HPES
High Performance
Embedded Systems

EEE4120F

High Performance Embedded Systems

Many of these aspects will be removed from 2023 exam syllabus

Lecture 23

HPES Development Process and Management Aspects

Spiraling to success

**** Relates to Martinez, Bond and Vai Ch 4.***

Lecturer:
Simon Winberg

Today's theme –

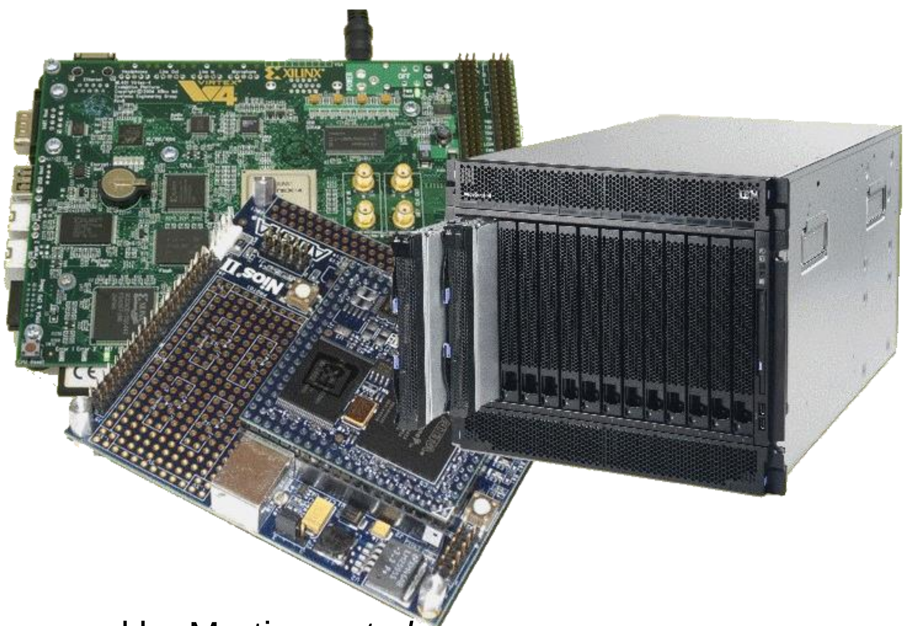
HPES development process and methodology

– towards happy projects

Lecture Overview

- Where work is done, division of labour
- HPEC* management
- HPES development process
- Setting system objectives
- Costs & risks
- Monitoring progress
- Documentation
- Effort, Productivity and Progress

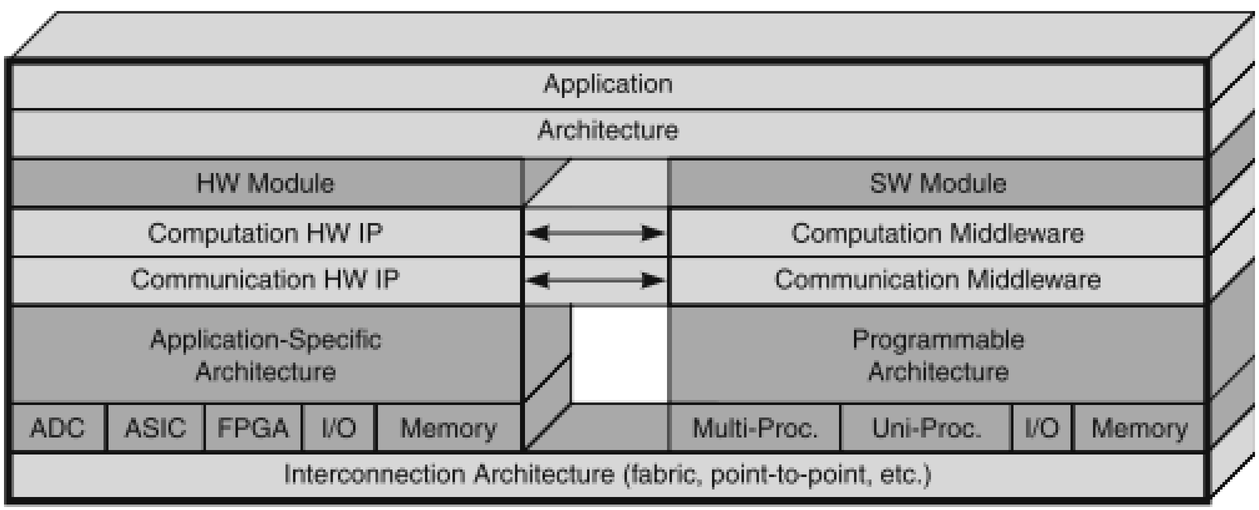
(Optional extra slides re intro to Doxygen)



*HPEC = High Performance Embedded Computer, term used by Martinez *et al.*

HPEC System – where work is done

- Useful to consider 'where work is done' in relation to Martinez *et al.*'s "canonical framework" (illustrated below) that identifies key subsystems and components of a High Performance Embedded Computing (HPEC) system…
- These projects many members of the development team, involved at various levels of the system.



HPES Design: Personnel and Division of Labour

- HPES system development is influenced by the usual suspects:
  - requirement, plans, and implementation decisions for the systems.
- There is likely separation between significant subsystems, e.g. between backend and frontend, as well as between hardware and software.
- May draw on a range of experts from various disciplines ... (lets consider some ...)

HPES Design: Personnel and Division of Labour

- Application expects e.g.
  - Radar experts to advise on radar system design & processing algorithms
  - Medical system experts to design on standards, fault tolerance and safety requirements
  - Masses of others…

HPES Design: Personnel and Division of Labour

- Application expects e.g.
  - radar experts to advise on radar system design & processing algorithms
  - Medical system experts to design on standards, fault tolerance and safety requirements
- Hardware specialists e.g.
  - Computer platform design experts
  - Radio Frequency (RF) experts for design of the RF hardware
  - Experts to design of power supplies to provide the power needed by the system

HPES Design: Personnel and Division of Labour

- Application expects e.g.
  - radar experts to advise on radar system design & processing algorithms
  - Medical system experts to design on standards, fault tolerance and safety requirements
- Hardware specialists e.g.
  - Computer platform design experts
  - Radio Frequency (RF) experts for design of the RF hardware
  - Experts to design of power supplies to provide the power needed by the system
- Software & HDL specialists e.g.
  - Personnel experienced in high performance signal processing
- And these individuals need to work effectively together on the system being constructed…

