DD2480

Software Engineering Fundamentals

Programutvecklingsteknikens grunder

7,5 credits

Lecture 1

Outline of Lecture 1

- Part 1: About the course
- Part 2: What is software engineering?
- Part 3: Overview of Assignment 1

Teaching team

- Lecturers

  Cyrille Artho (Course responsible)

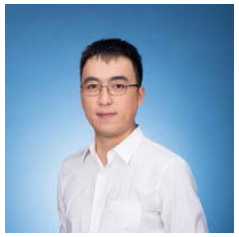
  Elena Troubitsyna

- Teaching assistants

  Mikhail Shcherbakov

  He Ye

  Zhang Long



  Jan van den Brand

  Md Sakib Khab Nizam

Guest lecturers

- Philipp Haller (KTH)
- Stefan Nilsson (KTH)
- To be confirmed
  - Laurent Ploix (Spotify)
  - Ivar Jakobson -- a major contributor to UML, Objectory, Rational Unified Process (RUP), aspect-oriented software development and Essence.

Important URLs

- Course description:

https://www.kth.se/student/kurser/kurs/DD2480?l=en

- Canvas page:

https://kth.instructure.com/courses/4387/

- KTH Github repo:

https://gits-15.sys.kth.se/TCS/DD2480

Important: Registration

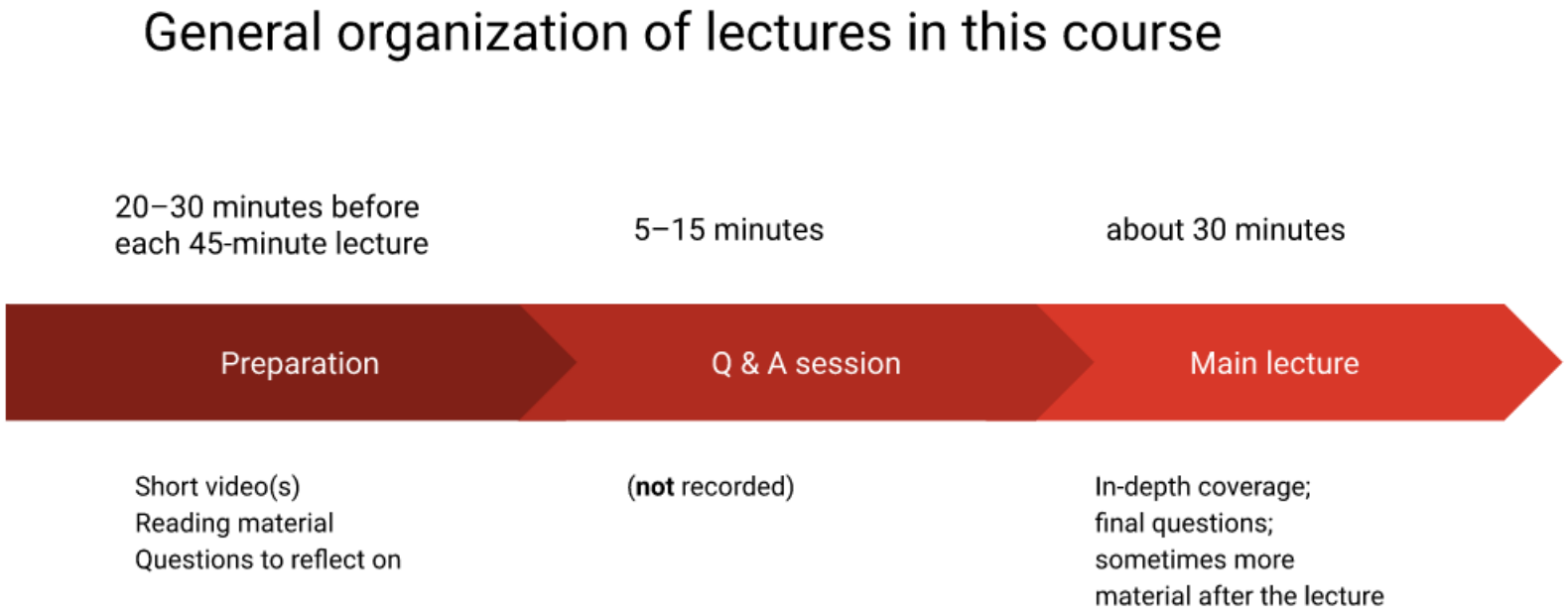
- **REGISTER for DD2480 in LADOK and CANVAS**

