Introduction to Programming

Tim Pengajar

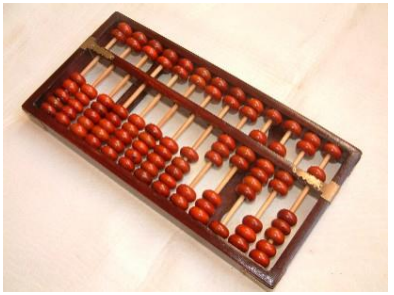
IF1210 Dasar Pemrograman

Tujuan

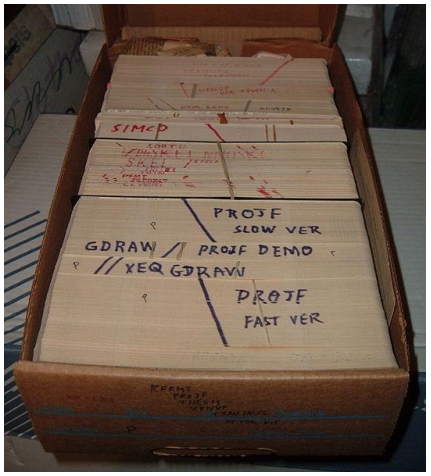
- Mahasiswa dapat memahami:
  - Sejarah pemrograman komputer
  - Paradigma pemrograman
  - Bahasa pemrograman
  - Berbagai hal terkait pemrograman (lingkungan, pemroses, dll.)
  - Pemrograman vs *software engineering*
  - Alur pengajaran pemrograman di STEI

Sejarah

- Konsepsi komputasi dalam bentuk aritmatika sederhana sudah ada sejak peradaban kuno
- Ada Lovelace (1830): algoritma pertama untuk menghitung deret Bernoulli, diproses pada Analytical Engine (Charles Babbage)
- Herman Hollerith (1880-an): menemukan cara merekam data dalam medium yang bisa dibaca mesin menggunakan punched cards (Hollerith cards)



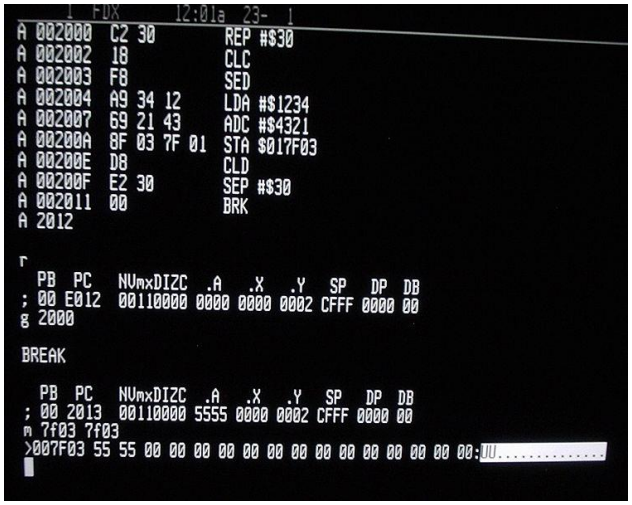
Abacus
(Sumeria, c 2500M)



Punched card

Sejarah

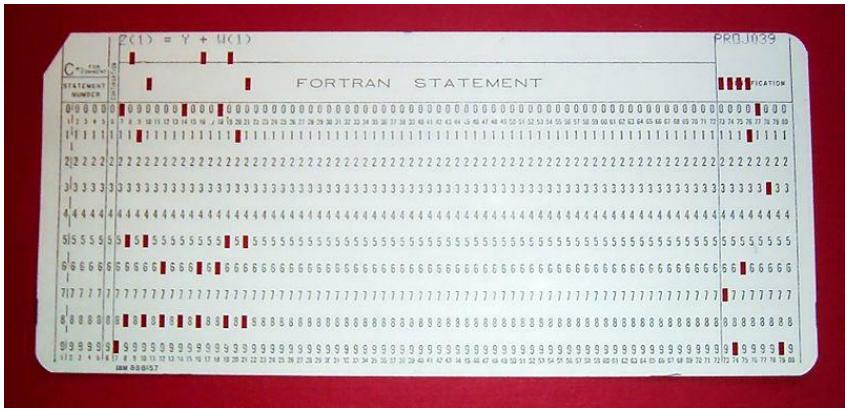
- Penemuan arsitektur komputer Von Neumann (1945):
  - Memungkinkan program disimpan di memori komputer
  - Menggunakan notasi biner
  - Bahasa pemrograman pertama: **bahasa mesin** (*machine language*) → instruksi operasi elementer yang langsung dipahami oleh mesin komputer
  - Selanjutnya berkembang **bahasa assembly** → instruksi bahasa mesin ditulis dalam bentuk simbol-simbol



bahasa mesin

Sejarah

- Bahasa pemrograman tingkat tinggi pertama:
  - Bahasa pemrograman tingkat tinggi pertama: **FORTRAN** (1954), dikembangkan pertama kali oleh IBM
  - Bahasa pemrograman komersial pertama: **COBOL** (1959), didesain pertama kali oleh Grace Hopper
  - Kode-kode program pertama ditulis di atas punched cards
- Seiring dengan perkembangan komputer, ribuan bahasa pemrograman berkembang



Kode FORTRAN awal dalam sebuah punched card

Programmer menggunakan keypunch untuk "memrogram" di atas punched card

Pemrograman

- Pemrograman [komputer] adalah proses untuk memformulasi persoalan komputasi menjadi program [komputer]
- Pemrograman tidak hanya ***coding***
- Pemrograman adalah analisis persoalan (membuat spesifikasi), implementasi (*coding*), *testing*, *debugging*
- *Programming : art / craft / engineering*?

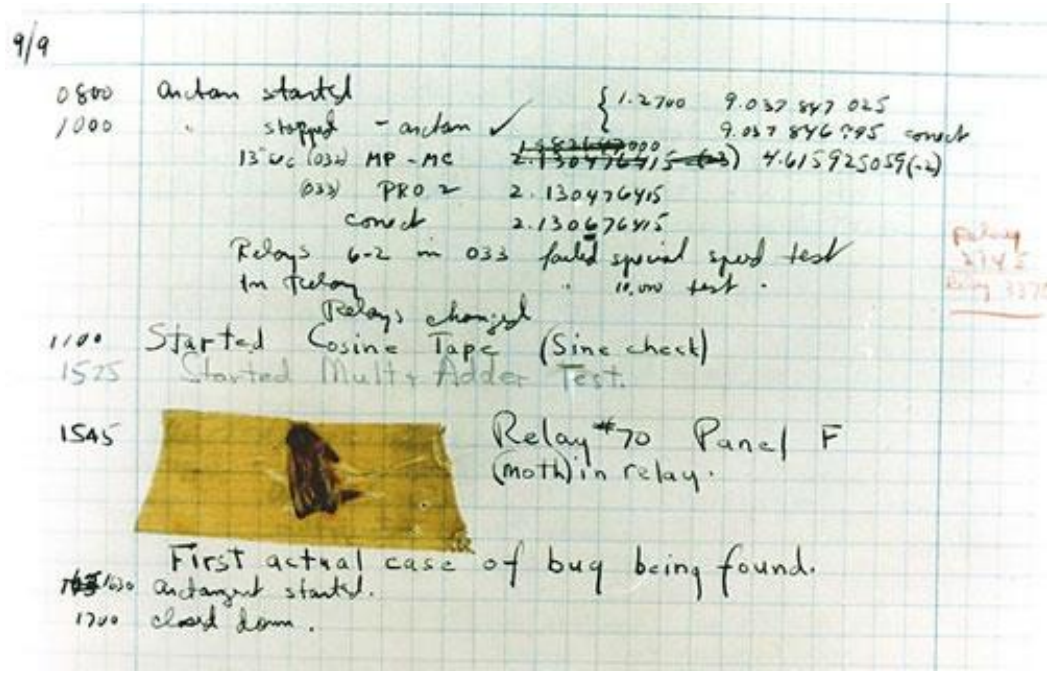
Kegiatan Pemrograman

- Analisis persoalan, membuat spesifikasi, menyusun **algoritma**
- *Program writing* (*coding*) → implementasi pada bahasa pemrograman tertentu
- *Program execution (observation, debugging, testing)*
- *Program reading*
- *Program correctness* and *complexity analysis*
- *Program maintenance*

Video : Coder vs Programmer (1:00)

Intermezzo

- "**Bug**" adalah istilah yang umum digunakan untuk kesalahan pada program
- *Bug* pertama pada komputer sebenarnya adalah *bug* (serangga sungguhan) yang ditemukan pada komputer Harvard Mark II, dicatat pada tgl. 9 Sept 1947 oleh Grace Hopper



Bug komputer pertama

Belajar Pemrograman vs Belajar Bahasa Pemrograman

Ada **RIBUAN** bahasa pemrograman di dunia saat ini!!