Roles for division of labour

- Manager
- Team leader
- Hardware designer
- Software designer
- Implementer / Programmer
- Engineering Technician
- Test Designer, Test Analyst
- Tester
- Tool specialist
- Documentation writer
- Librarian

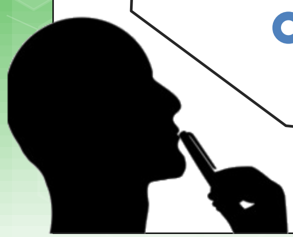
These are also some ideas for the roles you might give team members in the YODA project

HPES Management

- The management of the development must be tailored to meet the particular technology choices, which could comprise a variety of technologies and tools…
- Each technology has its own development cycle, cost, technical limitations, and risks. E.g.:

Some thinking points

  - Developing a custom ASIC
    - tends to slow down the implementation.
    - Could have high risk (e.g. cannot afford more than one run prior to initial release);
    - Need strategies to mitigate these potential risks (e.g. backup plan of using an FPGA)
  - Programmable processors
    - also have various risks (e.g. licensing tools, what happens if they go out of production).

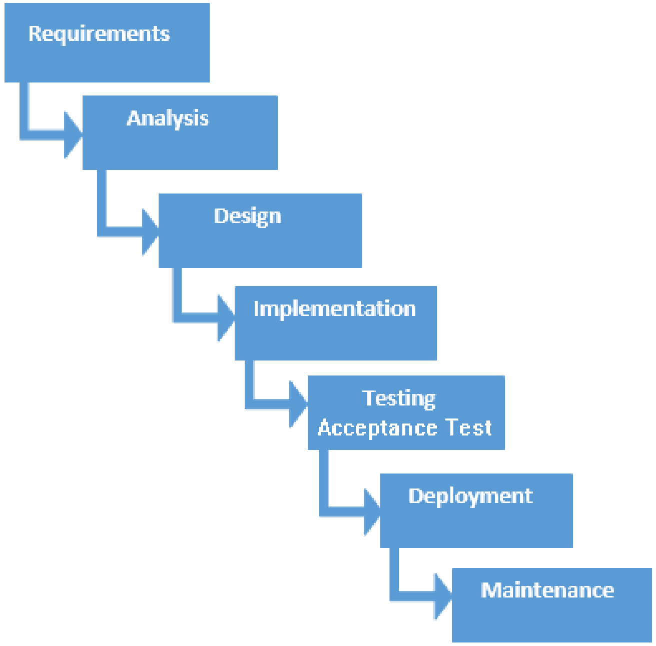


HPES Development Process

EEE4120F

HPES Process Model

- Commonly planned broadly with the waterfall model in mind to cover all needed steps
- Relevant to both the software and hardware aspects of the system
- In practice
  - The spiral model provides a better guide to HPES projects overall, with the waterfall model being applicable to iterations of HW/SW development within cycles of the spiral model

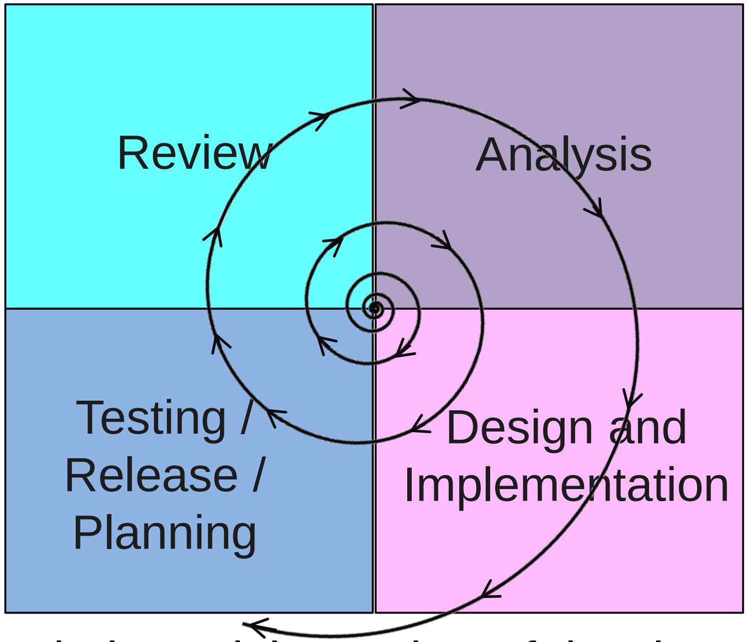


Classic representation of the Waterfall model

Spiral Model: the natural way

- As per general development projects, HPES, Reconfigurable and High-performance computing systems tend to follow the Spiral Model (Bloehm 1988) with phases of…

Starting small (i.e. start from centre of spiral and expand out) with little risk. Adding features and mitigating risk with each additional iteration.

