase

during development a software engineer attempts to define how data are to be structured, how function is to be implemented within a software architecture, how procedural details are to be implemented, how interfaces are to be characterized, how the design will be translated into a programming language (or nonprocedural language), and how testing will be performed. The methods applied during the development phase will vary, but three specific technical tasks should always occur: software design, code generation, and software testing.

Can you describe different stages at the marshmallow challenge?
Can you associate testing with any stage of the marshmallow challenge?

The support phase

focuses on change associated with error correction, adaptations required as the software's environment evolves, and changes due to enhancements brought about by changing customer requirements. The support phase reapplies the steps of the definition and development phases but does so in the context of existing software

The support phase

focuses on change associated with error correction, adaptations required as the software's environment evolves, and changes due to enhancements brought about by changing customer requirements. The support phase reapplies the steps of the definition and development phases but does so in the context of existing software

you bought a computer with Windows and got the support that would last forever; for how long will this forever last?

The support phase

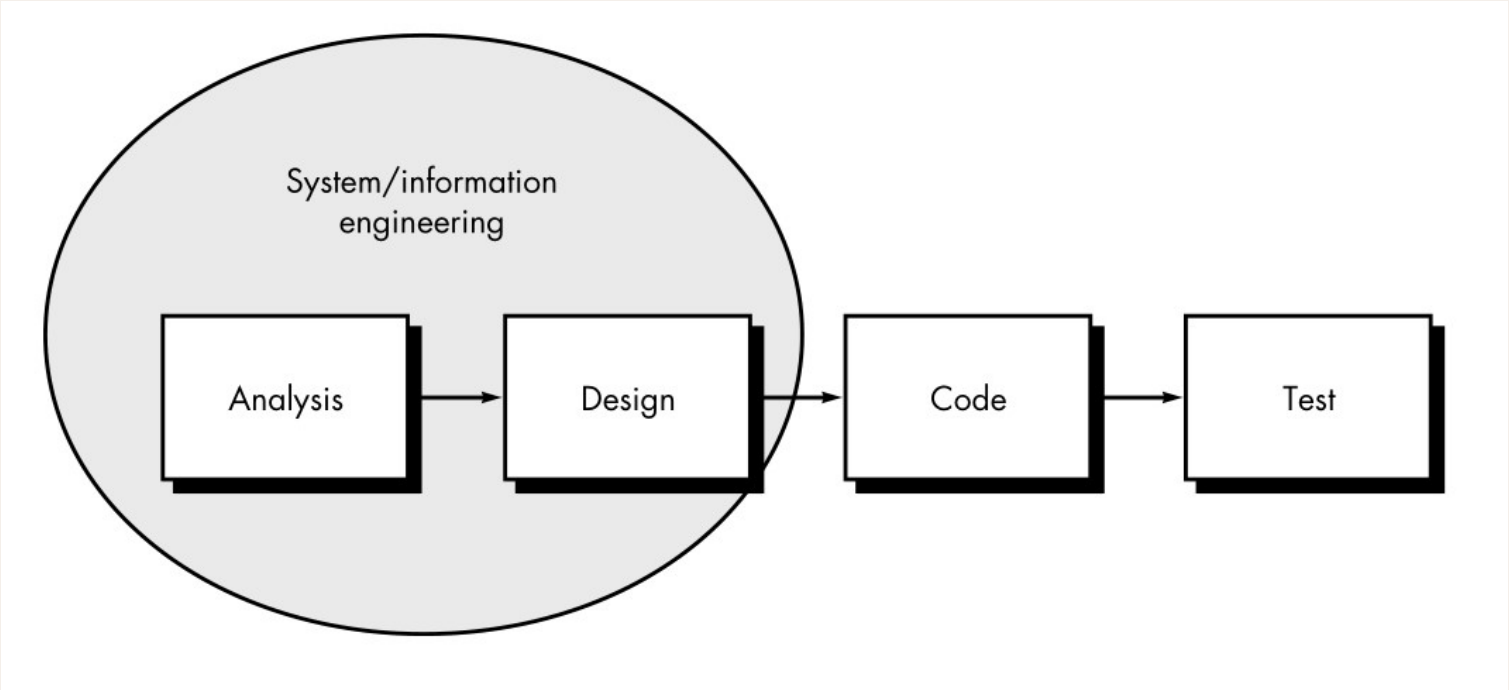
focuses on change associated with error correction, adaptations required as the software's environment evolves, and changes due to enhancements brought about by changing customer requirements. The support phase reapplies the steps of the definition and development phases but does so in the context of existing software

