CSc 131
Computer Software Engineering

Chapter 4
SWE Principles

Herbert G. Mayer, CSU CSC

Status 9/15/2019

Syllabus

- SW Engineering
- SWE Quality
- SW Development Process
- Sound SWE Guidelines
- Satisficing
- References
- Appendix

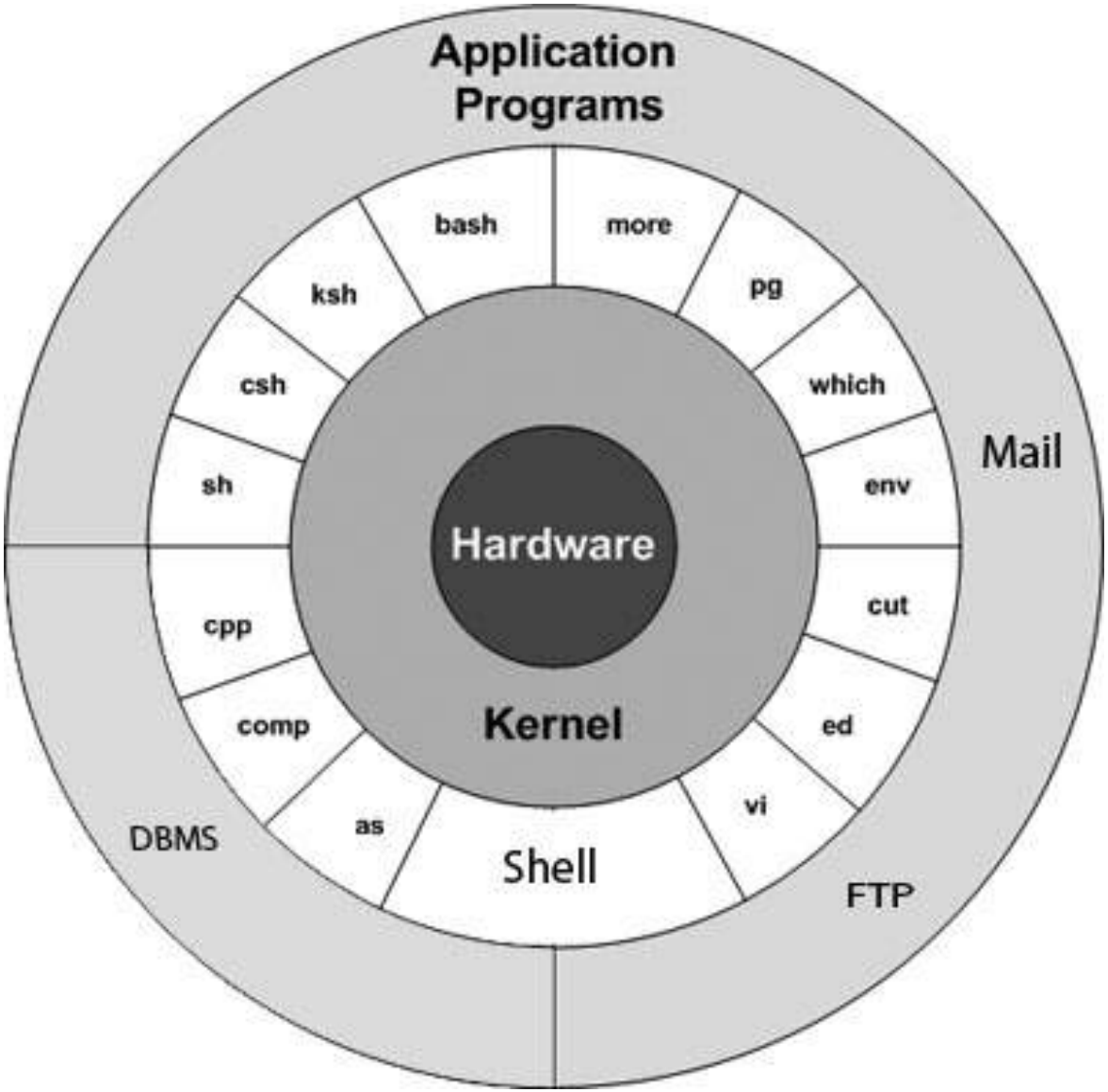
SW Engineering (SWE)

- **Goal of any SWE project is the successful creation of usable, large, complex SW product, such that:**
  - **Final product meets documented and agreed-upon goals**
  - **SWE goals include: performance, function, reliability, ease of use, cost, portability, robustness, legality (ownership, no infringement), backup, and future enhancement**
- **Large meaning >> 100 k LOC;** debatable what is large
  - **Linux kernel e.g. has > 15 * 10^6 lines of code ttl [1]**
- **Performance: execution speed poses no bottleneck**
- **Function: does all and only what is needed, not more**
- **Reliability: Same input + state causes same reaction, no lock-up ever, no crash ever**
- **Ease of Use: Behavior (input, output, function) is simple, w/o surprises; best if intuitively obvious!**

SWE Quality

- **Cost: within bounds of estimated space, $, time, i.e. time to develop, test, document, distribute, backup**
- **Portability: use widely available tools, language supported by more than 1 compiler, single language best**
- **Ideally build (Unix *make*) the product on more than developer platform, at least at each product release**
- **Robustness: no bugs; or small defined ratio of bugs; using bug tracking, managing, and repair strategy**
- **Legality: have or purchase rights for use for all tools; establish clear rights to own your created SW**
- **Backup: define time Δ between 2 complete backups: recovery needed, cost is known in case of total loss**
- **Future enhancement: able to extend function**

Linux Kernel & OS



Repeat: What is SW Engineering?