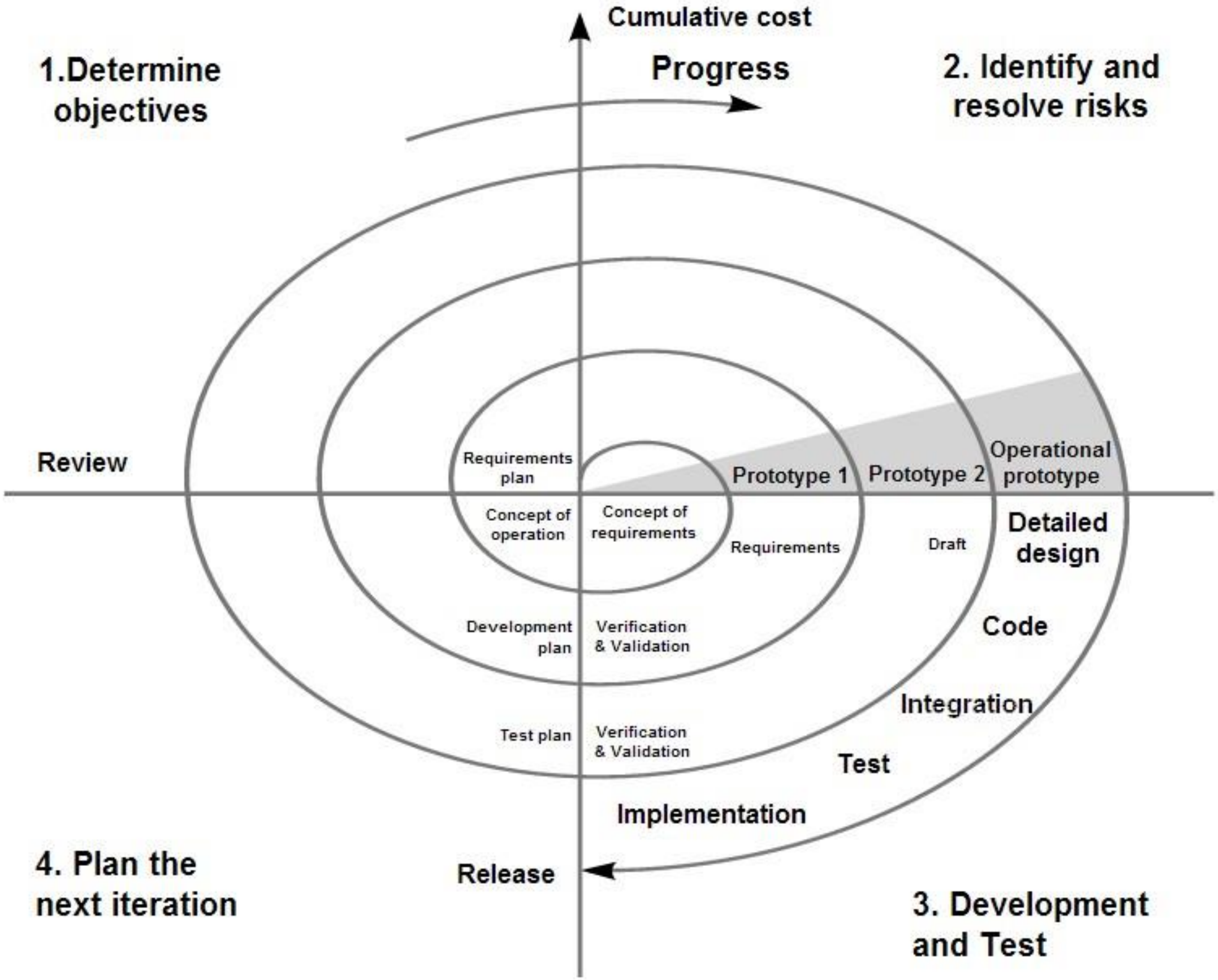


Main phases of development (usually starts with requirements; the subsequent iterations start with a requirements review and deciding what next to do.)

A spiral* model overview of development

* B. W. Boehm, "A spiral model of software development and enhancement," *Computer, vol. 21, pp. 61-72, 1988.*

Spiral model (Boehm, 1988)



More detailed classic model.

Each cycle begins with the identification of the <u>objective</u> of the portion of the product to be elaborated

When does it end?

Image from Wikipedia open commons

See discussion in Martinez, Bond and Vai, 2008, "High Performance Embedded Computing Handbook", CSC Press, pg 43.

HPES System Objectives

- Objectives of each cycle
  - specified in terms of the overall system into which the computing resource/processing is to be incorporated
- In early stages of the process
  - System objectives are usually more in terms of algorithms and processing needed (see next slide).

- Subsequent stages (after a few iterations)
  - More high-level design and experimental rapid prototyping of subsystems (e.g. writing a rough C routine version of a decided upon algorithm to evaluate its performance)
- Later stages (after a good understanding of processing needs and algorithms to use)
  - More focused on hardware subsystems fabrication
  - Software implementation
  - Testing

- Final stage(s)
  - Preparing and performing acceptance test(s)
  - Installation and post installation maintenance

System level goals

- Functional requirements
  - What the system should do
  - Operations to perform
  - Input → Output relations
  - Use cases (to be satisfied)
- Non-functional requirements
  - The ‘ilities’
    - Availability, Scalability, Reliability, Reusability, Maintainability
  - Performance
    - Speed of operation, throughput, response time (max. latencies)
  - Size, Weight, And Power (SWAP)

Goals of the Early Cycles

- In the early cycles, the goal is typically: reduce the largest technical risks and initiate lengthy tasks that may influence the overall (contractual) schedule.
  - If can't eliminate big risks need to reconsider continuing
- Some common risks addressed in early cycles of HPES development:
  - Form-factor constraints
  - Algorithm and functional uncertainty
  - Synchronization and control (exploiting regularity of data flow in computation)
  - Software complexity
  - Selecting COTS components
  - Custom ASIC design (high volume production only)

Costing and Risk management

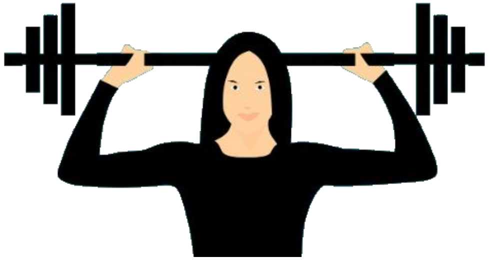
- Once the requirements, alternatives, and constraints are established, risk analysis is performed.
- Once development has progressed successfully, it should be feasible to retire certain risks at a Plan (stage 4) iteration
- At this point, management may review the cost of the previous cycle, scheduling of the next iteration and also revise the overall costing and development timeline.

Monitoring development progress & productivity

EEE4120F

Monitoring progress

- An important management tasks for HPES projects is developing accurate estimates for, and ways to measure development ‘PECS’:
  - Progress
  - Effort
  - Cost and
  - Schedule



- This may need to be tailored according to activities and types of technologies used during the project e.g.
  - MATLAB/Simulink coding vs. C coding vs. Assembly coding vs. FPGA HDL coding

Monitoring productivity/progress

- Need ways to measure both performance and progress for
  - Hardware development progress and
  - Software development progress

Hardware design progress

EEE4120F

Hardware progress

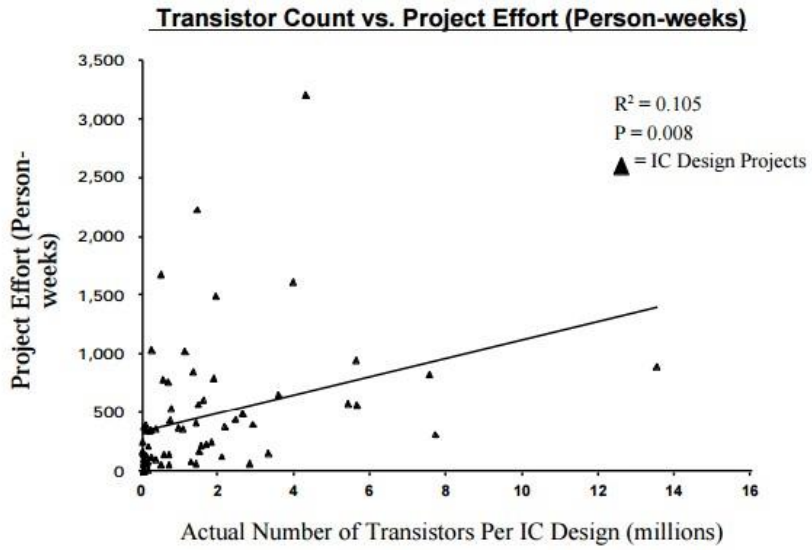
- A often mentioned method:
  - Determining the number of transistors, components and interconnects used in a design…

… surely that will give you a good impression of where the design is at?