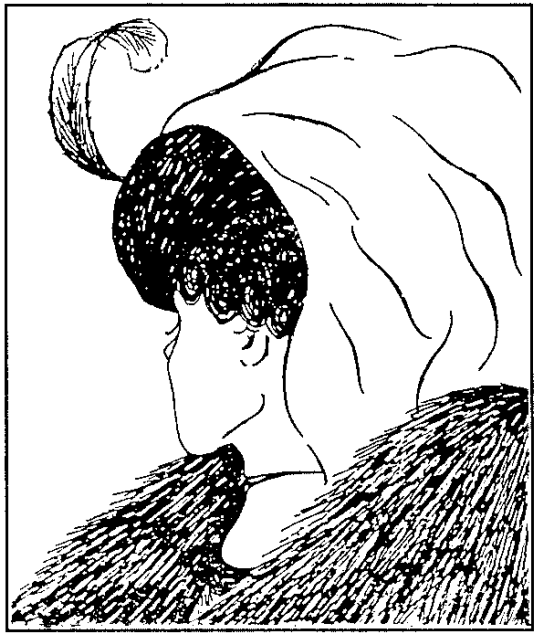
- Tidak mungkin semua bahasa pemrograman dipelajari di kuliah
- Oleh karena itu yang diajarkan adalah "belajar pemrograman" → melalui **pola pikir komputasional** dan **paradigma pemrograman**

Belajar memrogram ≠ Belajar bahasa pemrograman

- **Analogi**: Belajar menulis karya ilmiah ≠ belajar tata bahasa Indonesia

Paradigma Pemrograman

- **Paradigma** [pemrograman] adalah sudut pandang penyelesaian persoalan dengan [program]
- Setiap persoalan menggiring kita pada **pendekatan khusus** untuk pemecahannya → paradigma memberikan strategi analisis khusus pemecahan masalah
- Jenis persoalan tertentu dapat dipecahkan dengan baik dengan paradigma tertentu



What do you see?
By shifting perspective you might see an old woman or a young woman.

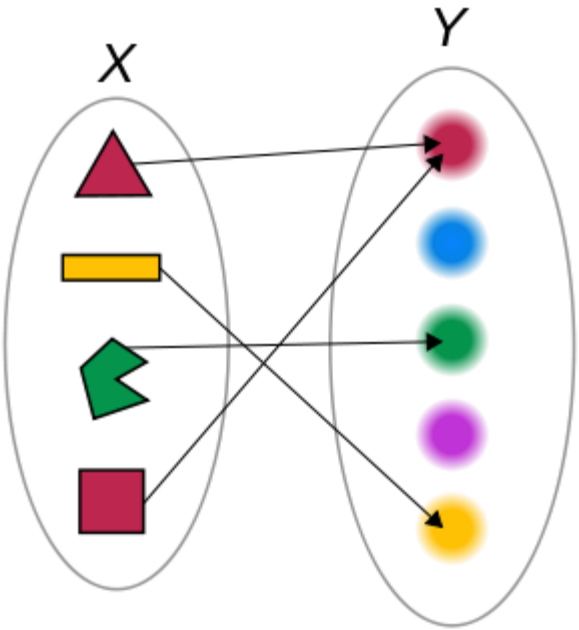
Paradigma Pemrograman

- Jenis paradigma pemrograman yang dikenal:
  - **Fungsional**
  - **Imperatif / prosedural**
  - Berorientasi objek (*object oriented*)
  - Deklaratif
  - Relasional
  - Event-driven
  - Konkuren
  - …
- Beberapa akan dijelaskan singkat pada slide berikutnya

Dipelajari di kuliah IF1210

Paradigma Fungsional

- Didasari oleh konsep pemetaan dan **fungsi** di matematika
- Pemrogram mengasumsikan bahwa ada fungsi-fungsi yang dapat dilakukan → penyelesaian masalah didasari atas **aplikasi dari fungsi-fungsi**
- Kelakuan program adalah suatu rantai **transformasi** dari sebuah **keadaan awal** menuju ke suatu **keadaan akhir**, yang mungkin melalui keadaan antara



- Lebih lanjut mengenai paradigma fungsional → nantikan minggu depan!

Paradigma Imperatif/Prosedural

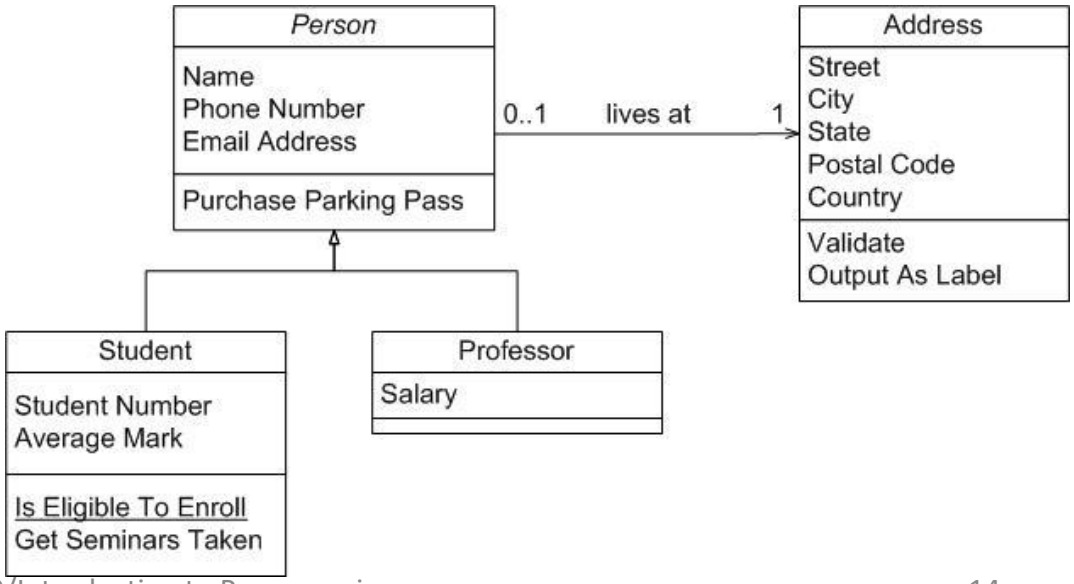
- Didasari oleh konsep mesin Von Neumann (*stored program concept*):
  - sekelompok tempat penyimpanan (**memori**), yang dibedakan menjadi memori **instruksi** dan memori **data**; masing-masing dapat diberi nama dan harga
  - Instruksi akan dieksekusi satu per satu secara sekuensial oleh sebuah pemroses tunggal
- Program didasari oleh **strukturasi informasi** di dalam memori dan **manipulasi** dari informasi yang disimpan tersebut

Program = Algoritma + Struktur Data

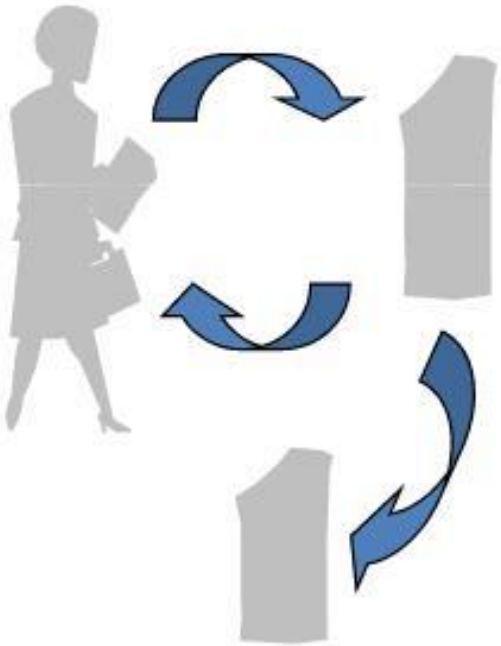
- Lebih lanjut → nantikan beberapa minggu ke depan!

Paradigma Berorientasi Objek

- Didasari oleh konsep **objek**
  - Sebuah **objek** mempunyai atribut (kumpulan sifat) dan mempunyai kelakuan (kumpulan reaksi, metoda)
  - **Kelas** adalah objek mempunyai atribut yang sama dan diturunkan ke semua objek yang berada dalam kelas yang sama