you bought a computer with Windows and got the support that would last forever; for how long will this forever last?

What do you expect from the developers of Windows?
To what extent your expectations are met?

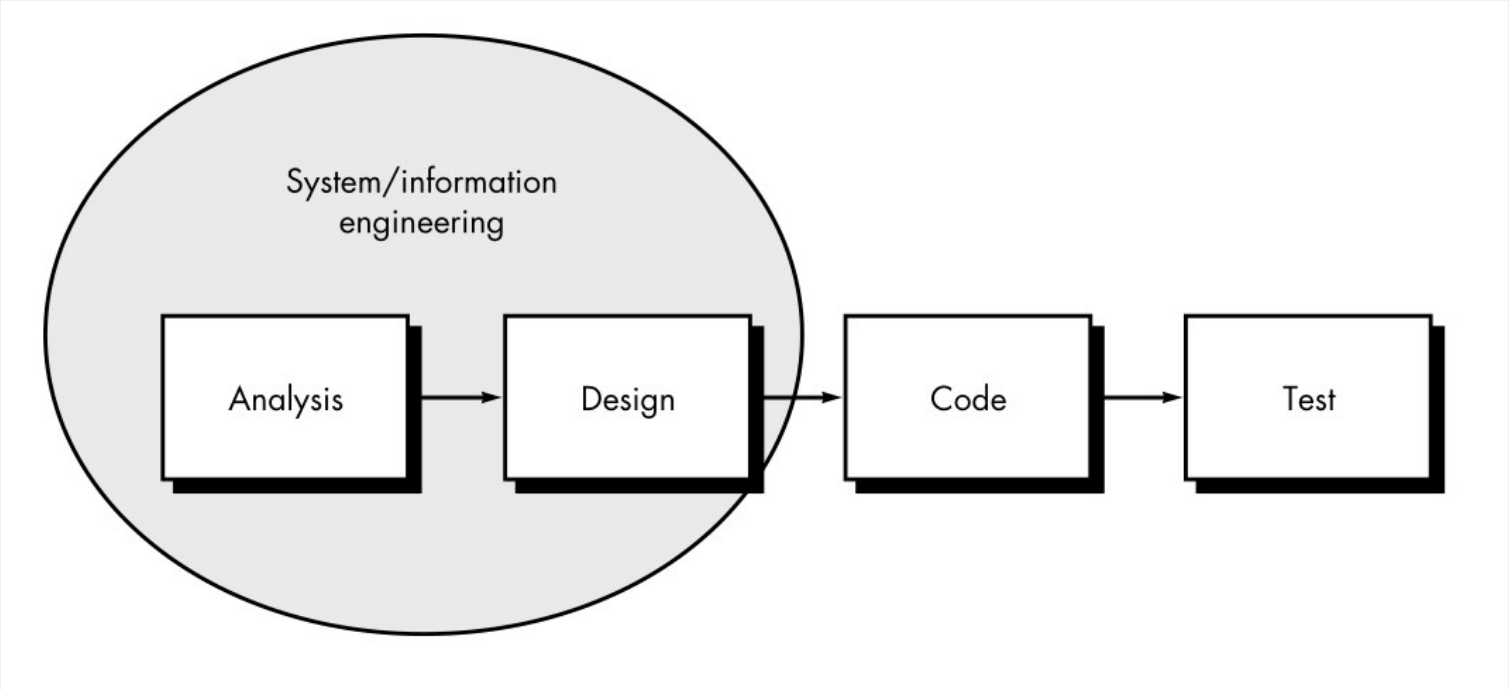
Classic lifecyle or waterfall model

Scheme



Classic lifecyle or waterfall model

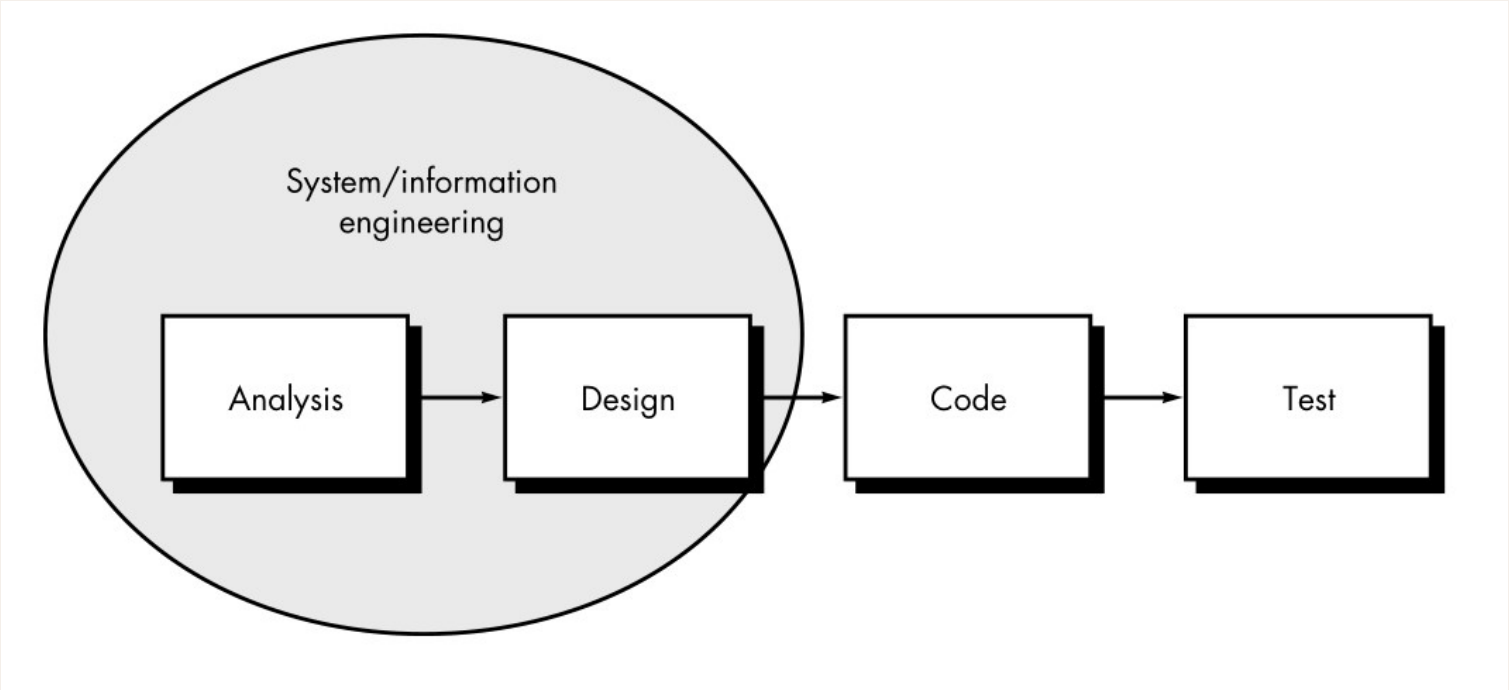
Scheme



Is this scheme applicable to Windows?