Q Do you think this is a valid and fair approach?

Hardware progress

- Not necessarily
  - The graph on left shows number of transistors vs. weeks of design effort
  - As can be seen there may be little correlation in number of components in a design to amount of effort
  - Consider further there are usually cycles of design-test-optimization, so over time there may be increases and decreases in components used
- But agreeably it is likely the number of components will increase over long time periods when looking at one project
- It is better to consider the completion of functional units (or required functionality)



* Numetrics Management Systems (2000) “Measuring IC and ASIC Design Productivity”

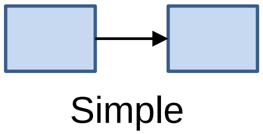
Hardware progress

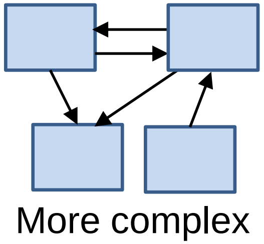
- Factors indicating progress
  - Requirements/Specifications provided
  - Number subsystems completed
  - Functionality completed (i.e. not looking necessarily in relation to specific requirements but more functions given, e.g. counter added to design)
  - System complexity (interconnects & modules)
  - Design size
  - IP usage
- Considered in relation to
  - Technology/tools used
  - Application domain
  - Frequency/speed of operation (e.g. high frequency, faster systems are more difficult to build)

Based on http://www.eetimes.com/document.asp?doc_id=1276032

Design Complexity Measures

- Two aspects of system design complexity*
  - Structural design complexity
    - Depth of hierarchies
    - Number of components
    - Number of connections
    - Interconnections between components
  - Functional design complexity
    - Number of functions provided
    - Complexity of the functions (e.g. symbols needed to describe the operation)
    - Sophistication of communication, handshaking protocols, flow control
    - Data management





* Dan Braha and Oded Maimon, 1998, “The Measurement of a Design Structural and Functional Complexity” In IEEE Transactions On Systems, Man, and Cybernetics

Cyclomatic Complexity

- **Cyclomatic complexity** is a software metric (measurement), used to indicate the complexity of a code section (or whole program). It is a quantitative measure of the number of linearly independent paths through a program's source code.

The approach is usually to think of (or visualize) the code section of interest as a graph, relating separable blocks of closely interdependent sequenced code (see next slide).

Good introductory starting point if you want to get more into this technique:
https://www.geeksforgeeks.org/cyclomatic-complexity/ (recommended tut)
more in-depth: https://dev.to/designpuddle/coding-concepts---cyclomatic-complexity-3blk

Read more about it at: https://en.wikipedia.org/wiki/Cyclomatic_complexity

McCabe's Cyclomatic Complexity Metric

Considers design as a graph representation.

Uses the formula of "cyclomatic complexity", which is as follows:

$$M = V(G) = e - n + 2p$$

(2p as connections between parts potentially more difficult to manage and design around that individual parts)

where:

V(G) = cyclomatic number of Graph G

e = number of edges

n = number of nodes

p = number of separate *connected components** of the graph (or system**)

*Connected components can be considered blocks of the code that are separated from the main sequence (e.g. the 'then' code that runs of when an if condition is true, an if causes a 1-part component separation).

** The graph represents the system being designed, although it could be extended to the 'development' system, i.e. people working on different parts using possibly different tools.

McCabe, 1976

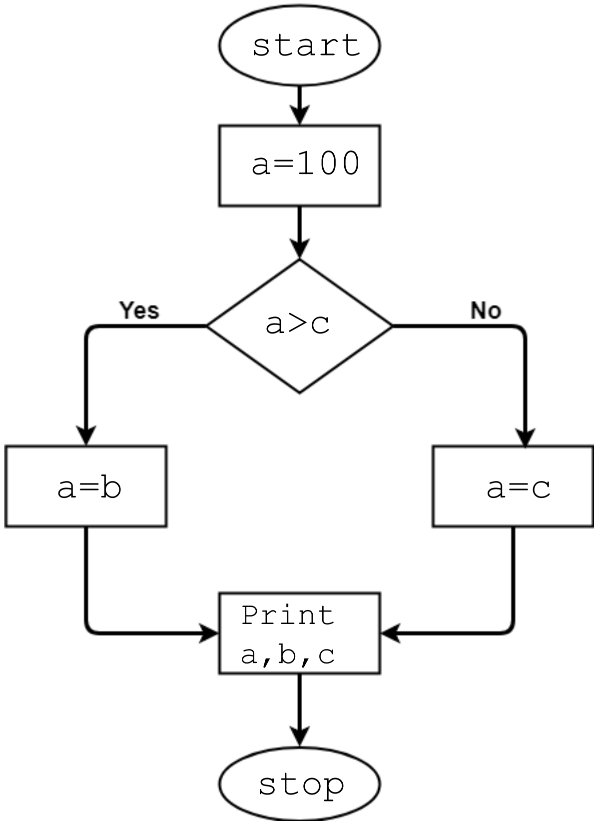
Cyclomatic Complexity Metric

- Need to find the linearly independent paths in the code section
- Usually done using the control flow graph of the program… essentially separating the part before the IF, and the two options after the IF (which might be doing an operation, or not doing the operation if there's no ELSE).
- See example on next slide

Cyclomatic Complexity Metric – Example 1

- Example block of code to consider:

```
void main ()
{
 int a = 100;
 if (a > c)
   a = b;
  else
   a = c;
 printf("%d %d %d",a,b,c);
}
```