- Procedural



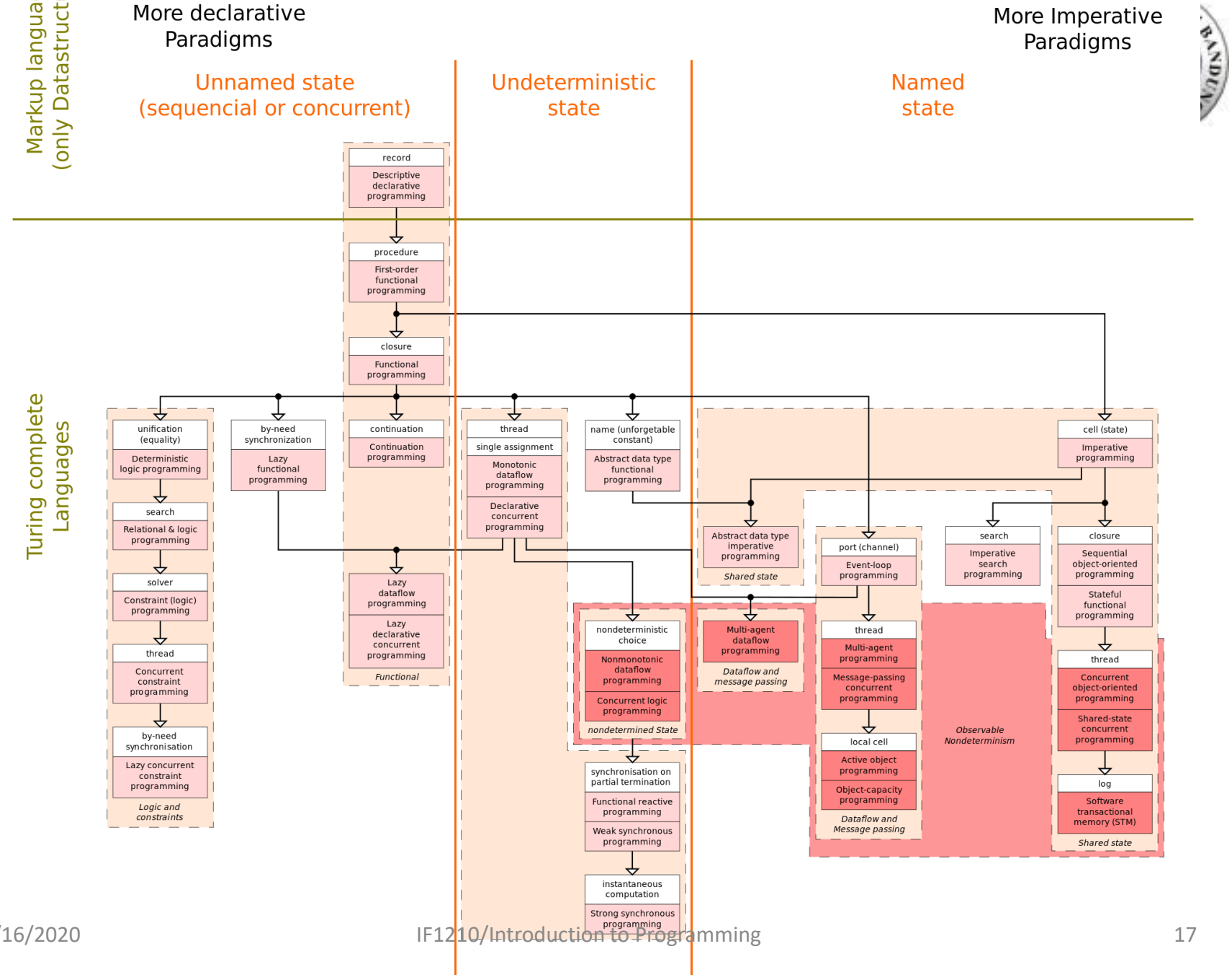
Withdraw, deposit, transfer

- Object Oriented

Customer, money, account

Beberapa paradigma lain…

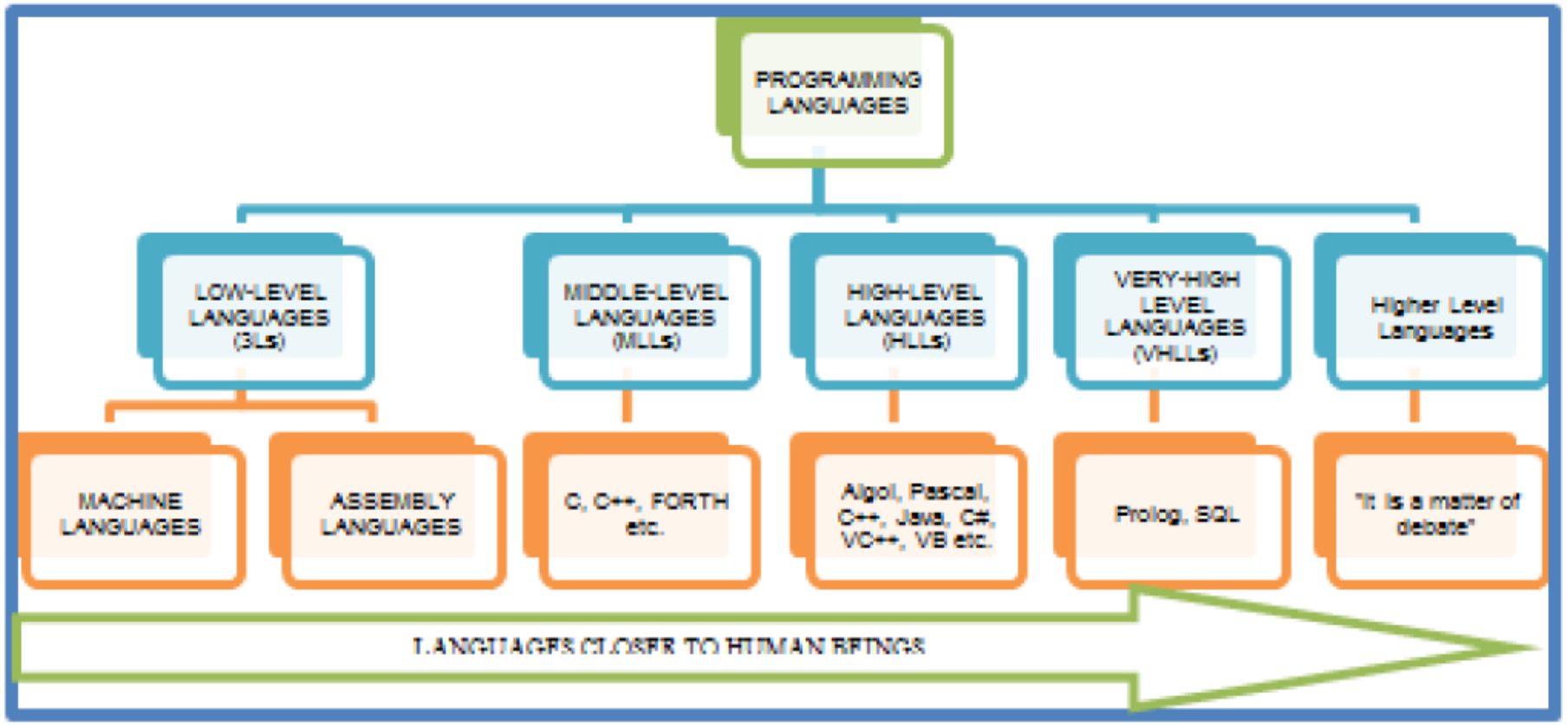
- Paradigma **deklaratif**:
  - Program adalah kumpulan aksioma (fakta dan aturan deduksi)
  - Program memakai aturan deduksi dan mencocokkan pertanyaan dengan fakta-fakta yang ada untuk menjawab pertanyaan
- Paradigma **relasional**: didasari dari konsep relasi pada matematika
- Paradigma **konkuren**: erat hubungannya dengan arsitektur sistem untuk pemrosesan paralel
- Paradigma **event-driven**: didasari pada konsep event, misalnya aksi user, sensor, messages, dll.
- …


Markup languages (only Datastructures)
More declarative Paradigms
More Imperative Paradigms
Unnamed state (sequencial or concurrent)
Undeterministic state
Named state
Turing complete Languages
record
Descriptive declarative programming
procedure
First-order functional programming
closure
Functional programming
unification (equality)
Deterministic logic programming
search
Relational & logic programming
solver
Constraint (logic) programming
thread
Concurrent constraint programming
by-need synchronisation
Lazy concurrent constraint programming
Logic and constraints
by-need synchronization
Lazy functional programming
continuation
Continuation programming
Lazy dataflow programming
Lazy declarative concurrent programming
Functional
thread
single assignment
Monotonic dataflow programming
Declarative concurrent programming
name (unforgetable constant)
Abstract data type functional programming
nondeterministic choice
Nonmonotonic dataflow programming
Concurrent logic programming
nondetermined State
synchronisation on partial termination
Functional reactive programming
Weak synchronous programming
instantaneous computation
Strong synchronous programming
Abstract data type imperative programming
Shared state
Multi-agent dataflow programming
Dataflow and message passing
port (channel)
Event-loop programming
thread
Multi-agent programming
Message-passing concurrent programming
local cell
Active object programming
Object-capacity programming
Dataflow and Message passing
search
Imperative search programming
Observable Nondeterminism
cell (state)
Imperative programming
closure
Sequential object-oriented programming
Stateful functional programming
thread
Concurrent object-oriented programming
Shared-state concurrent programming
log
Software transactional memory (STM)
Shared state