- Groups for working on assignments are formed based on registration
  - No registration => No group => No passed assignments

- **You must be enrolled as a student by January 25th at the latest.**
- Note: You are enrolled as a student and not just registered as an observer ("antagna").
- Remind your friends to register too!

Course format

- Lectures are digital: each lecture has a zoom link announced.
  - Active learning approach



Course content

- Requirements engineering.
- Revision control, continuous integration, the life cycle for software.
- Design patterns, components.
- Testing and debugging.
- Software maintenance, configuration management, refactoring.
- Quality assurance: Estimation and measurement of performance and code complexity, scalability.

Intended learning outcomes

After completion of the course, you should be able to:

- apply revision control to a software project,
- systematically test and debug a program,
- combine different types of software testing technologies in a project,
- understand and use patterns for the design and implementation of software,
- deploy quality assurance techniques and judge their results.

Course at a glance

- Eight lectures (incl. guest lectures), 4 assignments.

- 1st assignment 1 worklabs+1 grading lab
- 2nd assignment 2 worklabs+1 grading lab
- 3rd assignment 2 worklabs+1 grading lab
- 4th assignment 2 worklabs+1 grading lab
- Changes are still possible after grading lab
- Meet your group as soon as it is formed

Lectures

- Lecture 1: Overview of the course, what is software engineering
- Lecture 2: Open source software, GitHub
- Lecture 3: Continuous integration; working as a team
- Lecture 4: Refactoring, debugging, software packages
- Lecture 5: Software complexity and test coverage
- Lecture 6: API: what and how
- Lecture 7: Testing
- Lecture 8: Requirements engineering. SEMAT- software engineering methods and theory

Course material

- No single text book
- For each lecture: the links to the relevant reading material and/or video
- All the links are available in Canvas

Assignments

- Assignment #1: DECIDE -- Implement a program using modern development techniques
- Assignment #2: Continuous Integration
- Assignment #3: Test coverage
- Assignment #4: Refactoring

- Done in groups

Group membership

- Groups are created randomly
- It simulates real-life conditions: in a company, software engineers can't choose their co-workers.
  - Additionally, we want to ensure that everyone is assigned to a group, exchange students are mixed with local students and groups have diverse skill sets.
- Fairness in assessment: for each assignment, there should be a "statement of contributions“

- Groups will be published in Canvas (People -> Groups) before the first lab
- Contact each other immediately
- Contacting your groupmates and coordination inside the group is your own responsibility

Grading of Assignments

- Grading criteria for each individual assignment (see Assignments in Canvas)
  - Grade per group (unless exceptions)
- The grades will be given and reported in Canvas directly.
- Assignments are mini-projects which are graded:

  - F: Fail (so bad that the course is failed). For example, if nothing is done.
  - Fx: Fail with a chance of trying again later during the course.
  - P: Pass.
  - P+: Pass with distinction.

- To get P+ the assignment should be submitted in time

Course grade

- Course grade is based on the grades of four assignments
- Each part has to be passed (so a P+ cannot compensate for Fx or F)
- Final grade formula

4xP+ = A

1xP + 3xP+ = B

2xP + 2xP+ = C

3xP + 1xP+ = D

4xP = E

Failure in any lab = F (failure)

Grading scheduling

- Grading scheduling is done by filling the group name in the appropriate shared spreadsheet, it must be done at least 24 hours in advance
- All group members must attend the grading session in order to have a better discussion
- Rescheduling is only possible if asked by email at dd2480@kth.se 24 hours in advance
- Generally rescheduled grading can be arranged within ±2 days compared to the original date
- If the grading is after the original deadline, the assignement reports still have to be submitted by the original deadline.
- Rescheduling only works for good reasons, like time conflicts with other courses, medical reasons, etc
- Rescheduling due to sick leave is only accepted with an official document, and must be announced by email at dd2480@kth.se as soon as possible

Communication

- Main communication channel is Canvas
- For grading sessions related matters: dd2480@kth.se

Programming languages

- You will need to deliver a real software written in a real programming language
- Preferred language is Java
- Other mainstream languages are also supported
- If you are consider using less common languages please discuss it with course examiner – Cyrille Artho – first.