Classic lifecyle or waterfall model

Scheme



Is this scheme applicable to Windows?
Rather not as the processes go in parallel and are repeated many times
Is this scheme applicable to simple drawing project?
Rather, yes because of its simplicity

Software requirements analysis

The requirements gathering process is intensified and focused specifically on software. To understand the nature of the program(s) to be built, the software engineer must understand the information domain for the software, as well as required function, behavior, performance, and interface. Requirements for both the system and the software are documented and reviewed with the customer.

Design

Software design is actually a multi-step process that focuses on four distinct attributes of a program: data structure, software architecture, interface representations, and procedural (algorithmic) detail. The design process translates requirements into a representation of the software that can be assessed for quality before coding begins. Like requirements, the design is documented and becomes part of the software configuration

Code generation

The design must be translated into a machine-readable form. The code generation step performs this task. If design is performed in a detailed manner, code generation can be accomplished mechanistically.

Testing

Once code has been generated, program testing begins. The testing process focuses on the logical internals of the software, ensuring that all statements have been tested, and on the functional externals; that is, conducting tests to uncover errors and ensure that defined input will produce actual results that agree with required results.

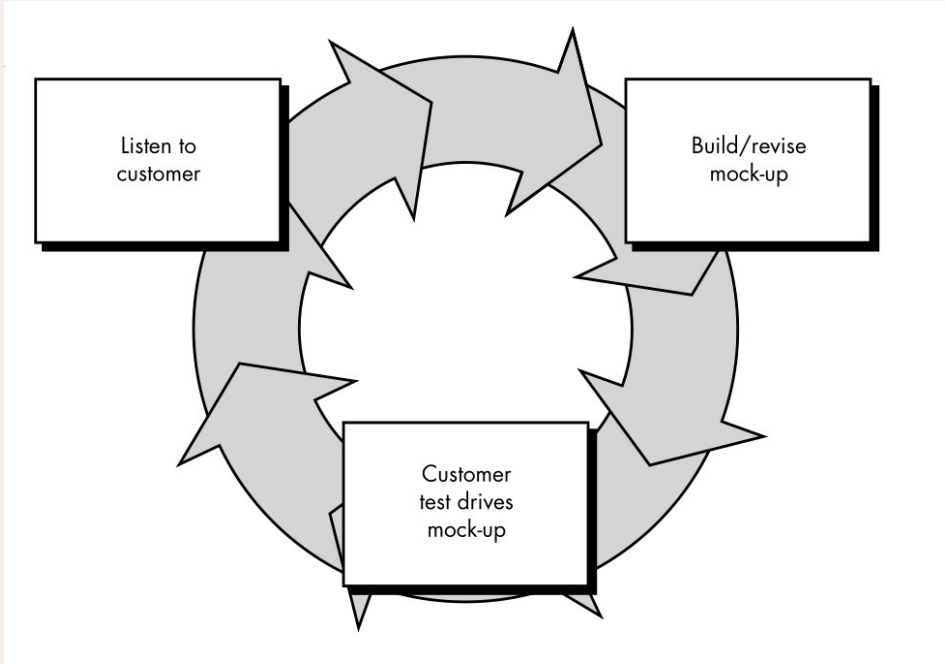
Support

Software will undoubtedly undergo change after it is delivered to the customer (a possible exception is embedded software). Change will occur because

- errors have been encountered,
- the software must be adapted to accommodate changes in its external environment (e.g., a change required because of a new operating system or peripheral device)
- the customer requires functional or performance enhancements