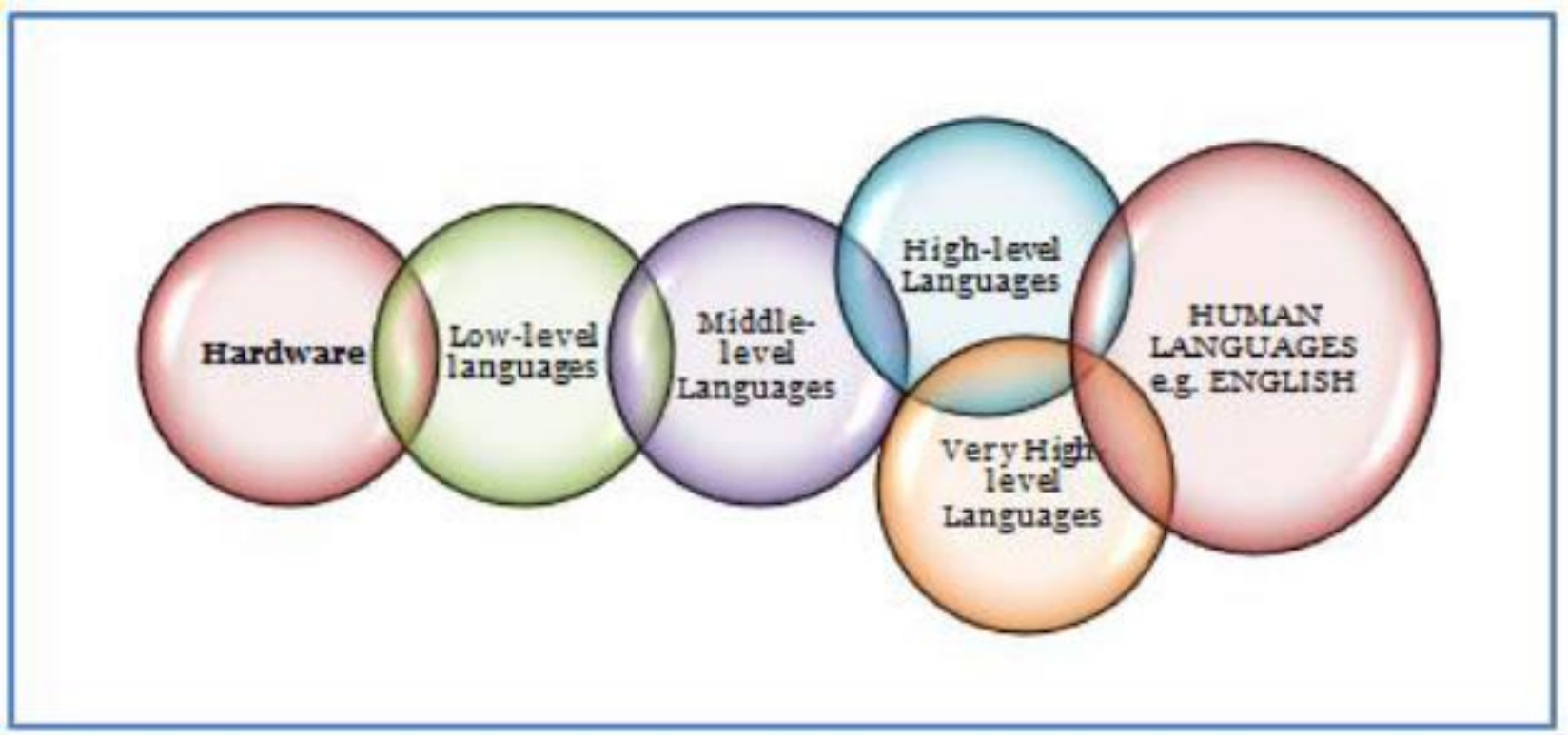
Contoh Bahasa Pemrograman Terkait Paradigma

- Paradigma fungsional: **Haskell**, LISP, Scheme, Erlang, Scala, Miranda…
- Paradigma prosedural: Basic, C, Pascal, Ada, Fortran, COBOL, **Python**, …
- Paradigma berorientasi objek: Eiffel, SmallTalk, Java, C++, C#, …
- Paradigma deklaratif: Prolog
- Paradigma relasional: SQL
- Paradigma event-driven: Visual Basic, Delphi, Visual C++, …
- Paradigma konkuren: Java, C#, Erlang, …
- Beberapa bahasa memiliki kemampuan multi-paradigma: **Python**, Java, …

Taksonomi Bahasa Pemrograman



Overlapping of Languages



Bahasa Mesin (1)

- `1010 0011 0001 1001` (*Machine Language)*
- `ADD  R3,  R1,  R9` (*Assembly Language)*

- Machine dependent
- Fast processing
- Error prone
- Difficult to use or debug, to understand
- Efficient code for the machine

Bahasa Mesin (2)

- No need of translator
- Programmer requires the knowledge of computer architecture
- Need to remember a lot of machine codes
- Need to remember all memory addresses
- Different machine language for the different computer

Bahasa Assembly (1)

Language Code (Machine) (16-BIT INSTRUCTION SET)	*Assembly Language Code (Equivalent)*			
1000000100100101	LOAD	R1	5	
1000000101000101	LOAD	R2	5	
1010000100000110	ADD	R0	R1	R2
1000001000000110	SAVE	R0	6	
1111111111111111	HALT			

- Machine dependent
- Compared to machine lang., easy to understand, to remove error, to modify
- Need of Translator for the execution of the Program

Bahasa Assembly (2)

- Efficient code for the machine
- Fast processing
- Different assembly language for the different computer
- Used for specific applications:
  - used in applications which are cost sensitive (washing machines & music systems) & time critical (aircraft controls).
- Don’t solve user’s all programming problems
- Assembly language programs are not portable

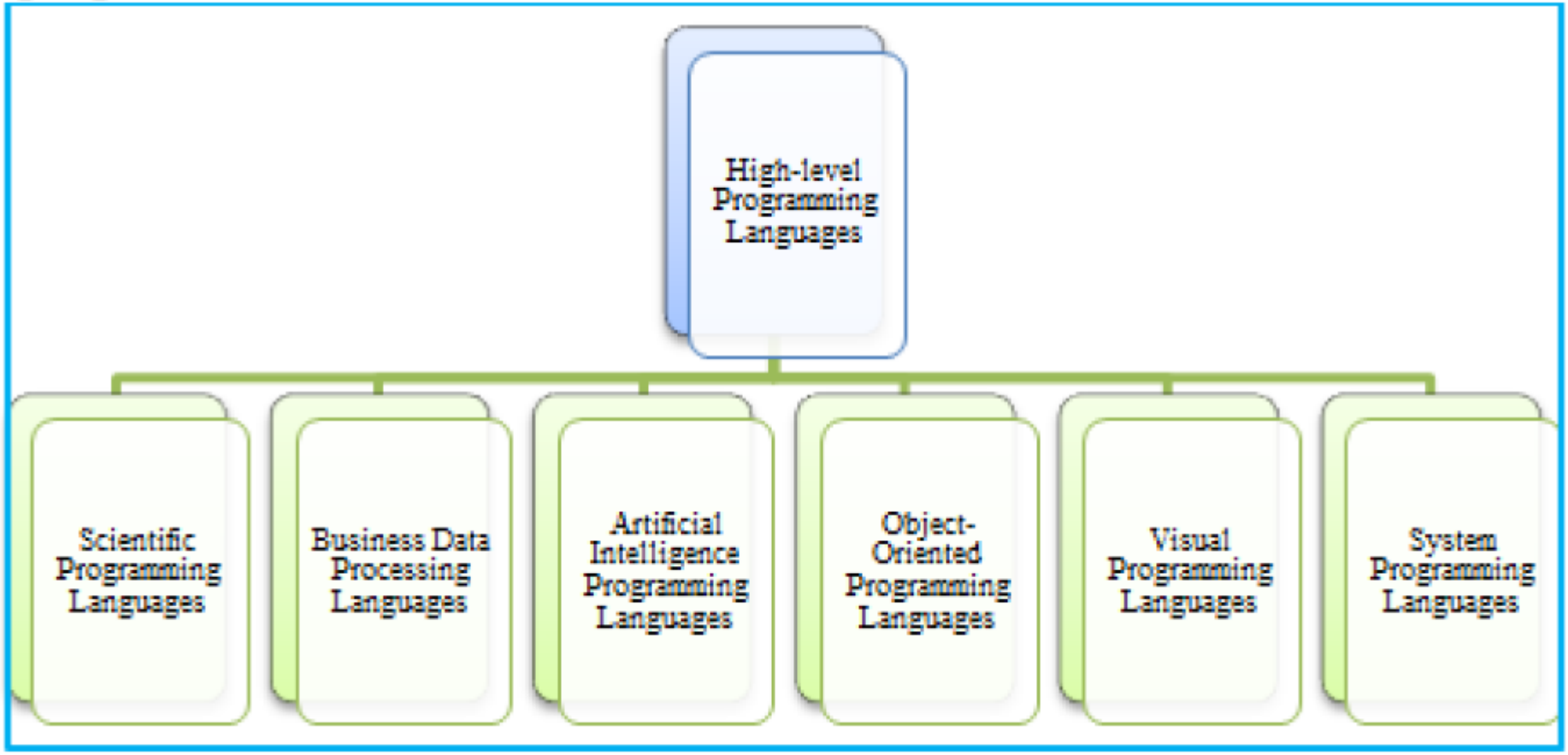
Middle-Level Languages

- Bridge the gap of high-level and low-level languages
- Need more technical skills as compared to the high-level language programmers
- Providing a small set of controlling and data-manipulating instructions which can be utilized by the developers.
- E.g. C, C#, C++, Java

High Level Languages

- Problem-oriented Languages
- Understandability
- Easy to debug
- Portability
- Easy to Use
- User-friendliness
- Little time to write Programs
- Easy Maintenance
- Machine Independence
- Need of a Translator
- Less Efficient

High Level Programming Languages



Very High Level Languages

- 4GL, mostly non procedural
- the users and the developers to describe the results they need
- near to english or other natural languages

Multiply the numbers A and B

And put the result into C

- so much interactive and a dialogue in between the human being and the computer machine is supported