As can be seen in the flow graph (or a flow chart) there are:

n = 7 nodes (we're including start and stop, don't have to),

e = 7 edges (including links to start and stop, don't have to).

p = 1 connected component (i.e. The if splitting the path into two options)

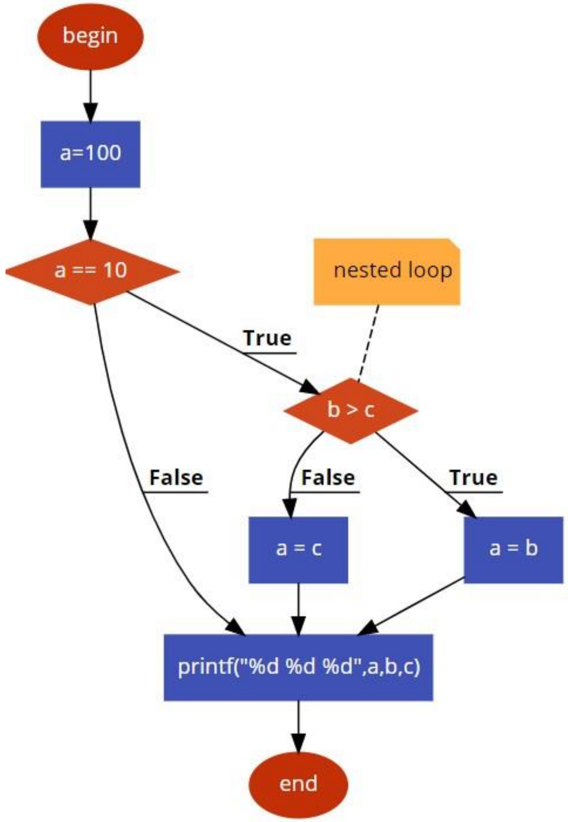
using M = e – n + 2p

M = 7 – 7 + 2(1) = 2

Cyclomatic Complexity Metric – Example 1

- Example block of code to consider:

```
if (a == 10){
 if (b > c)  // nested loop
   a = b
 else
   a = c
}
printf("%d %d %d",a,b,c);
```



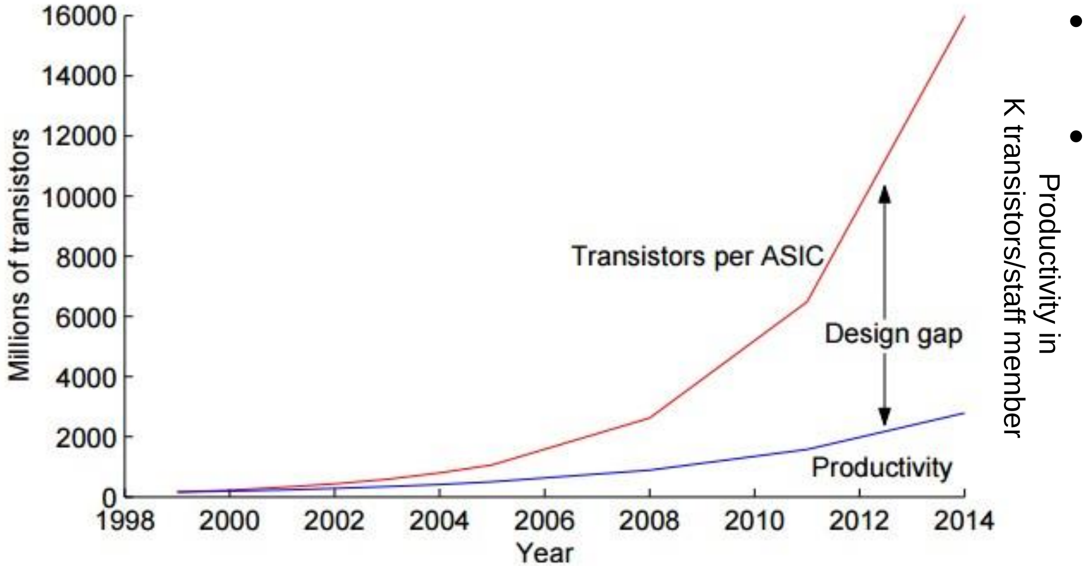
flow graph of code on left

As can be seen in the flow graph there are:
n = 8 nodes (we're including start and stop, don't have to),
e = 9 edges (including links to start and stop, don't have to).
p = 1 connected component
using $M = e – n + 2p$
$M = 9 – 8 + 2(1) = 3$

Flow chart developed using https://app.code2flow.com/

Design Productivity Gap

- Design productivity gap:
  - The difference between the transistors (resources) available in a single semiconductor die and the ability for the transistors to be used effectively in a design



- 1980s leading chip needing 100 transistors/month *
- 2002 leading chip needing 30,000 transistors/month *

Can (to some extent) substitute vertical axis for complexity of the system

* Vahid, Frank, and Tony Givargis. *Embedded system design: a unified hardware/software introduction*. Vol. 52. New York: Wiley, 2002.

Software design progress

EEE4120F

Monitoring progress of software

- Usual (easy) approach:
  - Measuring Lines Of Code (LOC) is one way
- SLOC: a possible improvement (mentioned in seminar 2)
  - SLOC = non-blank, non-comment source lines of code (SLOC) : some relation to the complexity of the code
- Potential inaccuracies and unfairness?
  - Well documented code is typically considered more valuable and reusable… but taking longer to get a solution could cause a product to fail.
  - … (next slide) …

Difficulty of monitoring progress based on LOC

- Potential inaccuracies and unfairness of using SLOC to measure progress / performance …
  - May have sudden needs for large blocks of code to be provided due to library limitations or incompatibility (e.g. textbook example, needing to fill in functions that were expected to be in the library)
  - Commented code is often better (and possibly easier to understand and share) than uncommented code.
  - Some difficult problems may have a short but non-obvious answer (e.g. coding a FIR filter)
  - Some easy problems may have a long and obvious answer (e.g. GUI code)
  - Some development tasks end in dead ends not contributing to the final design
    - Code/design may be thrown away
  - Some development tasks might be the result of a lot of learning (and the design team gaining skills) but having slower code production

Monitoring software progress

- Size of application
- Function points: measuring the *functionality* offered by a system
- Average number LOC between bugs
- Coupling (of classes, functions)
  - Measure of the strength of association between different entities
- Cohesion
  - Degree to which methods in a class (or functions in a module) are related to each other

