Project Management

Lecture 5

Ruzanna Chitchyan, Jon Bird, Pete Bennett
TAs: Alex Elwood, Alex Cockrean, Casper Wang

(Using materials created by N. Walkinshaw and R. Craggs)

Overview

- About Measurement
- Measurement under White Box:
  - Lines of code
  - Cyclomatic Complexity
- Measurement under Black Box:
  - Planning Poker
- Software Laws:
  - Patents, Copyright, Contract, Privacy

Measurement is Central to Quality

- How to plan for the project time and effort?
  - For the team?
  - For the customer?
- Which software/part of it needs more time for testing?
- Which developer should get a bonus payment for productivity?....

"You cannot **control** what you cannot measure."

Tom DeMarco, 1982

What is “Measurement”?

- Attributing values to objects.
  - The fuel efficiency of a car (gallons per mile)
  - The number of goals scored by a footballer
  - The cost of a house
- Can use these values as basis for comparison
  - What is the cheapest house?
  - Who is the best goal scorer?
- Can use these measurements and comparisons to **make better decisions**.
  - Which car should I buy (e.g., given five candidate cars)
  - Which striker should I put in my team?

Measurement is Difficult in Software Engineering

- Most entities are difficult to measure reliably
- Difficult or impossible to “pin down” a single value

E.g., Software Quality (ISO/IEC 25010):

- Functional Suitability
  - Functional Completeness
  - Functional Correctness
  - Functional Appropriateness
- Performance Efficiency
  - Time Behaviour
  - Resource Utilisation
  - Capacity
- Compatibility
  - Co-existence
  - Interoperability
- Usability
  - Appropriateness
  - Realisability
  - Learnability
  - Operability
  - User Error Protection
  - User Interface Aesthetics
  - Accessibility
- Reliability
  - Maturity
  - Availability
  - Fault Tolerance
  - Recoverability
- Security
  - Confidentiality
  - Integrity
  - Non-repudiation
  - Authenticity
  - Accountability
- Maintainability
  - Modularity
  - Reusability
  - Analysability
  - Modifiability
  - Testability
- Portability
  - Adaptability
  - Installability
  - Replaceability

Usual Metrics: Size and Complexity

- After development ...
  - How much effort will it require for maintenance?
  - Where should we direct testing effort?
  - How much effort was required for development?
  - Metrics are based upon source code (“white box”)

- Before development has started ...
  - How much programming effort will module X require?
  - What will be the estimated cost of the final product?
  - Metrics are based upon requirements / specification (“black box”)

White Box Complexity Metrics

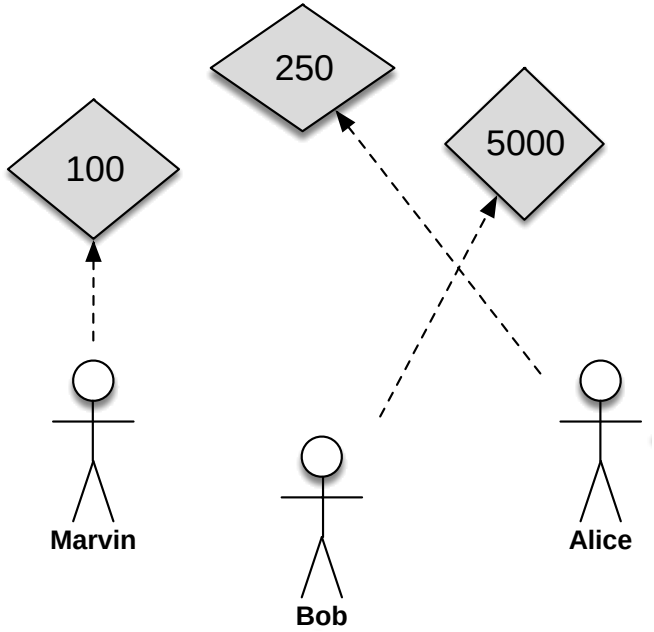
Number of lines in a file (or a group of files)

- Easy to compute
- Easy to understand and interpret
- Often sufficient for an approximate measure of size
- Widely used (perhaps the most widely used) metric

- Comments
- What is a line?
- Blank lines
- Not all “lines” are equal
- Ignores logical/ architectural complexity
- Highly language-specific