Important: Registration

- **REGISTER for DD2480 in LADOK and CANVAS**

- Groups for working on assignments are formed based on registration
  - No registration => No group => No passed assignments

- Y**ou must be enrolled as a student**
- Note: You are enrolled as a student and not just registered as an observer ("antagna").
- Remind your friends to register too!

Special needs

“If you have a disability, you may receive support from Funka, KTH’s coordinator for students with disabilities,
see https://www.kth.se/en/student/studentliv/funktionsnedsattning .
Please inform the course coordinator if you have special needs and show your certificate from Funka.

- Support measures under code R (i.e. adjustments related to space, time, and physical circumstances) are generally granted by the examiner.

- Support measures under code P (pedagogical measures) may be granted or rejected by the examiner, after you have applied for this in accordance with KTH rules. Normally, support measures under code P will be granted”.

Questions about course planning?

Part 2: What is software engineering?

Software… is everywhere

- "…if you bought a premium-class automobile recently, it probably contains close to 100 million lines of software code…"
- The global software engineering market is expected to grow approximately at USD 37.4 Billion by 2022

What is software?

What is software?

- Code
- Documentation, user manuals
- Designs, specifications
- Test cases
- Plans and schedules
- Models, proofs

What is software engineering?

Software Engineering: various definitions

- "*Multi-person construction of multi-version software*." [Parnas 1974]

- IEEE's 2010 definition: software engineering is "the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is, the application of engineering to software". [IEEE 2010]

Sources:
[Parnas 1974] D.L. Parnas.Software engineering or methods for the multi-person construction of multi-version programs - IBM Germany Scientific Symposium Series, 1974 - Springer
[IEEE 2010] ISO/IEC/IEEE 24765:2010 Systems and Software Engineering—Vocabulary.

Software Engineering: definitions

- *"A discipline whose aim is the production of fault-free software, delivered on-time and within budget, that satisfies the user's needs. Furthermore, the software must be easy to modify when the user's needs change."* [Schach. Object-Oriented and Classical Software Engineering]

Programming versus software engineering

- Is there a difference?

Programming versus software engineering

OR

Programming versus software engineering

Small project	Large to huge project
You	Teams
Build what you want	Build what they want
One product	Family of products
Few sequential changes	Many parallel changes
Short-lived	Long-lived
Cheap	Costly
Small consequences	Large consequences