- "The systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software" –The Bureau of Labor Statistics – IEEE Systems and SW engineering
- "The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software" – IEEE Standard Glossary of Software Engineering Terminology
- "An engineering discipline that is concerned with all aspects of software production" – Ian Sommerville
- "The establishment and use of sound engineering principles in order to economically obtain software that is reliable and works efficiently on real machines" – Fritz Bauer

SW Engineering

Decision: Derive from existing vs. build from scratch

- Often new SW product is similar to already existing
- Then why create a new product to solve SW problem?
  - Why not use existing one (old one) as base and derive the new from it?
- Often "derive from existing source" is right decision
  - Provided clear ownership!
  - Else project may still be agreeable to paying license fee!
  - Source language of existing is strongly similar
  - Else source to source conversion, or manual mapping!
- Function of existing and new solution generally differ
  - So real SW development effort must be invested
- Key question: is projected cost of derivative less than the cost for new development?

SW Engineering

Decision: Derive from existing vs. build from scratch

- **If new SW product has no similar, existing base, develop new products; generally costly**
- **Note: Unlikely than in 2020s a SW company will write new HLL compiler!**
- **Advantage of composing new SW: Full control over choice of source language, PDE, legal ownership**
- **Often SW development engineers are more happy, more motivated, to write new SW from scratch**
- **SW developers generally must sign non-disclosure and code ownership agreement**
- **Satisfied programmers create stable SWE organization**

SW Development Process

- Claim: Development of large SW products is hard!
  - Yes, large fuzzily specified, perhaps >> 100 k LOC
- Counterintuitive!
  - Coding is so easy, and fun!
  - Algorithms (many) are provable
  - Where is the problem, the difficulty?
- Yet SW development is a human enterprise
- More issues than purely technical ones do surface
- Key issues: human problems, complexity of overall task, work constraints, geo distribution of developer teams, subjective PL preferences, instabilities of acquired tools (e.g. compute platform, operating system, dev. language maintenance, etc.), programmer preferences, manager / subordinate frictions, company stability, gender-based conflicts

SW Development Process

- Solutions to manage complexity of SW development generally are not unique to SW!
- Steps for solution:
- Build successful organization: managers + reports
- Possibly multiple levels of managed teams
- Manager instills SWE discipline! Not simple!
  - Hard for some programmers
  - Programmers at times exhibit Prima-Ballerina-syndrome ☺
  - SW development manager must recognize, handle maturely
- Build SW + infrastructure that is easy to use
- Devise a good SW development infrastructure
- *Sell* infrastructure to team: publish, train, document principles that have to be practiced

SW Development Process

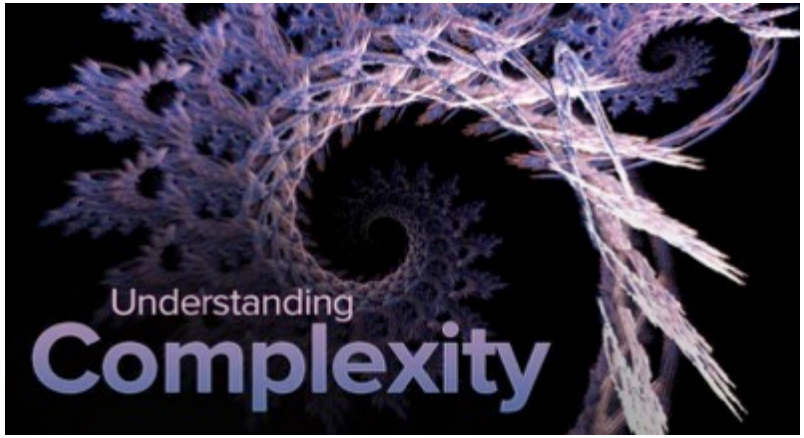
SW engineering principles helpful in large SW development process:

- Complexity control
- Separation of concerns
- Module composition and decomposition
- Problem analysis
- Record keeping
- Satisficing
- Reuse

SW Development Process

Complexity Control:

- Testability and maintainability are important for SWE
- They consume time in development life-cycle of the product; i.e. much time
- Sounds counter-intuitive to novice programmer
- Cyclomatic complexity is way to quantify complexity at source program (e.g. C++ class or method) level
- Developer: McCabe [2]



SW Development Process

Complexity Control, Cont'd:

- McCabe illustrates how using code size isn't a great way to contain, or even quantify complexity
- E.g. program as small as 50 lines consisting of 25 consecutive if-then constructs would have 33.5 million distinct control paths
- I.e. 33.5 million different paths in which program could execute
- Chances are only a small fraction would ever be tested in real executions
- Students: do read ref [2]
- Shows ways to reduce such path complexity!