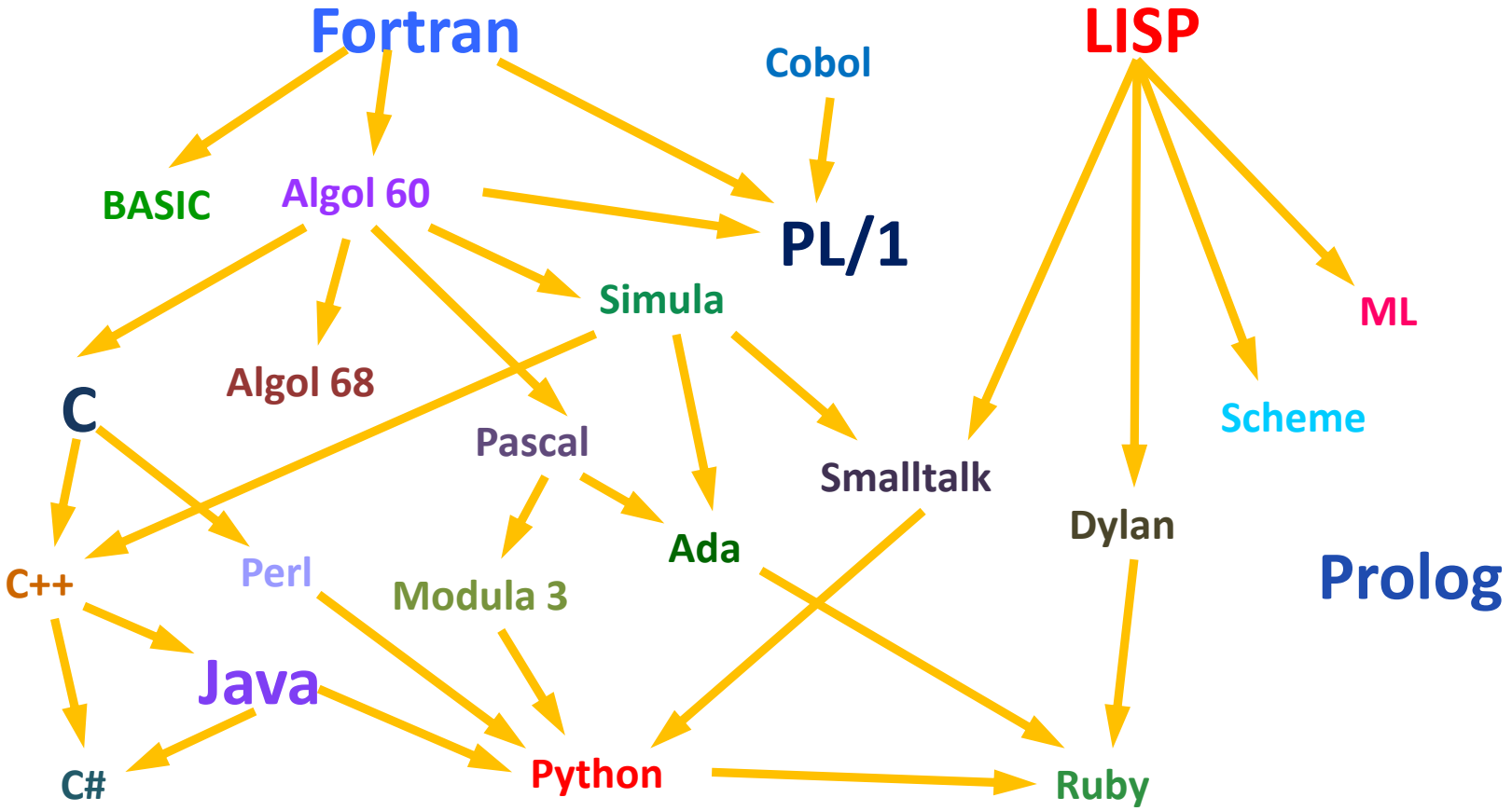
Examples

- Oracle, 4GE (4th Generation Environment):
  - End-user Query Language (e.g. SQL),
  - Screen Formatter (e.g. Oracle‟s screen painter in SQL *Forms),
  - Report Generator (SQL *Report),
- Mathematica
  - Command input is represented in natural languages (text)
- ...

Higher Level Prog. Languages

- 5GL, yet to come
- Use the ideas of AI
- Interface between human being and machine to permit affective use of natural language and images
- Ultimately, the computer will directly understand human beings
- Example:
  - Brain Computer Interface
- Video: Man Controls Robotic Hand with Mind (2:52)

Intermezzo…
A family tree of languages



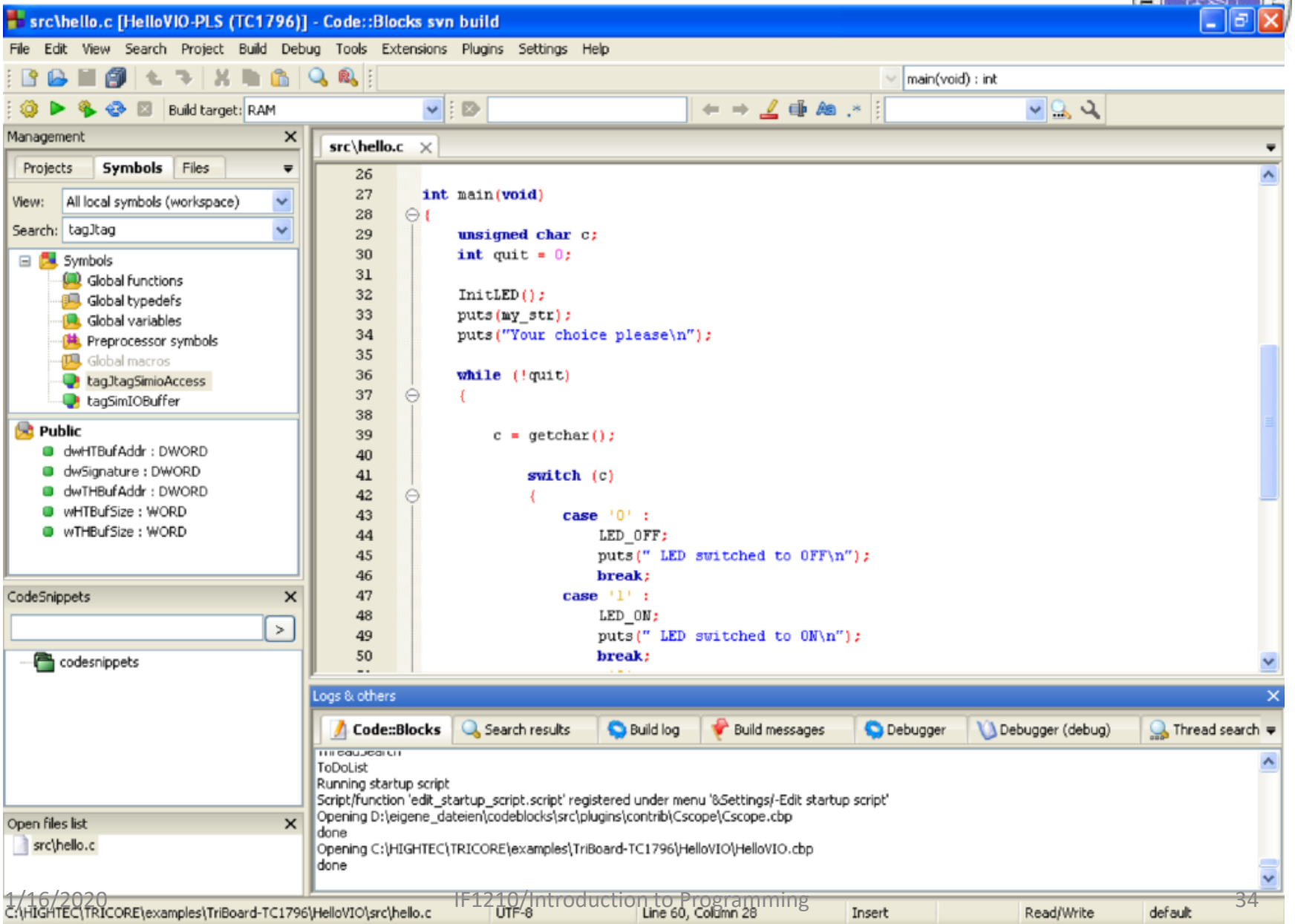
Intermezzo

- Bahasa pemrograman **Pascal** diberi nama berdasarkan matematikawan dan filosofer Perancis, Blaise Pascal
- Bahasa pemrograman **Ada** diberi nama berdasarkan Ada Lovelace
- Bahasa pemrograman **C** merupakan pengembangan dari bahasa pemrograman **B**
- Bahasa **Java** asalnya diberi nama *Oak* (dari pohon ek yang berdiri di depan kantor pembuatnya), lalu diubah menjadi *Green*, lalu baru menjadi *Java* (dari *Java coffee*, yang banyak diminum oleh para pembuatnya)
- Nama Bahasa Python adalah penghargaan ke grup pelawak Britania, Monty Python
  - *Therefore: "An important goal of Python's developers is keeping it fun to use"*

Artefak dan Lingkungan Pemrograman

- Artefak utama: *source code*
  - Didukung oleh dokumentasi
- Lingkungan pemrograman:
  - Editor *source code*: mulai dari editor sederhana s.d. IDE (Integrated Development Environment)
    - Textual vs visual programming
  - Pemroses bahasa: Kompilasi/interpretasi
  - *Runtime environment*