Programming **Software engineering**

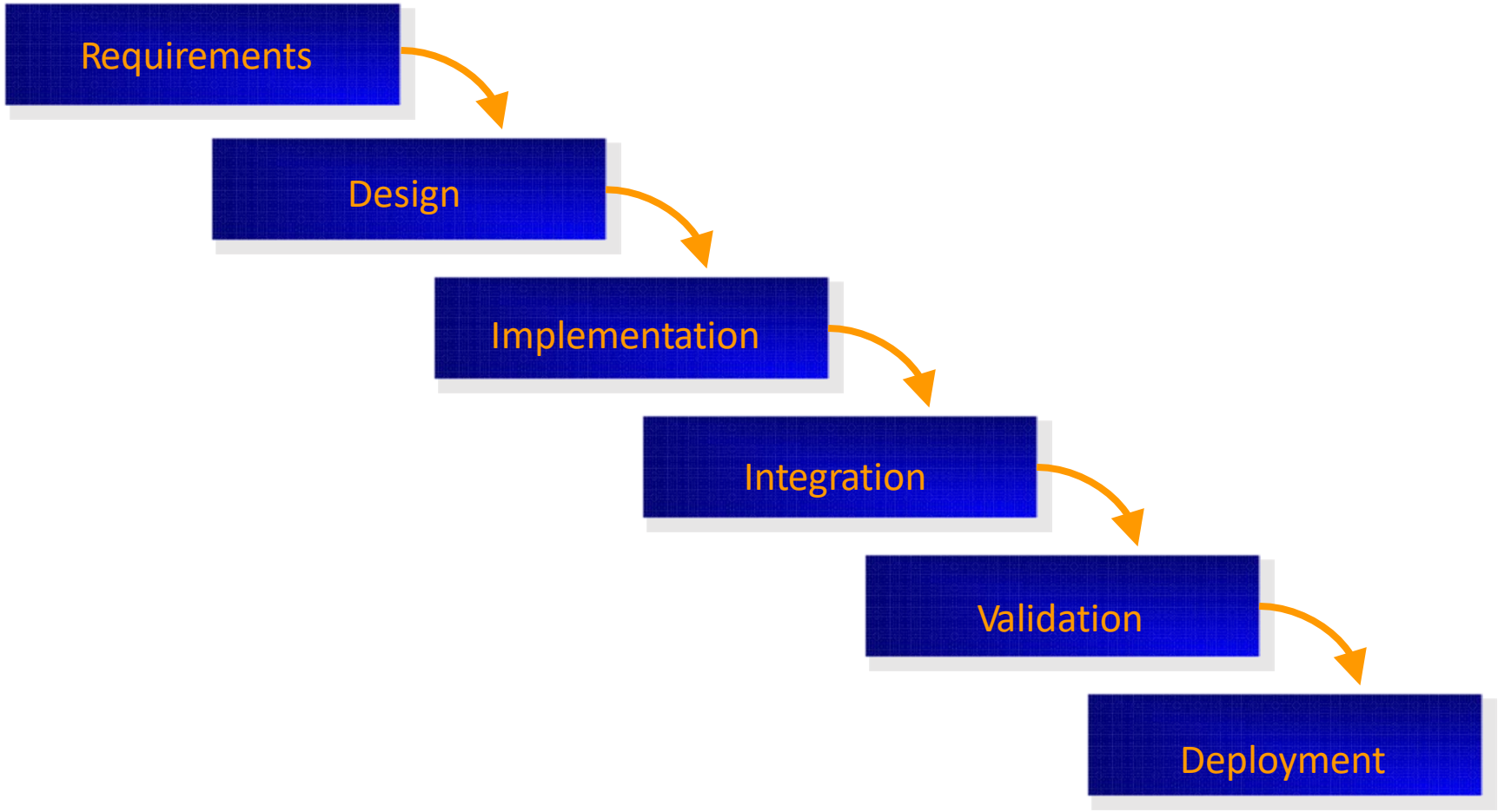
Science, Engineering, Management, Human Factors

- Science: empirical studies; theories characterizing aggregate system behavior (e.g. reliability)
- Management: organizing teams, directing activities, correcting problems
- Human factors: user task understanding and modeling; ergonomics in user interface design
- Engineering: tradeoffs, pattern solutions to typical problems
  - Tradeoffs and representative qualities
    - Pick any two:
      - Good, fast, cheap
      - Scalability, functionality, performance

Software Engineering Principles

- Rigor and Formality
- Separation of Concerns
- Modularity and Decomposition
- Abstraction
- Anticipation of Change
- Generality
- Incrementality
- Reliability

Software development process - Waterfall Model



Waterfall Model

- The waterfall model is the classic lifecycle model – it is widely known, understood and (still often) used.
- In some respect, waterfall is the ”common sense” approach.
- Introduced by Royce 1970.

Advantages of waterfall model

- Easy to understand and implement
- Still widely used and known
- Reinforces good habits: define-before-design, design-before-code
- Identifies deliverables and milestones
- Document driven and uses known documentation standards
- Works well on mature products and weak teams.