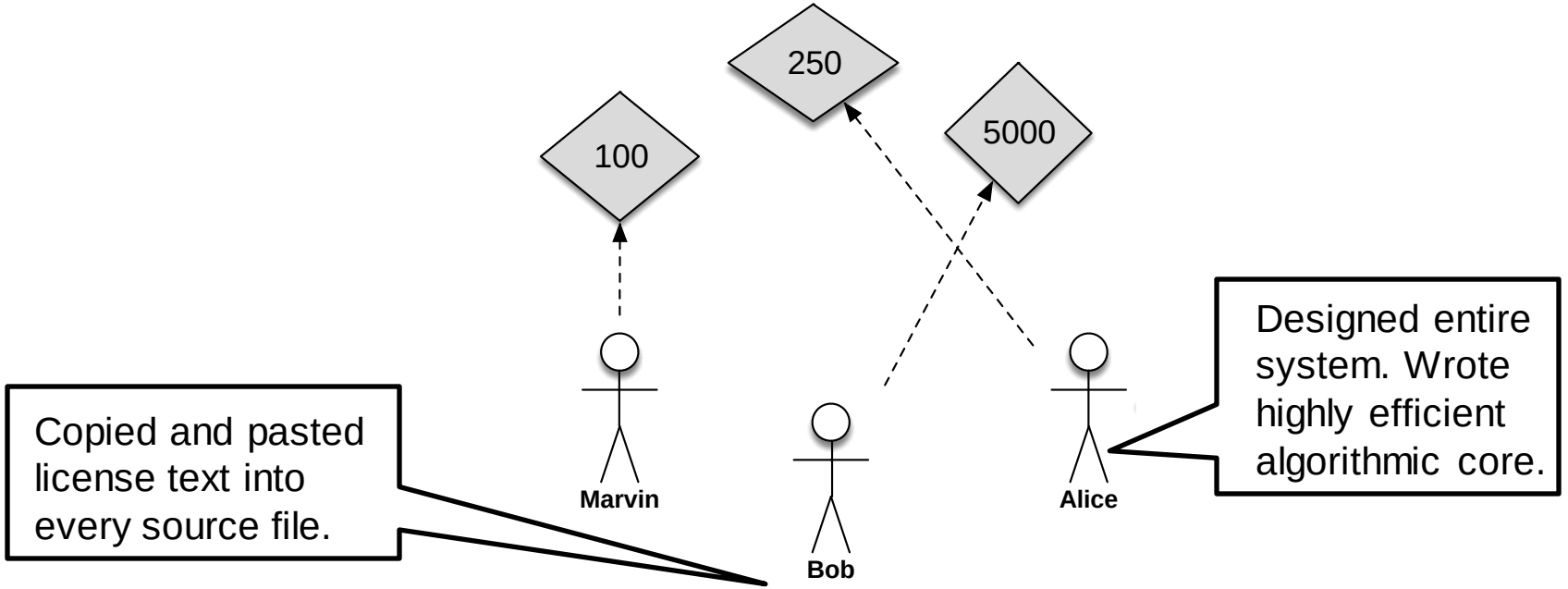
Example: Who is the most productive programmer?

Measured in lines of code



Example: Who is the most productive programmer?

Measured in lines of code



Cyclomatic Complexity

- Calculated from the control flow graph:

  V(G) = E – N + 2P

  E – number of edges;

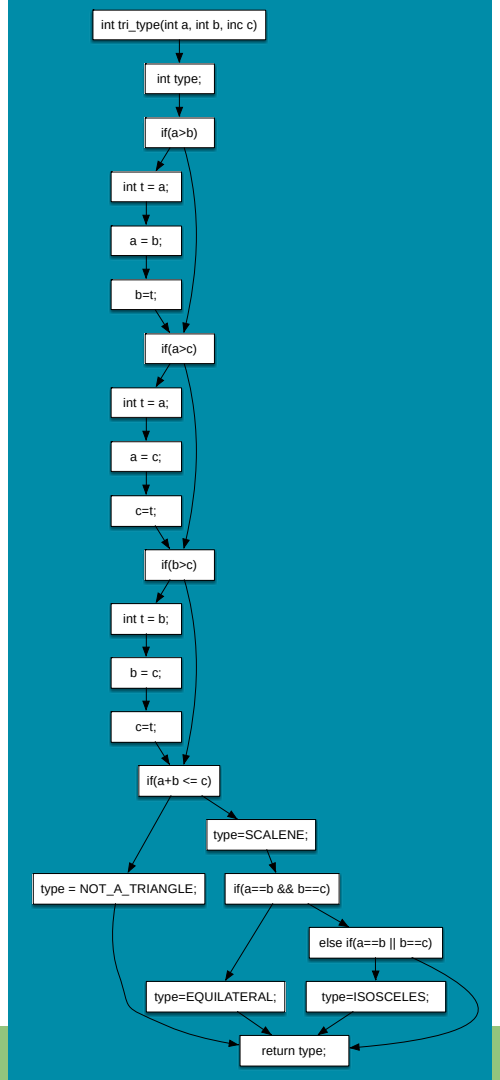
  N – number of nodes;