Contoh IDE: Code::Blocks



Pemroses Bahasa

- **Compiler** : menghasilkan *object code*, yang kemudian di link oleh linker menjadi *executable code*
  - Contoh : *compiler* Pascal, *compiler* C
- **Interpreter** : menerjemahkan dan melaksanakan instruksi demi instruksi
  - Contoh : interpreter LISP, interpreter Haskell
- Beberapa bahasa menyediakan kedua kemampuan (kompilasi + interpretasi)

Berbagai Area Pemrograman

- Textual vs visual programming
- Desktop-based vs internet-based programming
- Client server vs N-tier programming
- Online vs batch program
- Program yang berinteraksi dengan device → "device driver"

Skala dan Kompleksitas Program

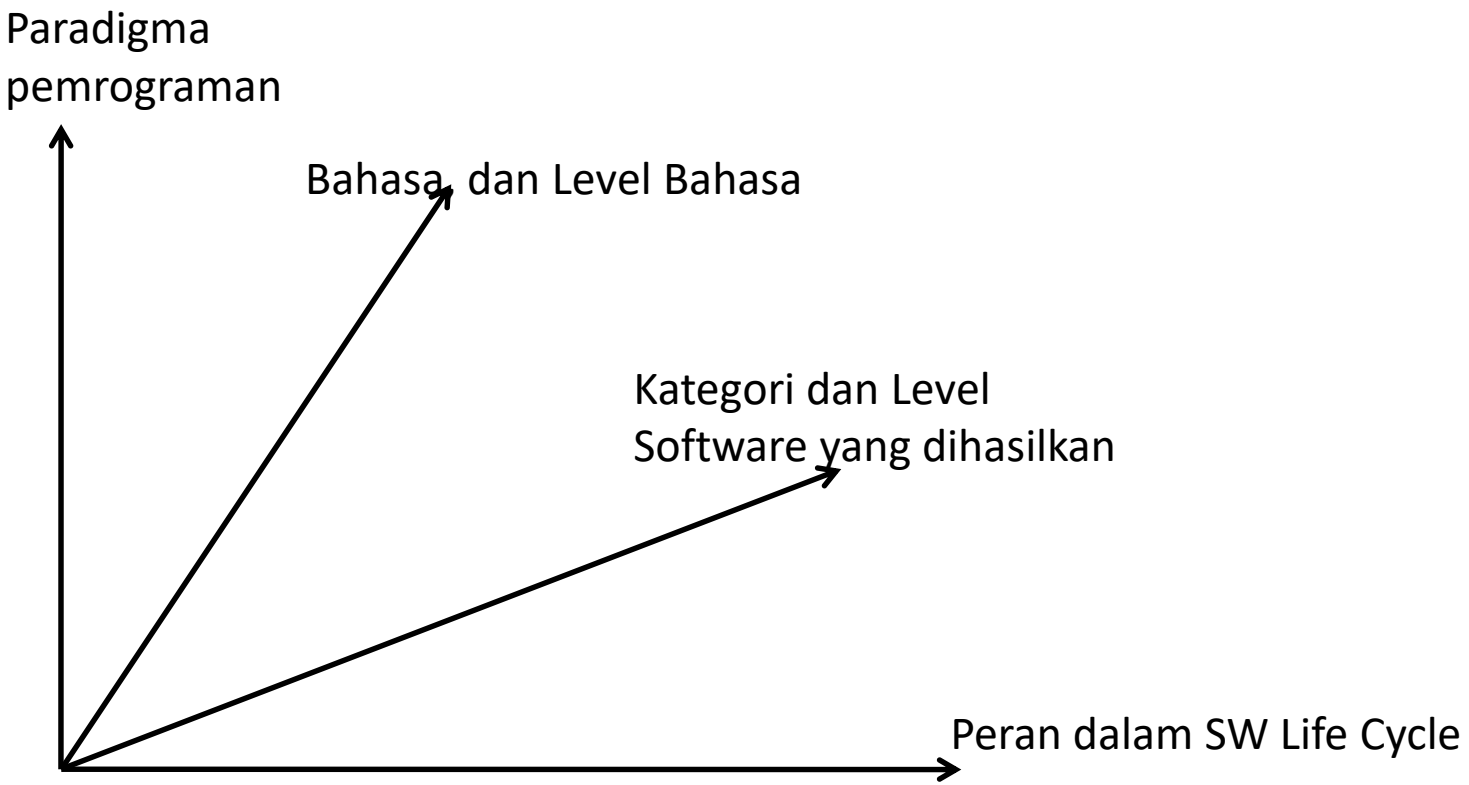
- Skala Program : relatif
  - **Program Kecil** (1 file, 50 LoC, ….)
  - Program Sedang
  - Program Besar
- Kompleksitas program: **algoritma dasar** s.d. algoritma kompleks untuk teknik pemecahan persoalan lanjut (dynamic programming, branch and bound, advanced searching, advanced data structure, ..)
- Akan diajar secara berjenjang

Dipelajari di kuliah IF1210

Jenis-Jenis Programmer

- End user programmer, coder, component user
- Designer dan architect (small/medium/large scale SW, enterprise wide)
- Frame work and Component provider
- CASE Tools programmer
- System programmer
- Tester

Taksonomi Programmer



Level Programmer ☺

- Dead Programmer: Dijkstra, Kay, Jobs
- Successful Programmer: Gates, Carmack, DHH
- Famous Programmer
- Working Programmer
- Average Programmer
- Amateur Programmer
- Unknown Programmer
- Bad Programmer

Programmer Competency Matrix

Computer Science

	2^n (Level 0)	n^2 (Level 1)	n (Level 2)	log(n) (Level 3)
data structures	Doesn't know the difference between Array and LinkedList	Able to explain and use Arrays, LinkedLists, Dictionaries etc in practical programming tasks	Knows space and time tradeoffs of the basic data structures, Arrays vs LinkedLists, Able to explain how hashtables can be implemented and can handle collisions, Priority queues and ways to implement them etc.	Knowledge of advanced data structures like B-trees, binomial and fibonacci heaps, AVL/Red Black trees, Splay Trees, Skip Lists, tries etc.
algorithms	Unable to find the average of numbers in an array (It's hard to believe but I've interviewed such candidates)	Basic sorting, searching and data structure traversal and retrieval algorithms	Tree, Graph, simple greedy and divide and conquer algorithms, is able to understand the relevance of the levels of this matrix.	Able to recognize and code dynamic programming solutions, good knowledge of graph algorithms, good knowledge of numerical computation algorithms, able to identify NP problems etc.
systems programming	Doesn't know what a compiler, linker or interpreter is	Basic understanding of compilers, linker and interpreters. Understands what assembly code is and how things work at the hardware level. Some knowledge of virtual memory and paging.	Understands kernel mode vs. user mode, multi-threading, synchronization primitives and how they're implemented, able to read assembly code. Understands how networks work, understanding of network protocols and socket level programming.	Understands the entire programming stack, hardware (CPU + Memory + Cache + Interrupts + microcode), binary code, assembly, static and dynamic linking, compilation, interpretation, JIT compilation, garbage collection, heap, stack, memory addressing...

Programmer Competency Matrix

Software Engineering

	2^n (Level 0)	n^2 (Level 1)	n (Level 2)	log(n) (Level 3)
source code version control	Folder backups by date	VSS and beginning CVS/SVN user	Proficient in using CVS and SVN features. Knows how to branch and merge, use patches setup repository properties etc.	Knowledge of distributed VCS systems. Has tried out Bzr/Mercurial /Darcs/Git
build automation	Only knows how to build from IDE	Knows how to build the system from the command line	Can setup a script to build the basic system	Can setup a script to build the system and also documentation, installers, generate release notes and tag the code in source control
automated testing	Thinks that all testing is the job of the tester	Has written automated unit tests and comes up with good unit test cases for the code that is being written	Has written code in TDD manner	Understands and is able to setup automated functional, load/performance and UI tests

Programming vs Software Engineering

- Pemrograman adalah salah satu fase dalam *software engineering* (rekayasa perangkat lunak)
- Definisi formal ***software engineering***:

  "the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software"

  IEEE Standard Glossary of Software Engineering Terminology," IEEE std 610.12-1990, 1990

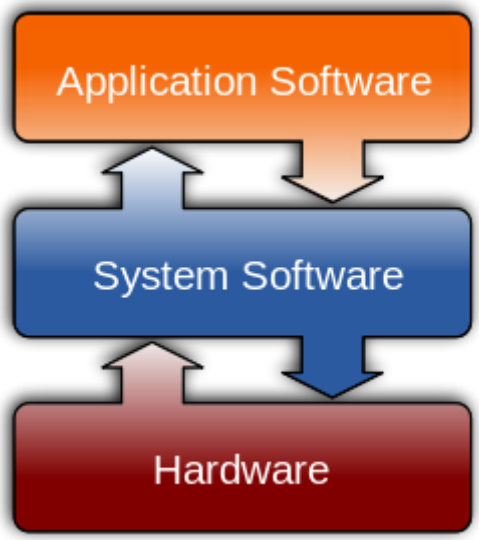
Analogi
Membangun software vs membangun program [kecil]