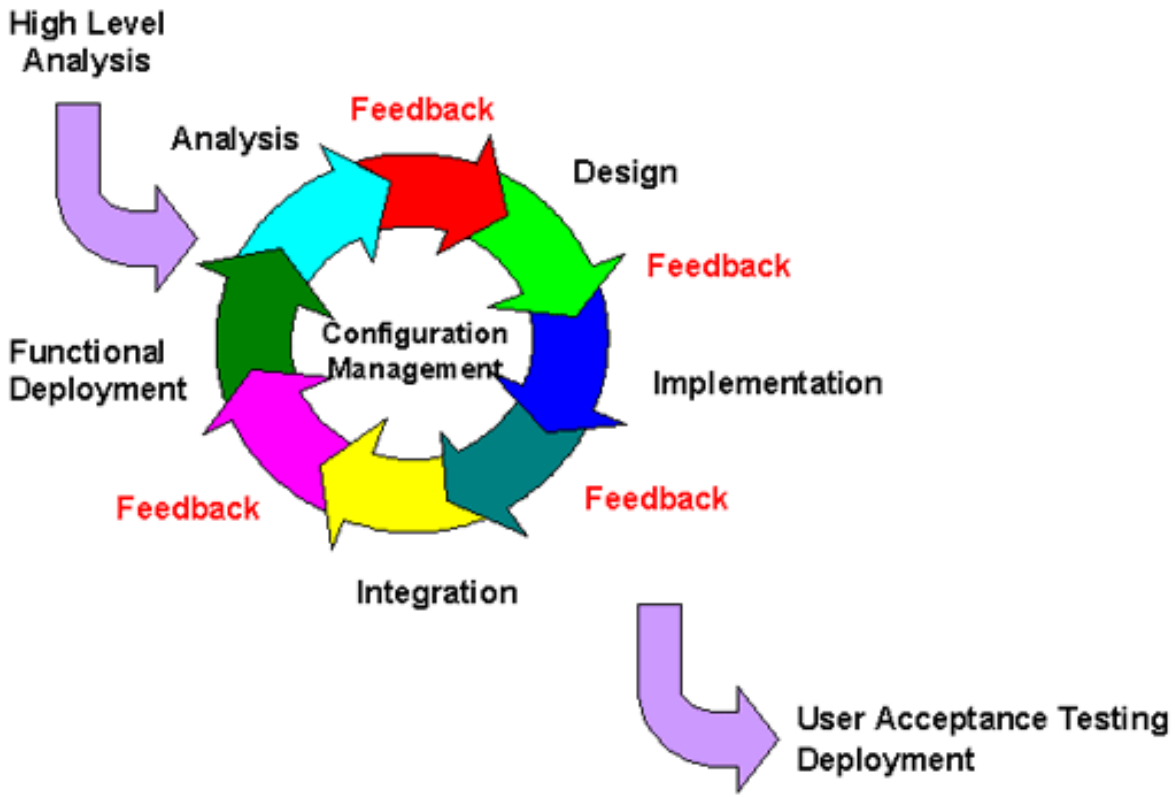
Disadvantages of waterfall model

- Idealised – unrealistic idea of ”fixed” requirements
- Doesn’t adhere to iterative nature of exploratory software development
- Precise requirements from customers are rare
- Postpones writing actual software – late problem discovery
- Hard to cope with the unforeseen risks and manage risks
- Difficult and expensive to make changes to documents
- Might be too expensive for small teams and projects

The idea of iterative development

- The main idea is to develop software in an ***experimental*** way continuously asking for the customer's feedback
- Iterative – spiral – model defined by Boehm in 1988
- Motivated by the need of iterative ***risk analysis*** and ***risk management***.

Software development process - Iterative development at glance



Project Iteration Flow

Pros and cons of iterative development

- Each cycle follows a waterfall model
  Determining objectives -> Specifying constraints -> Generating alternatives -> Identifying risks -> Resolving risks -> Developing next-level product -> Planning next cycle

Pros:

- Reflects the explorative nature of software development
- Can cope with not fully defined requirements
- Provides flexibility

Cons

- Needs good technical expertise in risk analysis
- More difficult to understand for management

Agile development

Agile manifesto (2001)

XP = Extreme Programming emphasises:

Individuals and interactions

Over processes and tools

Working software

Over documentation

Customer collaboration

Over contract negotiation

Responding to change

Over following a plan

Agile principles

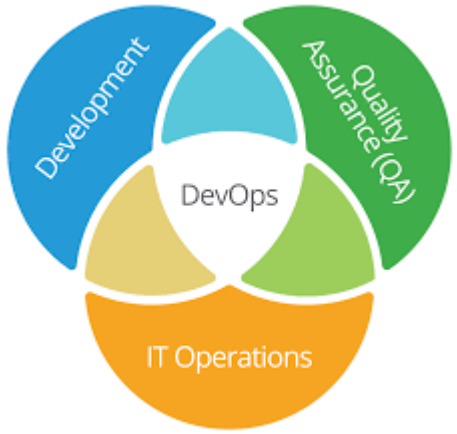
- Continuous delivery of software
- Continuous collaboration with customer
- Continuous update according to changes
- Value participants and their interaction
- Simplicity in code, satisfy the spec

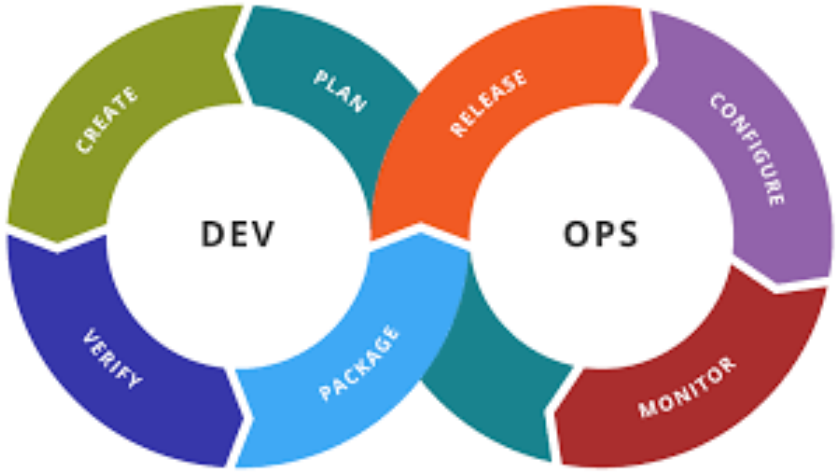
Software development process -- Agile development

Variety of practices

- Programming in pairs
- Test driven development
- SCRUM, KANBAN
- Feature-driven development
- Dynamic Systems Development Method
- Continuous planning, change , delivery
- Shared coding standards and ownership of code

Software development process (DevOps)





Factors affecting the quality of a software system

- **Complexity:**
  - The system is so complex that no single programmer can understand it anymore
  - The introduction of one bug fix causes another bug
- **Change:**
  - The “Entropy” of a software system increases with each change: Each implemented change erodes the structure of the system which makes the next change even more expensive (“Second Law of Software Dynamics”).
  - As time goes on, the cost to implement a change will be too high, and the system will then be unable to support its intended task. This is true of all systems, independent of their application domain or technological base.

Subdisciplines of software engineering (1/2)

- Software requirements (Requirements engineering)
  - The elicitation, analysis, specification, and validation of requirements for software.
- Software design
  - The process of defining the architecture, components, interfaces, and other characteristics of a system or component.
- Software construction
  - The detailed creation of working, meaningful software through a combination of programming (aka coding), verification, unit testing, integration testing, and debugging.
- Software testing
  - An empirical, technical investigation conducted to provide stakeholders with information about the quality of the product or service under test.

Subdisciplines of software engineering (2/2)

- Software maintenance
  - The totality of activities required to provide cost-effective support to software.
- Software configuration management
  - The identification of the configuration of a system at distinct points in time for the purpose of systematically controlling changes to the configuration, and maintaining the integrity and traceability of the configuration throughout the system life cycle. Software versioning
- Software engineering management
  - The application of management activities—planning, coordinating, measuring, monitoring, controlling, and reporting—to ensure that the development and maintenance of software is systematic, disciplined, and quantified.
- Software development process
  - The definition, implementation, assessment, measurement, management, change, and improvement of the software life cycle process itself.
  - Software engineering models and methods impose structure on software engineering with the goal of making that activity systematic, repeatable, and ultimately more success-oriented
- Mathematical foundations
- …

Scientist vs Engineer

- Computer Scientist
  - Proves theorems about algorithms, designs languages, defines knowledge representation schemes
  - Has infinite time…
- Engineer
  - Develops a solution for an application-specific problem for a client
  - Uses computers & languages, tools, techniques and methods
- Software Engineer
  - Works in multiple application domains
  - Has less than 3 months...