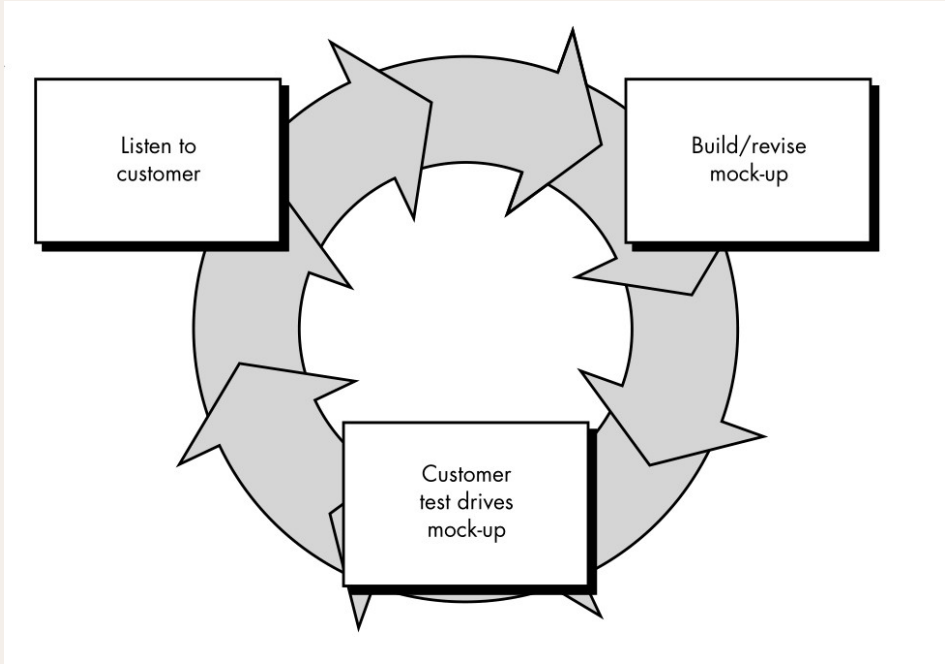
Software support/maintenance reapplies each of the preceding phases to an existing program rather than a new one.

Prototyping model: Scheme



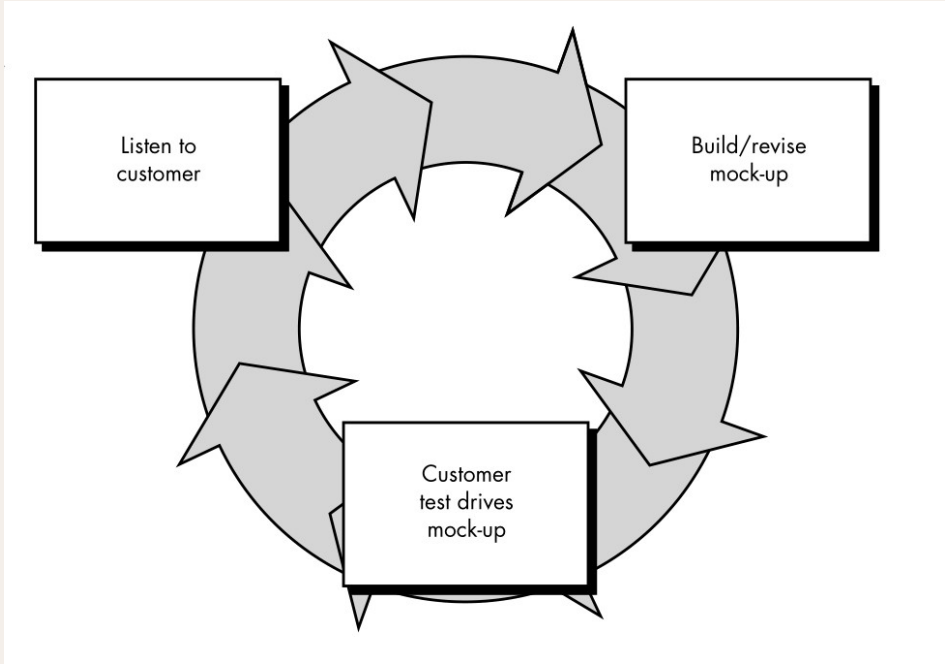
Is this scheme applicable to windows?