Various other OO metrics can be considered, not included or examined in this course

Function Points (my preferred metric)

- Function Points gauge the functionality offered by a system
- A *function* can be defined as a collection of executable statements that performs a certain task
- Function points can be calculated before a system is developed
- They are language and developer independent (could apply to C / Java / Python / assembly / HDL)

Function Points

- A function point count is calculated as a weighted total of five major components that comprise an application, these are:
  - External Inputs
  - External Outputs
  - Logical Internal Files/modules
  - External Interface Files – *files accessed by the application but not maintained by it*
  - External Inquiries – *types of online inquiries supported*

Calculating a Function Point

- A simple way to calculate a function point count is as follows:

Function point count (or fpc) =

(Number of external inputs x 4) +

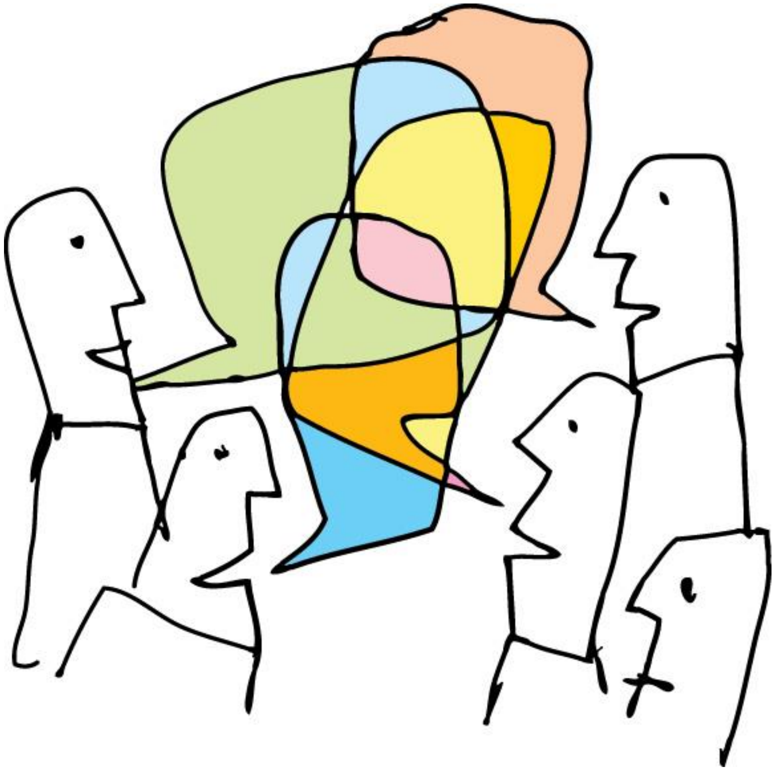
(Number of external outputs x 5) +

(Number of logical internal files x 10) +

(Number of external interface files x 7) +

(Number of external enquiries x 4)

These weightings are decided based on the degree of complexity of the development



Quick Class Activity:

Function Points Calculation

Function Point Example

See handout

Build a system that allows customers to submit product ratings of company X. These ratings will be stored in a file and company X staff will receive daily updates with new ratings. Customers can subscribe to weekly updates of product ratings that were submitted. Management can query the system for a summary of product ratings for a particular period.

Fill in a count for each of these items

External Inputs

Logical Internal Files

External Outputs

External Enquiries

Quick Class Activity:

Function Points Calculation

Function Point Example

Build a system that allows customers to submit product ratings of company X. These ratings will be stored in a file and company X staff will receive daily updates with new ratings. Customers can subscribe to weekly updates of product ratings that were submitted. Management can query the system for a summary of product ratings for a particular period.

1	External Inputs	1	Logical Internal Files
2	External Outputs	1	External Enquiries

Functional Points Example (cont.)

Functional Point Count calculation:

- External Inputs: 1
- External Outputs: 2
- Logical Internal Files: 1
- External Interface Files: 0
- External Enquiries: 1

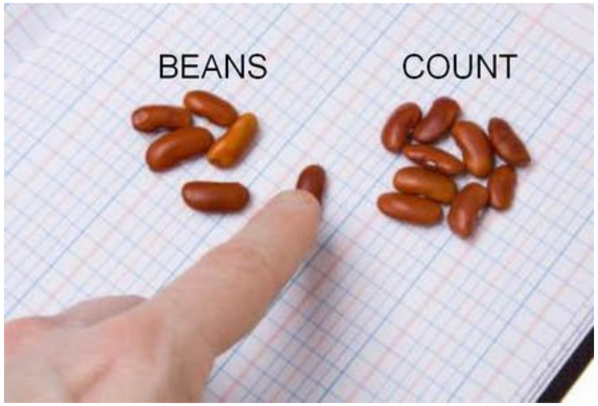
∴ *Function Point Count* = (1x4) + (2x5) + (1x10) + (0x7) +(1x4) = 28

Future Productivity Measurement

For individual developers or teams:

- Cost per Function Point
- Mean Time required to develop a Function Point
- Defects produced per hour
- Defects produced per function point

This is probably not used much in industry *at present*, but things are moving towards this direction. In my view it seems draconian and doesn't allow for how varied development work, especially embedded systems development, can be. Maybe for more straightforward programming (e.g. simpler web service development) it could be applicable. Basic moral of the story: don't tell the managers because they might just like this, and *we* likely would not!



Intermission

Eric Taylor Music - royalty free tracks
https://www.youtube.com/watch?v=gdJIE9lzNIQ

Spiral of Project Management

Functional Point Extensions

- The original Functional Points (shown in previous slides) are adequate for many applications
- However these have been extended for specialized domains (e.g. embedded systems) where the weightings need adjustment due to the nature and complexity of the applications developed.