  P – number of procedures (usually 1)

- Number of independent paths through the code
- Independent path – any path that introduces at least one new statement/condition

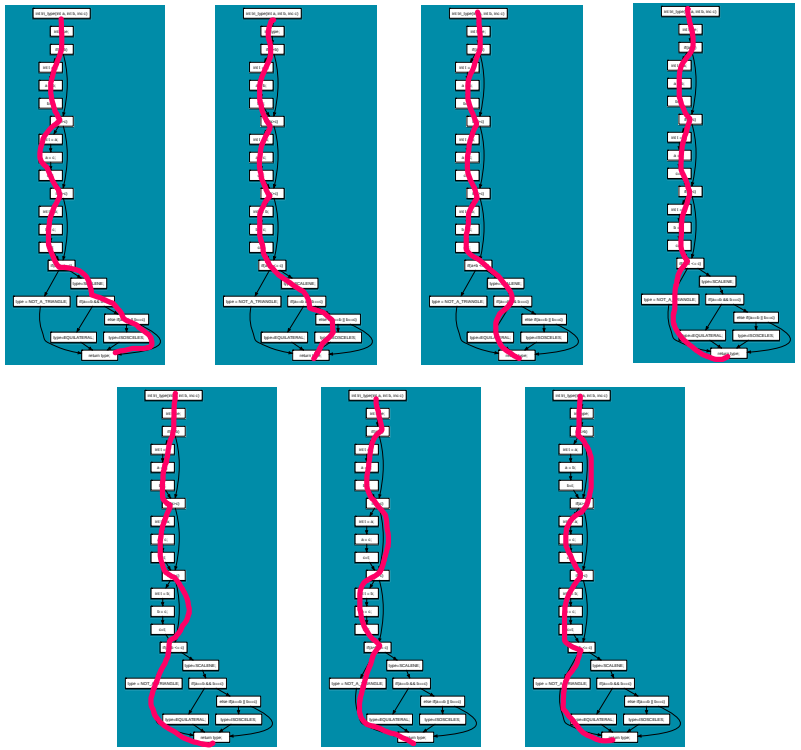
Triangle Example

```
1   int tri_type(int a, int b, int c) {
2     int type;
3     if (a > b)
4       { int t = a; a = b; b = t; }
5     if (a > c)
6       { int t = a; a = c; c = t; }
7     if (b > c)
8       { int t = b; b = c; c = t; }
9     if (a + b <= c)
10      type = NOT_A_TRIANGLE;
11    else {
12      type = SCALENE;
13      if (a == b && b == c)
14        type = EQUILATERAL;
15      else if (a == b || b == c)
16        type = ISOSCELES;
17    }
18  return type;
19  }
```