  … while changes occurs in requirements and available technology

Part3. Assignment #1. “DECIDE”

- Goal: write a function called DECIDE()
  - A part of a hypothetical anti-ballistic missile system
- Generates Boolean signal to launch or not an interceptor based on input radar tracking info
- Info is available at the instant the function is called
- 15 Launch Interceptor Conditions (LIC)
- Function determines which LICs are true for the input (up to 100 of planar data points representing radar echoes)
- Decision to launch made only if all relevant combinations of LICs met

Main concepts: LIC

- 15 LICs are defined in the requirements document.
- Each LIC represents a certain characteristic of radar tracking data

Examples:

LIC 0:

There exists at least one set of two *consecutive* data points that are a distance greater than the length, **LENGTH1**, apart where (0 ≤ **LENGTH1**)

LIC 1:

There exists at least one set of three *consecutive* data points that cannot all be contained within or on a circle of radius **RADIUS1**, where (0 ≤ **RADIUS1**)

…LIC 14 …

- Check the glossary of the terms

Main concepts: data points

- NUMPOINTS – number of planar data points
- POINTS – array containing the coordinates (X,Y) of data points
- Dynamic input parameters (available when DECIDE() is activated)
- LICs are evaluated over the data points (radar data)

Main concepts: CMV

- The *Conditions Met Vector* (CMV)
- Intermediate result
- Set according to the results of LIC calculations
- The global array element CMV[i] should be set to true if and only if the i_{th} LIC is met.

Main concepts: LCM

- Logical Connector Matrix (LCM) defines which individual LICs must be considered jointly (static input parameter)
- 15x15 symmetric matrix with elements ANDD, ORR, NOTUSED

LIC	0	1	2	3	4	...	14
0	ANDD	ANDD	ORR	ANDD	NOTUSED	...	NOTUSED
1	ANDD	ANDD	ORR	ORR	NOTUSED	...	NOTUSED
2	ORR	ORR	ANDD	ANDD	NOTUSED	...	NOTUSED
3	ANDD	ORR	ANDD	ANDD	NOTUSED	...	NOTUSED
4	NOTUSED	NOTUSED	NOTUSED	NOTUSED	NOTUSED	...	NOTUSED
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
14	NOTUSED	NOTUSED	NOTUSED	NOTUSED	NOTUSED	...	NOTUSED

Main concepts: FUV

- The *Final Unlocking Vector* (FUV)
- Intermediate result
- Generated from the *Preliminary Unlocking Matrix*.
- PUV indicates whether the corresponding LIC should be considered as a factor in signaling interceptor launch.
- FUV[i] should be set to true if PUV[i] (ith column of PUM) is false (indicating that the associated LIC should not hold back launch)
- or if all elements in PUM row i are true.

Main concepts: LAUNCH

- **LAUNCH** Final launch / no launch decision encoded as ”YES”, ”NO” on the standard output.
- The final result
  - is based on the FUV.
- “YES” if all elements in the FUV are true,

i.e., if and only if FUV[i] is true for all i, $0 \leq i \leq 14$.

Working on the assignment

- Goal is to implement the DECIDE program according to the modern development techniques.
- Grading focuses more on the software engineering part of the assignment rather than on the program itself
- You must use a development platform (Github, Bitbucket or KTH Github).
- Only the content published on the development platform is used for grading.
  - 1) the code 2) the issues 3) the pull requests 4) the continuous integration data (eg Travis) if available.

Final notes

- LICs are good units to split the work
- Ensure even distribution of work (simple and complex LICs)
- Organise your groups as soon as possible

- Grading criteria for the assignment in Canvas

- Observe the code of conduct

- Good luck!