Effort, Productivity and Progress

- Development effort, productivity and progress are not all the same thing
- **Effort** = amount of time involved (person hours; this is a simplistic view of effort)
- **Productivity** = rate of progress (high productivity → progress happening quickly)
- **Progress** = extent to which the desired objectives are complete (measured usually in terms of functionality provided and requirements satisfied)

Issues of Effort & Productivity

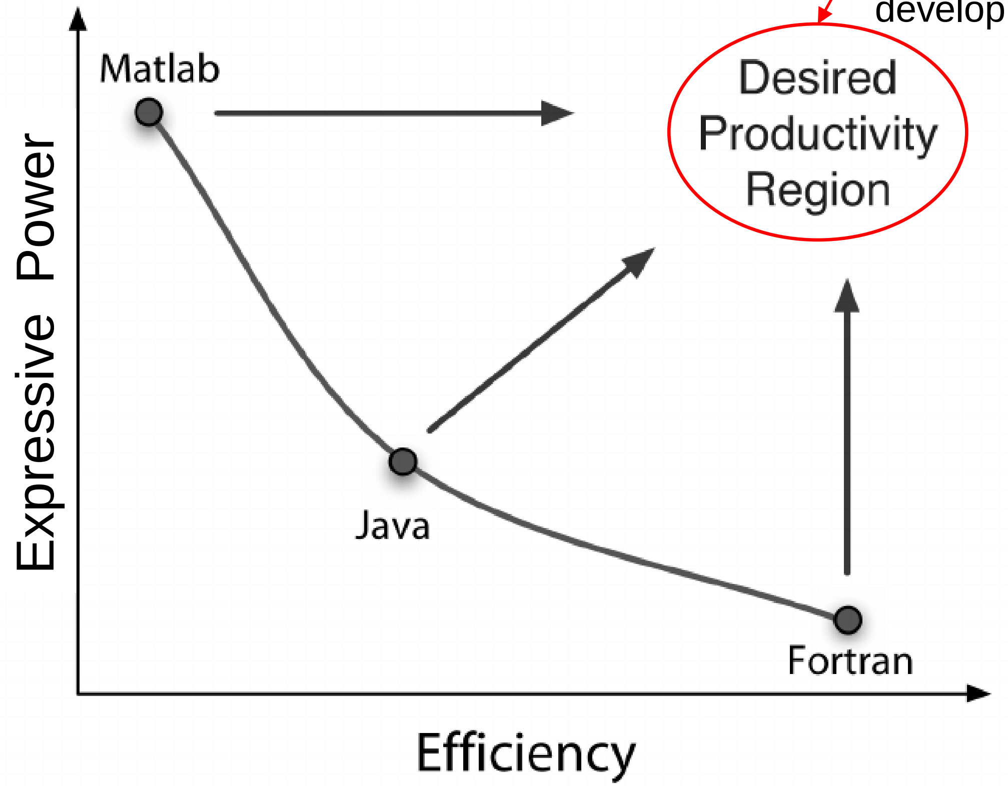
- Some tasks need more effort than others to gain a desired level of productivity
- Tools, programming language, prior knowledge, learning aptitude (among many other factors) can all clearly impact this significantly
- The expressive power of a language can influence the productivity achieve by using that language…

Expressive Power vs Developer Efficiency

- Expressive Power = ability of a language to provide advanced primitives and constructs to reduce the amount of effort required to program a solution

Might be the "silver bullet" of software development

Often there is a tradeoff between the expressive power of a language and its efficiency. For example according to the study by Kennedy et al. they demonstrated how certain commonly used languages can have noticeable tradeoffs between the expressive power



Kennedy, K., Koelbel, C., Schreiber, R., Kennedy, K., Koelbel, C., and Schreiber, R. Defining and measuring the productivity of programming languages. The International Journal of High Performance Computing Applications, (18)4, Winter 2004 (2004), 441–448.

Mythical man month

- The Mythical Man-Month * also known as “Brooks's law”:
  - Central theme is adding manpower to a late project makes it even later…
- The second-system effect *:
  - The tendency of small, elegant, and successful systems to be plagued with feature creep due to inflated expectations.

* Brooks, Jr., Frederick P. (December 2006) [1975]. "The Second-System Effect". *The Mythical Man-Month: essays on software engineering* (Anniversary ed.). Addison Wesley Longman. p. 53. ISBN 0-201-83595-9.

Documentation

- Documentation is important for the reuse and maintainability of designs
- A major barrier to reuse is lacking or poor documentation (the web is full of useful code libraries suffering from this)
- Automated documentation generation tools are a means to save time and improve the accuracy of design documents, such as use of **Doxygen**

Processes and trends

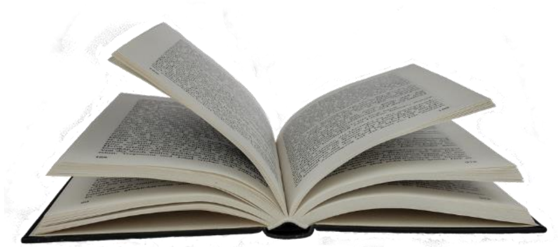
- Please ready through the rest of CH4 on your own. We've already seen much of what is said there, and experienced simplified instances of the development issues in pracs.

The slides that follow is a brief discussion of automated documentation generation using the Doxygen tool, it is optional reading and can be skipped for test purposes

Doxygen

EEE4120F

These slides are aimed more at additional reading and for application to Prac4 in which a brief intro to Doxygen is given.



Doxygen – code documentation tool

- Doxygen is a highly recommended tool for generating code documentation from comments in the code.
- It is a documentation system for C, C++, Java, among other programming languages.
- It helps to
  - Generate on-line or offline reference manuals from commented source files.
  - Extracting the code structure and visualising relations between software components using dependency graphs, and various UML modelling techniques such as inheritance diagrams, and collaboration diagrams that are generated automatically.
  - NOTE: This doesn't mean to say you can skip the software design phase of development but it can help synchronize what your implementation becomes with its design visualization

Where to get Doxygen

- Doxygen website
  - http://www.stack.nl/~dimitri/doxygen/

Using Doxygen

- Initial setup
  - Step 1: Create a Configuration File
  - Doxywizard is a GUI program for creating the config file
  - Construct templates (to copy and paste to save typing)
- Following cycle repeats:
  - Step 2: Document the Code
  - Step 3: Run the Doxygen
- Don’t usually run doxygen for each compile in code-compile-test cycle as it can take a while to complete.