Number of Edges = 27
Number of Nodes = 22

V = 27 – 22 + 2 = 7



Black Box Complexity Merics

Estimating Agile Projects

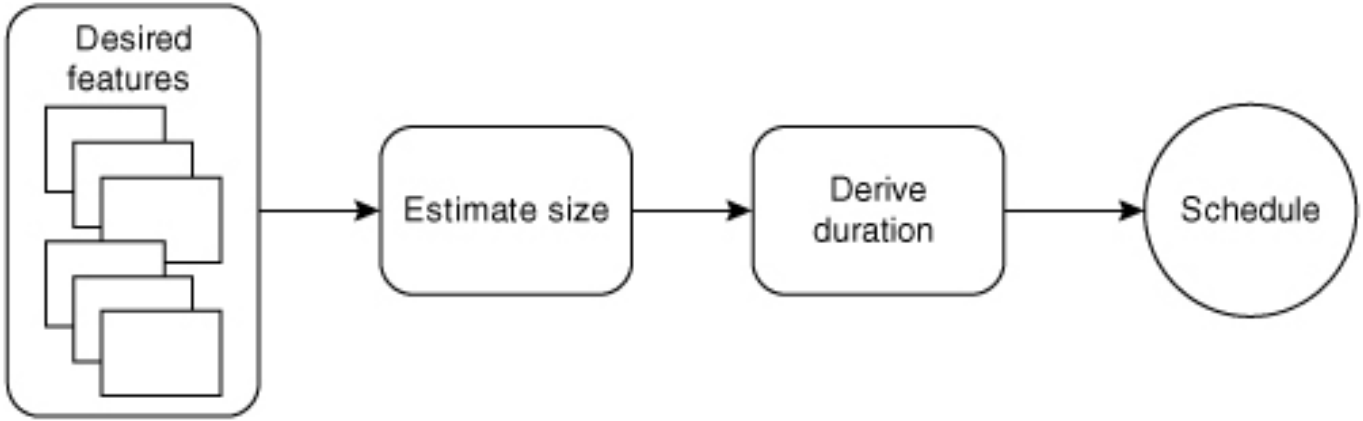


Figure from: Agile Estimating and Planning by Mike Cohn

Storey Points (Size Estimation)

- An informal, agile unit of “size measurement”
    - Usually an estimate from 1-10
- Derive an estimate from the whole team at sprint planning meetings
- Based on the idea of the “Wisdom of the Crowds”
    - The collective estimate of groups (i.e., of effort required for a story) is better than the estimate of an individual

Accuracy vs Effort in Project Estimation

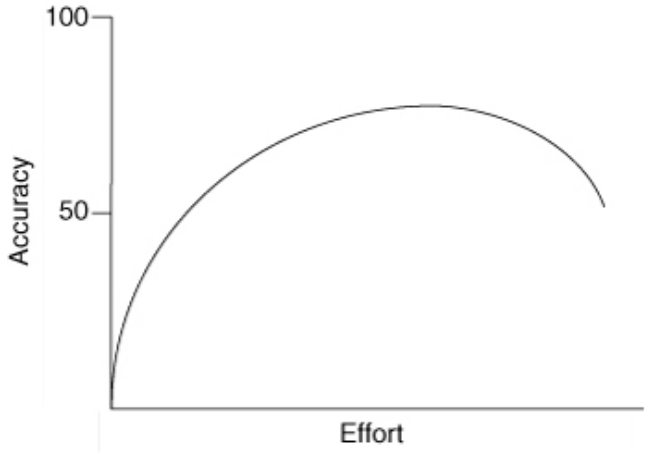
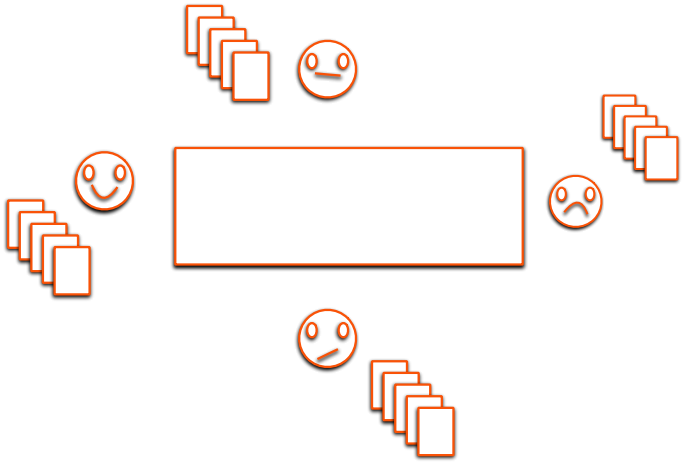