Prototyping model: Scheme



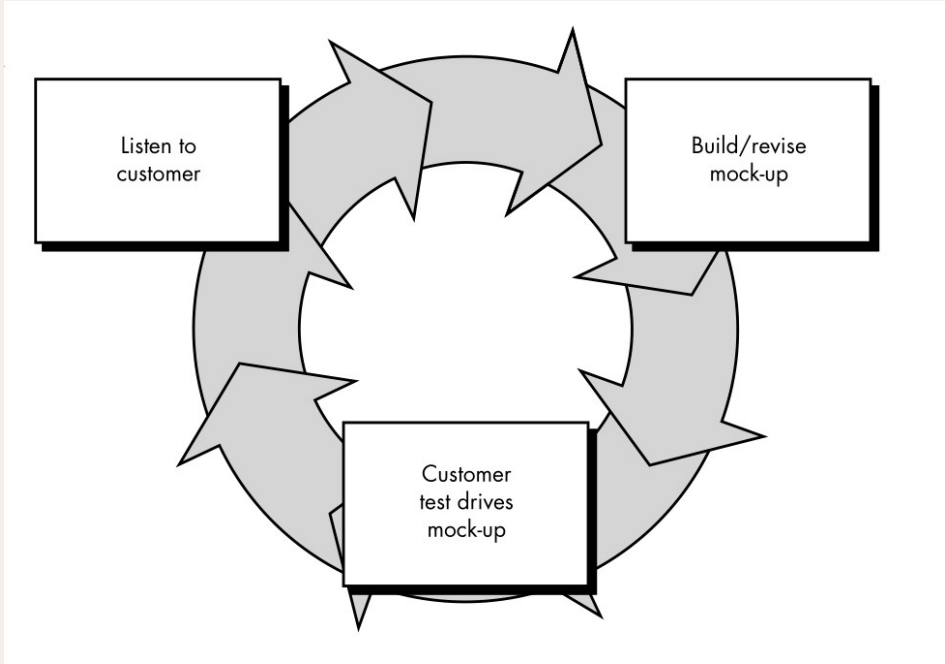
Is this scheme applicable to windows?
Probably, yes, but to what extent the developers follow the customers

Prototyping model: Scheme



Is this scheme applicable to windows?
Probably, yes, but to what extent the developers follow the customers
Is this scheme applicable to such open source projects as C or Python language themselves and their environments?

Prototyping model: Scheme



Is this scheme applicable to windows?
Probably, yes, but to what extent the developers follow the customers
Is this scheme applicable to such open source projects as C or Python language themselves and their environments?
Likely, yes, but who receives profits from the project and how?

Prototyping model

- requirements gathering: obtained from customers
- Then the prototype is evaluated by the customer/user and used to refine requirements for the software to be developed.

Other models

- Rapid application development
- Evolutionary software process models
  - spiral model
  - Concurrent Development Model
- Formal methods model
- Fourth generation techniques

Fourth generation techniques

The term fourth generation techniques (4GT) encompasses a broad array of software tools that have one thing in common: each enables the software engineer to specify some characteristic of software at a high level. The tool then automatically generates source code based on the developer's specification. There is little debate that the higher the level at which software can be specified to a machine, the faster a program can be built. The 4GT paradigm for software engineering focuses on the ability to specify software using specialized language forms or a graphic notation that describes the problem to be solved in terms that the customer can understand.

Fourth generation techniques

Currently, a software development environment that supports the 4GT paradigm includes some or all of the following tools:

- nonprocedural languages for database query
- report generation, data manipulation, screen interaction and definition, code generation
- high-level graphics capability;
- spreadsheet capability, and automated generation of HTML and similar languages used for Web-site creation using advanced software tools.

Initially, many of the tools noted previously were available only for very specific application domains, but today 4GT environments have been extended to address most software application categories.