How to use Doxygen

- Techniques for documenting
  - Code blocks or lines
  - Functions / member functions
  - Classes and structures
  - Class attributes
  - Code structures (e.g. for loop, if then else)

How to use Doxygen

- Doxygen comments start with a * or !
  - Examples:

```
/** description of function. */
/*! Another description */
//!  Another Doxygen comment
///  Also Doxygen comment with 3 x '/'
```

Documenting code

```
int var1; //! Document member or variable

/*** Document function,
     at top of declaration */
void myfunc (int a, int b)
{
}
```

Doxygen formatting

- Bulleted lists
  - Unnumbered:
    Use a column aligned minus sign –
  - Numbered
    Use a column aligned minus sign – followed by a #   (i.e. -# blurb)
  - Nested lists: indent the – or -#
- Arbitrary HTML code can be added
  - HTML commands (e.g. blurb) can be used inside comment blocks

Doxygen list example

```
/**
* List of items
*    -Top level issue A
*       -#Sub issue one
*       -#Sub issue two \n
*         another line for issue two.
*       -#Sub issue 3
*    -Top level issue B
          -#Sub issue one of B
*    -Top level issue C
*/
```

List of items

- Top level issue A
    1. Sub issue one
    2. Sub issue two
       another line for issue two.
    3. Sub issue 3
- Top level issue B
    1. Sub issue one of B
- Top level issue C

Doxygen

- These slides were meant as a brief into, for more details on Doxygen commands and syntax please see the Doxygen online manual

http://doxygen.nl/manual.html

General causes making a…

Common causes of failure

You? *Poorly Planned Project?*

- Requirements Analysis
  - Nothing recorded / no written requirements
  - Requirements vague or insufficiently described
  - Leaving it ‘till too late to actually formalize requirements
  - No directions on user interface
  - No end-user involvement (occasionally difficult to organize)
- Design
  - Insufficient design and planning done
  - No documents (or poorly formed)
  - Inefficient data structures / file formats
  - Infrequent or no design reviews
  - Lack of consultation/input from experts and senior engineering staff

Common causes of failure

- Implementation
  - Lack of, or insufficient coding standards (incl. inconsistent coding style etc.)
  - Infrequent or no code reviews
  - Poor in-line code documentation
- Subsystem/component testing & Integration
  - Insufficient component testing
  - Incomplete testing or running ineffective tests
  - No quality assurance

Most common cause of success??

- How can we avoid making the mistakes that lead to project failure? Besides the obvious point of having competent staff?
- Apparently* the answer is simply:
  - By using "simple common sense… which is often ignored in systems projects."*
- Need the three pillars of success:
  - A sound methodology
  - Solid technical leadership by someone who's successfully done a similar project
  - Management support

(a likely question or bonus question 😉)

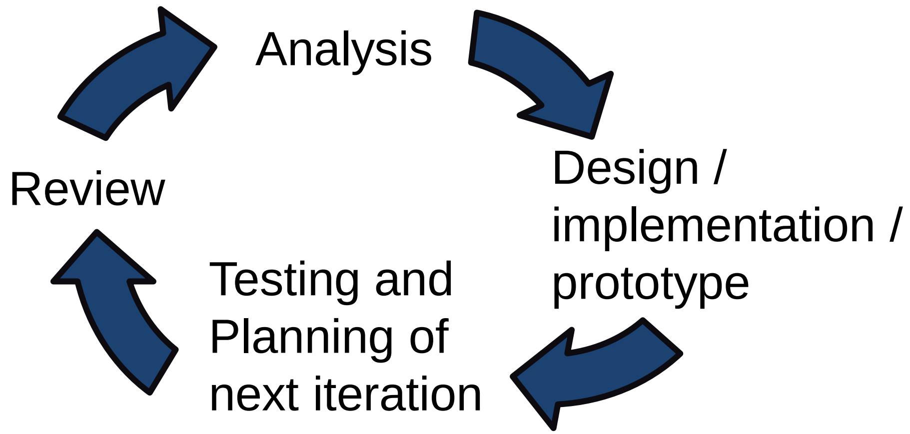
* M. I. Sanchez-Segura, J. García, A. Amescua, F. Medina-Dominguez, and A. Mora-Soto, "A Study on How Software Engineering Supports Projects Management," *Innovative Techniques in Instruction Technology, E-learning, E-assessment, and Education, pp. 161-165, 2008.*

Class Activity on Development Process

EEE4120F

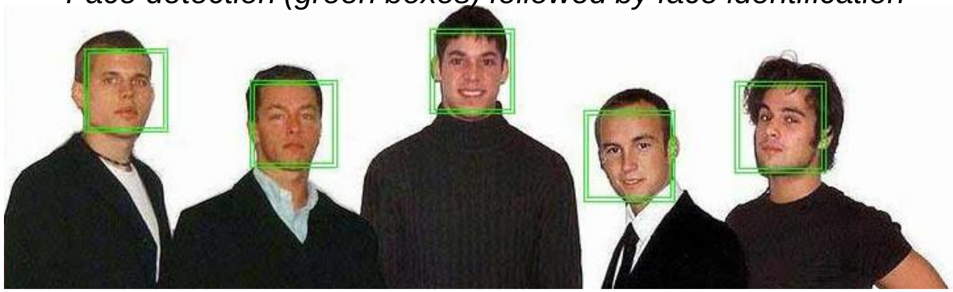
Reflections – short activity

- Keep in mind the main recurring phases of the spiral model:



Consider that you are embarking on a project that involves developing a face recognition system for *The Hawks* *. The system is accessed via possibly (very low budget) workstation PCs, which are used to upload photos to a remote central computing site where the face recognition functions are run. The central computing site comprises a fast PC with one or more digital accelerator to do the main number crunching. (to next slide..)

Face detection (green boxes) followed by face identification



* The Hawks, officially called the ‘Police's Directorate for Priority Crime Investigation’, is the South African current take on the US’s version of the FBI.

Image source – Wikipedia open commons

Todo – group task

- Form into groups to discussion:
  - How would the first step of the spiral model be carried out for the face recognition system
  - What are some of the risks to content with for the first thing to carry out
  - What would you do to test and analyse the results (if applicable)
  - What would the next cycle involve?

Any Questions?

End of Slideshow

Disclaimers and copyright/licensing details

I have tried to follow the correct practices concerning copyright and licensing of material, particularly image sources that have been used in this presentation. I have put much effort into trying to make this material open access so that it can be of benefit to others in their teaching and learning practice. Any mistakes or omissions with regards to these issues I will correct when notified. To the best of my understanding the material in these slides can be shared according to the Creative Commons “Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)” license, and that is why I selected that license to apply to this presentation (it’s not because I particulate want my slides referenced but more to acknowledge the sources and generosity of others who have provided free material such as the images I have used).

Image sources:

Face identification image; crowd scene – Wikipedia open commons

Group task image – OpenClipart.org

Human thinking, man clicking - Pixabay

Sunset – flickr open commons

Composite project success image – sources include flickr and openclipart.org