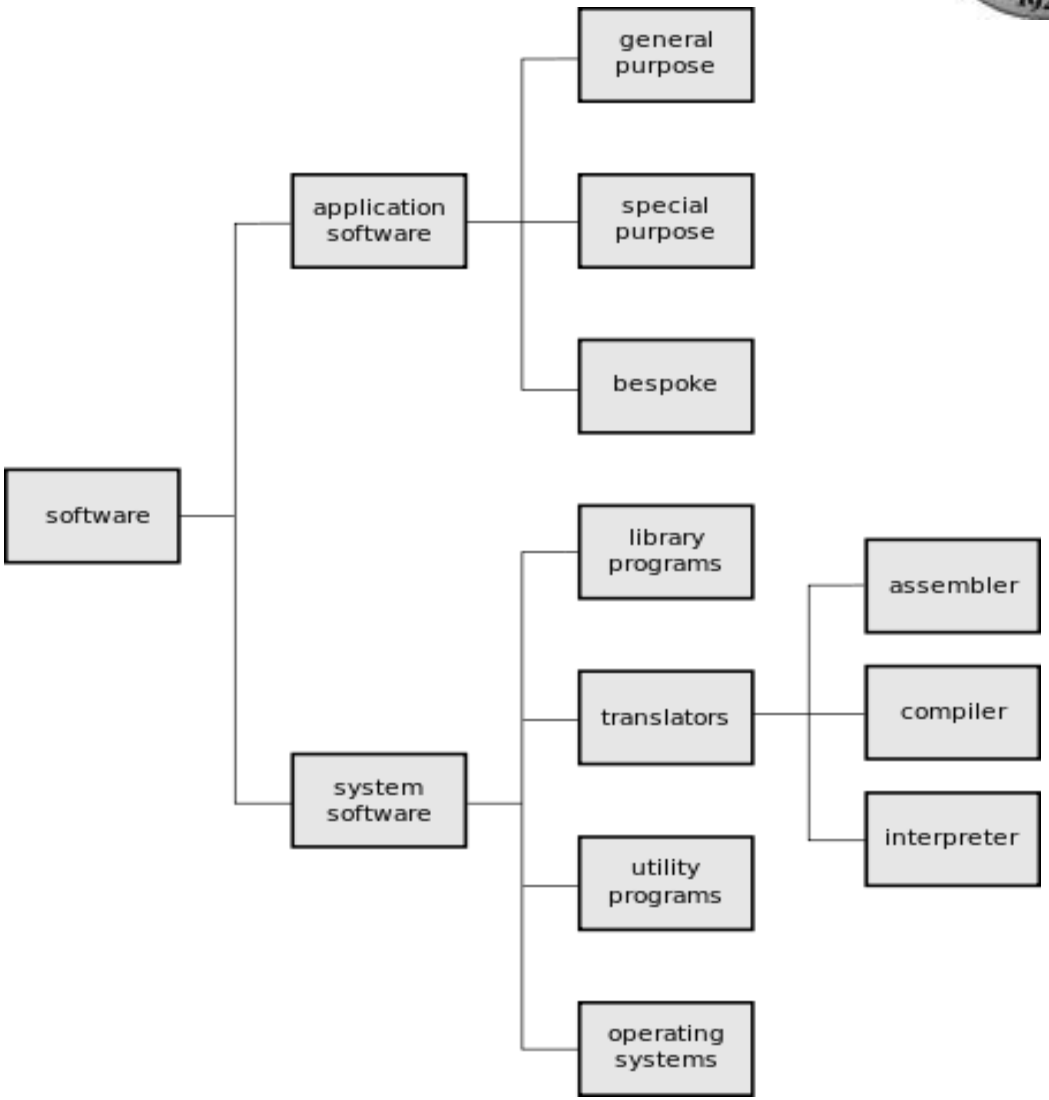
Software diibaratkan pencakar langit

vs

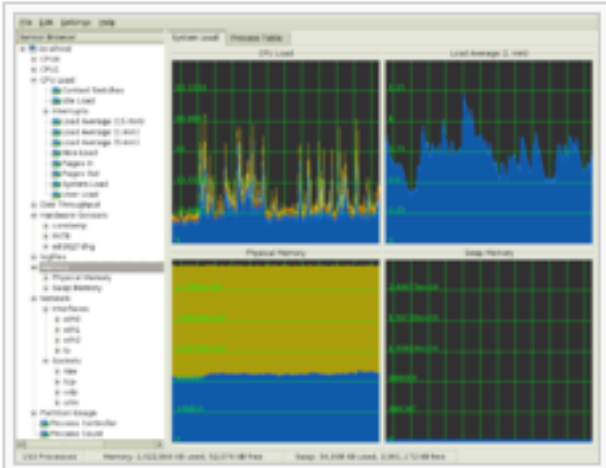
Program [kecil] diibaratkan rumah kecil

Kategori Software





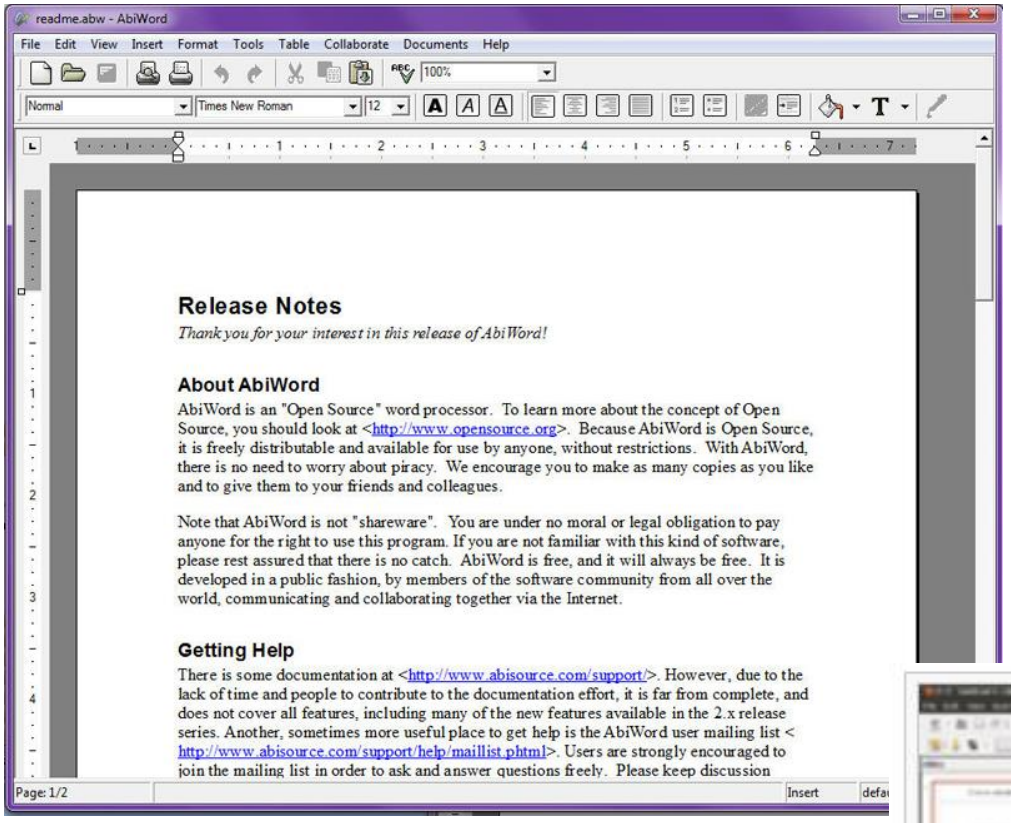
Xfce Graphical User Interface to a Linux operating system

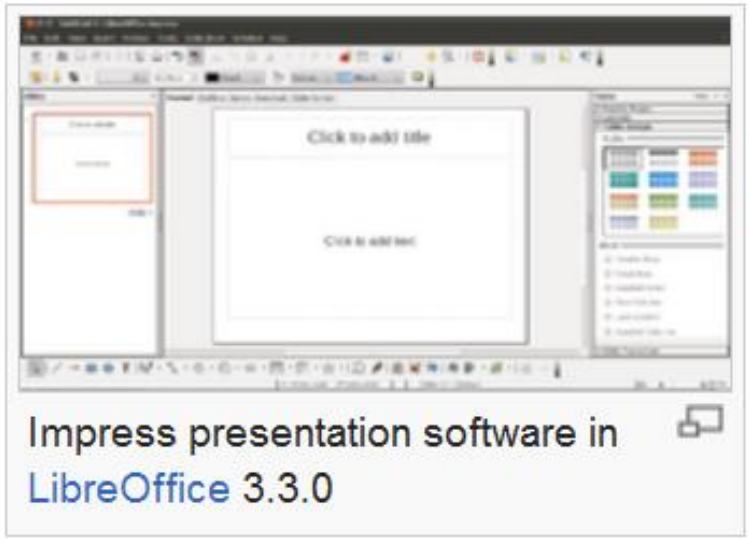


KDE System Guard monitoring the CPU usage, processes and memory usage

Illustration of an application which uses libvorbisfile to play an Ogg Vorbis media file