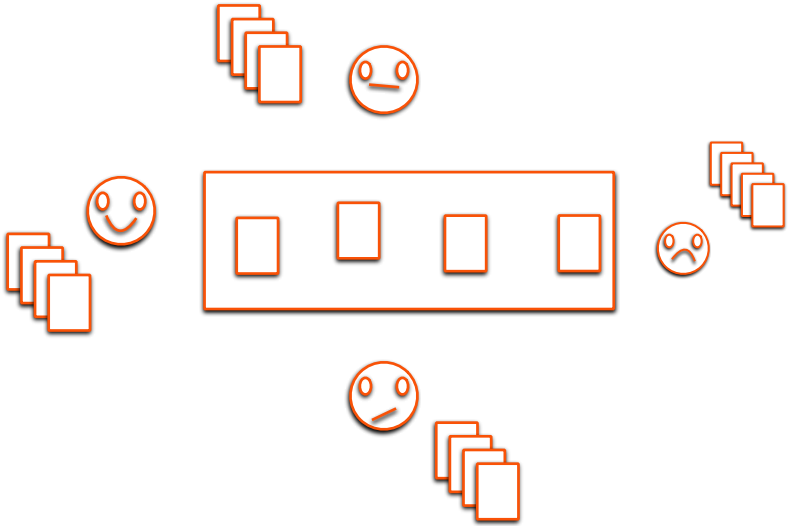
Figure from: Agile Estimating and Planning by Mike Cohn

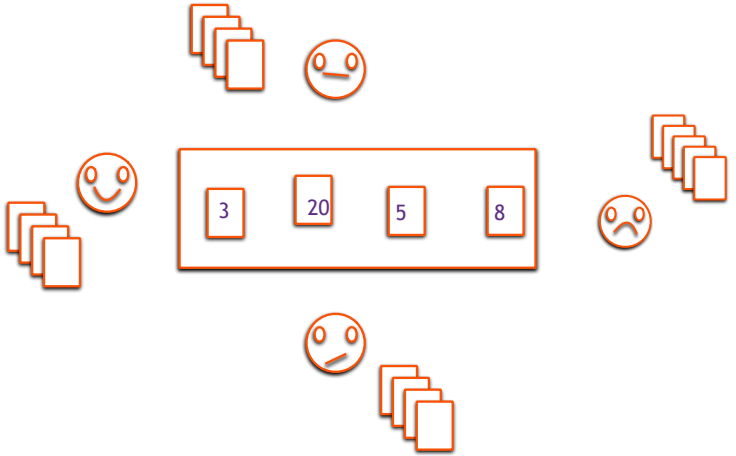
Planning Poker

- The whole team is involved
- Each member is given a set of numbered cards
- Numbers follow the Fibonacci sequence
- 1,3,5,8,13,20,...
  - Larger tasks become harder to estimate in exact terms
  - Low values - trivial to implement
  - High values - difficult to implement
- Each member is also given a “?” card