SW Development Process

Separation of Concerns:

- **Problems, including time constraints, are too large to solve all at the same time**

- **Split work and separate solutions across**
  - **Business domains: management, marketing, engineering**
  - **Work areas: SW creation, document writing, technical support, employee (not only programmers) training**
  - **Different phases of project; see PLC**

SW Development Process

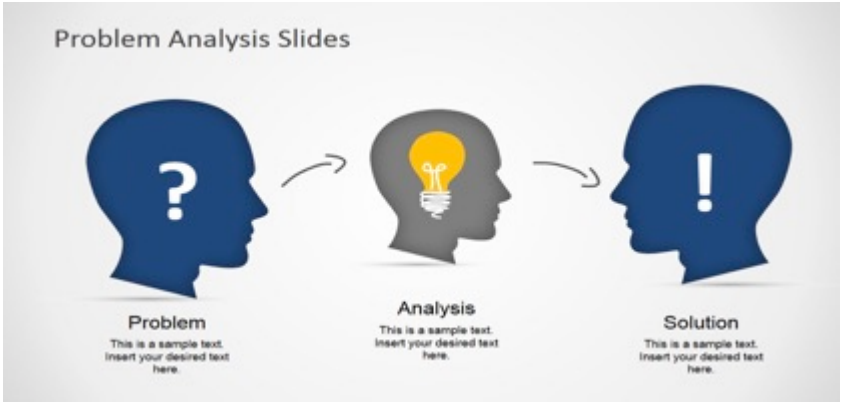
Module Composition and Decomposition:

- Chop (modularize) large problems into smaller ones that are jointly equivalent
- Solve those, with multiple SWE working in parallel
- Then build resulting, complex SW structures (problem solutions) out of smaller ones
- Requiring specific, defined interfaces, to function
- Some modules interface with more than one other module, increases need to carefully define interface
- Those interfaces should be as small as possible, but no smaller ☺
- AKA narrow interface

SW Development Process

Problem Analysis:

- Is analytic task: goal to increase SWE's understanding of an unbalanced, yet unsolved situation
- SW product to become that solution
- SWE must understand domain of data, problems, issues, resources, goals, business priorities, conflicting objectives
- Intent: new SW product will solve analyzed problem



SW Development Process

Record Keeping:

- Not just for current SW development project; keep future of SW organization + company in mind
- Create internal docs, electronic form, shared by team members with need-to-know
- May result in manuals + documents useful for similar projects in the future
- For SW modules, track revision history
- Have backup, including off-site
- Define backup interval (e.g. nightly) to contain maximum possible loss

SW Development Process

Record Keeping, Cont'd:

- Unix, Microsoft, etc. have several common tools to help keeping records, often by defined revision number
- Enables re-creation of a previous product (e.g. earlier release)

SW Development Process

Satisficing:

- Def: satisficing means the acceptance of an available SW option as satisfactory, though known to be not the complete or perfect solution
- Reason: e.g. insufficient time to create complete solution!
- But must solve current problem domain

SW Development Process

Reuse:

- Many SW problems have been solved before
- Either same or similar SW problem is already solved
- Thus past solutions provide strong starting base for new solution to current problem
- Use such earlier solutions: reuse
- Reuse saves time, $, grey cells
- Carries over quality confidence from past SWE product

Sound SWE Guidelines

- Any SW solution generated by SWE should behave as advertised; should do no more, no less
  - Documented in user manual (and internal docs)
- A SW solution should be easy to use
  - Sounds so straight forward, so open to interpretation ☹
- SWE process used to generate SW solution lends itself to modularization
- And to highly reliable execution
- SW solution comprises complete documentation what that solution is
- SW solution is reliable, stable, does not crash
- Maintenance cost over time is contained

Summary

- Software Engineer manages projects with the following goals in mid: functionality, portability, expandability, maintainability, and legality
- SWE uses documentation, modularization, record keepig, scheduling in parallel to succeed in complex SW development

References

1. Wikipedia: https://unix.stackexchange.com/questions/223746/why-is-the-linux-kernel-15-million-lines-of-code
2. Cyclomatic complexity: https://en.wikipedia.org/wiki/Cyclomatic_complexity

Appendix

Satisficing

- A portmanteau of satisfy and suffice, cooked up by Herbert A. Simon in 1956
- For a decision-making strategy that searches through available solutions until one is found that is good enough, i.e. until an acceptable threshold of "goodness" is reached
- Practiced by decision makers under pressure to find a solution when there isn't sufficient time to find the perfect solution
- Satisficing may find optimum for a simplified problem (simpler than the one to be solved); or find a satisfactory solution for actual problem