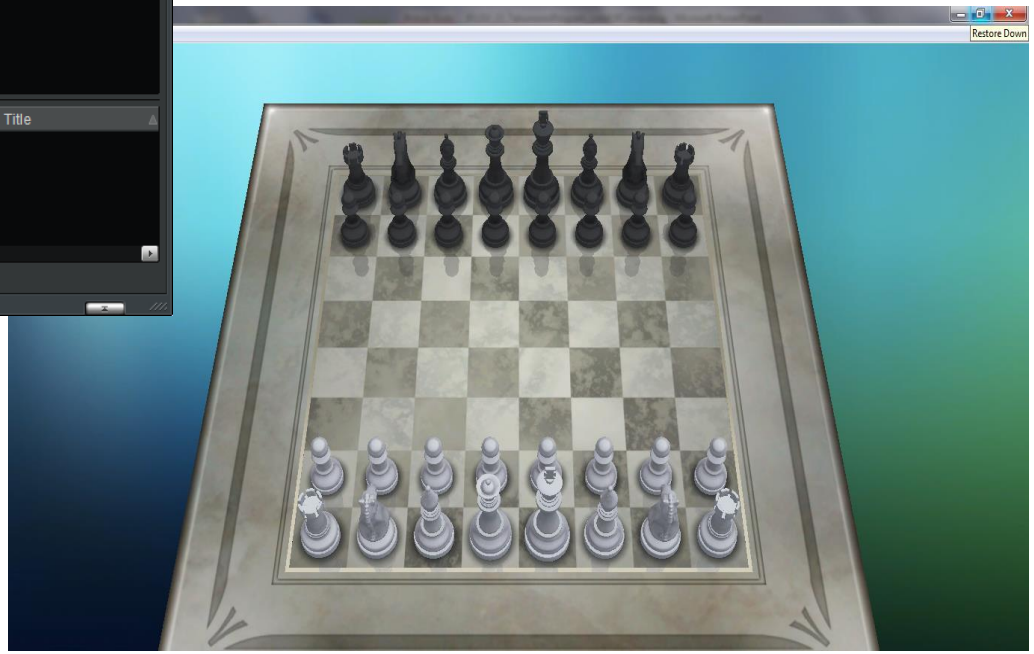
SYSTEM SOFTWARE



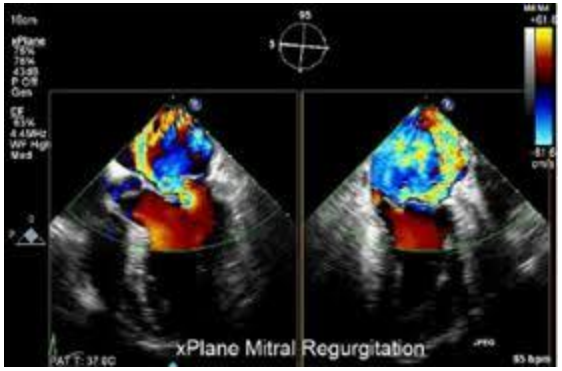


GENERAL PURPOSE SOFTWARE

Special Purpose Software

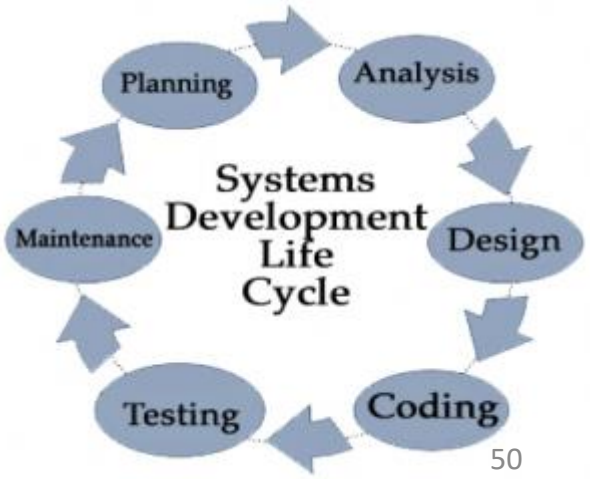


Bespoke Software



Software Engineering Life Cycle

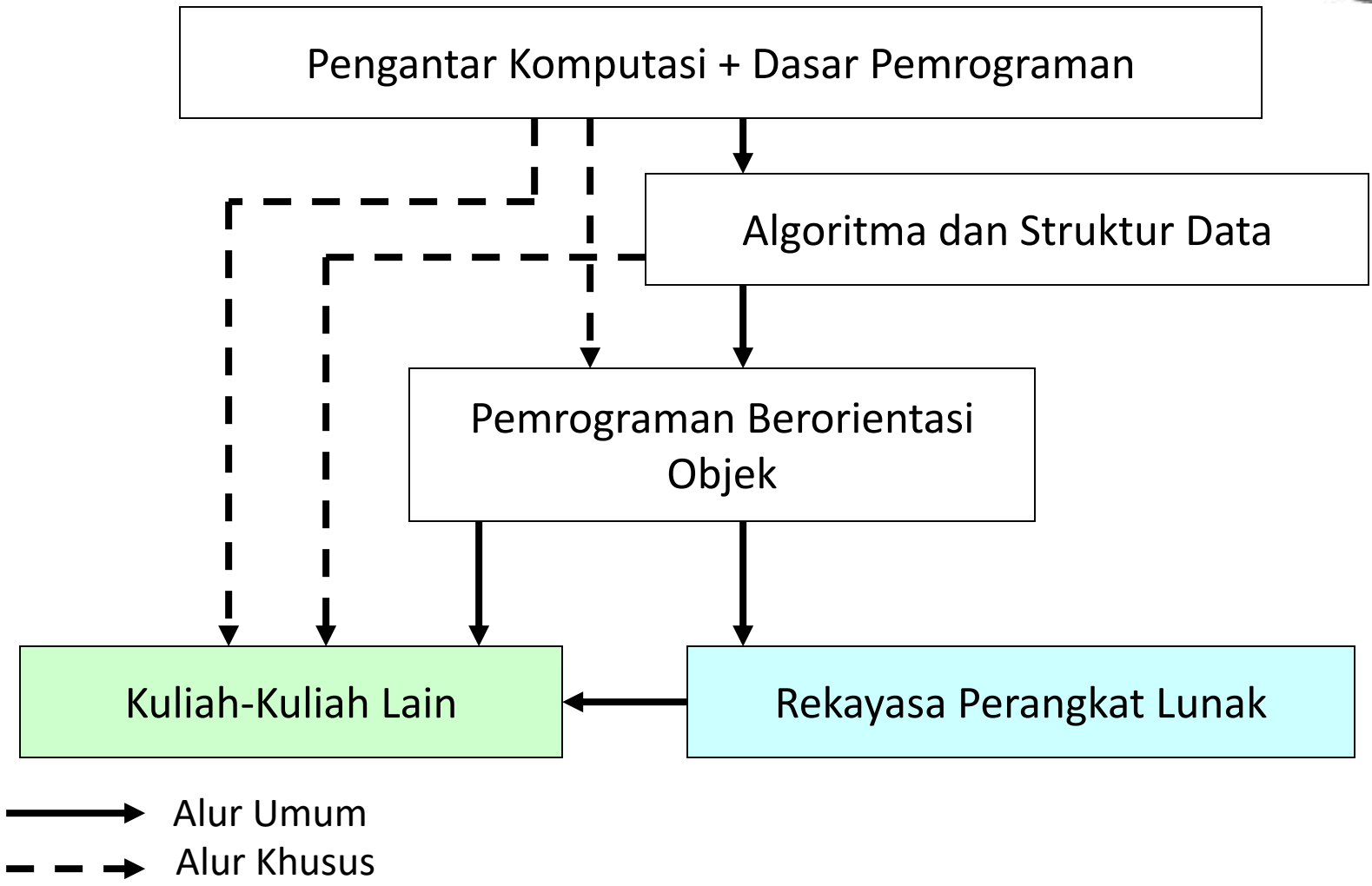
- Requirement analysis
- Software analysis and design
  - Kegiatan pemrograman: Analisis dan penentuan spesifikasi program
- Implementation (coding and debugging)
  - Kegiatan pemrograman: coding dan debugging
- Unit and component testing
  - Kegiatan pemrograman: testing
- Integration and System testing
- Maintenance



Sampai di mana IF1210?

- Kategori Program:
  - Application Software
- Bahasa:
  - Haskell, Pascal
- Paradigma:
  - Fungsional, Imperative/Prosedural
- Skala program:
  - Program kecil
- Kompleksitas program:
  - Algoritma dasar

Alur Pembelajaran Pemrograman di STEI



Alur Pembelajaran Pemrograman

- MK Lanjut terkait pemrograman:
  - Algoritma dan Struktur Data → prosedural, skala menengah
  - Logika komputasional → paradigma deklaratif
  - Basisdata → paradigma relasional
  - Pemrograman berorientasi objek → skala menengah - besar
  - Strategi algoritma → program kompleks, algoritma tingkat lanjut
  - Pemrograman web → lingkungan web/internet
  - Pemrograman platform khusus (misalnya mobile)
  - Dst.

Bahan

- Sumber utama untuk bahan mengenai sejarah pemrograman komputer diambil dari: http://en.wikipedia.org
- Diktat “Dasar Pemrograman, Bag. Pemrograman Prosedural” oleh Inggriani Liem, edisi April 2007
- Slide kuliah “Introduction to Programming and Software Engineering”, oleh Inggriani Liem, Revisi Tim Pengajar IF2030 sem. 1 2010/2011