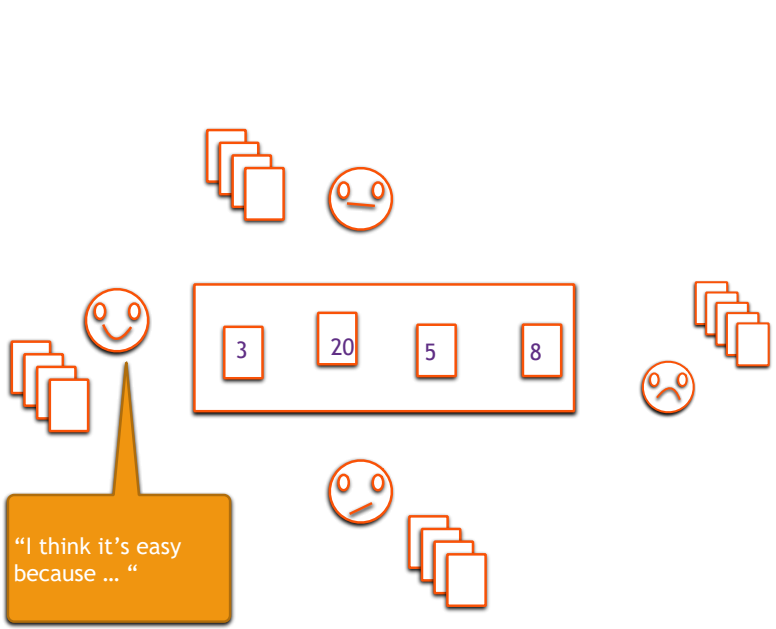


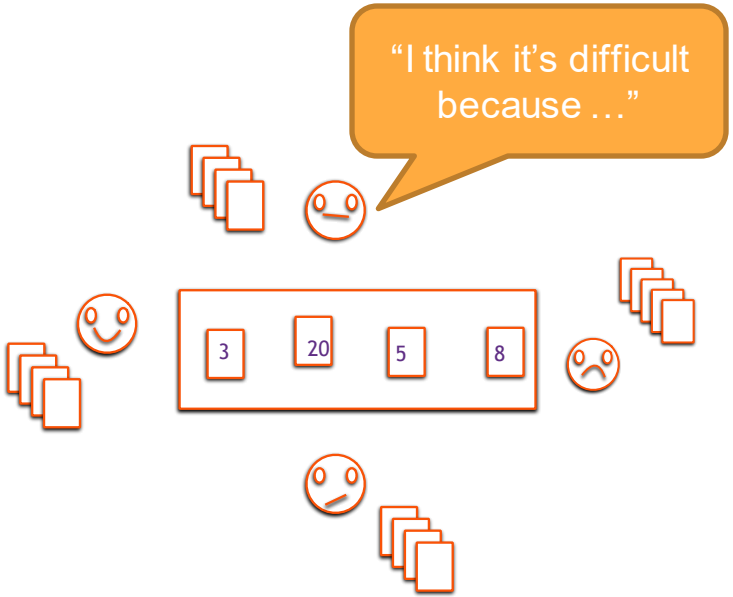
Planning Poker: Process





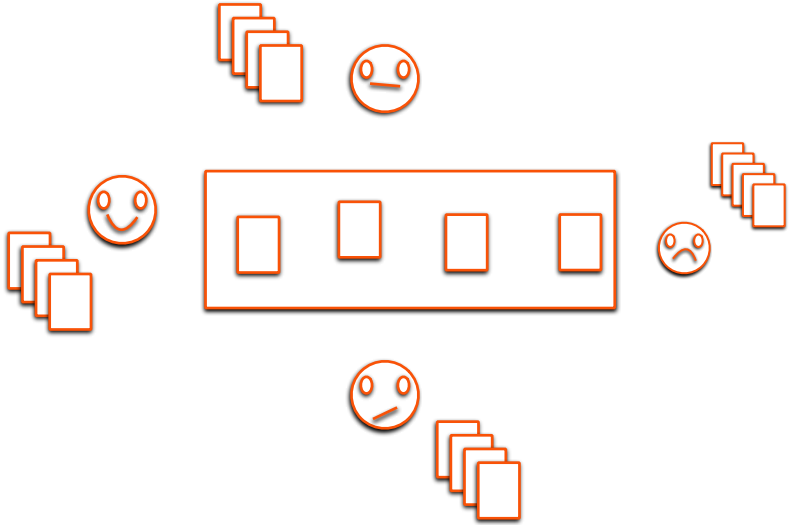
Planning Poker: Process

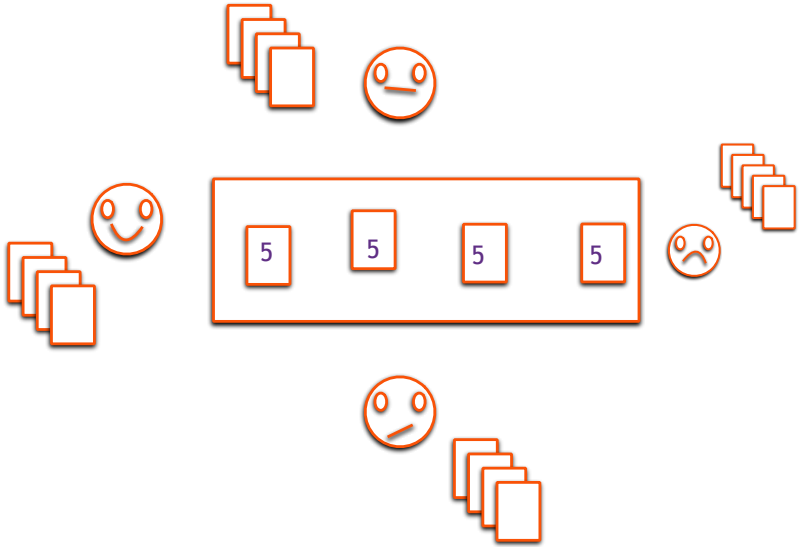




Planning Poker: Process

Cycle repeats for a maximum of 3 iterations (to avoid infinite loops!)

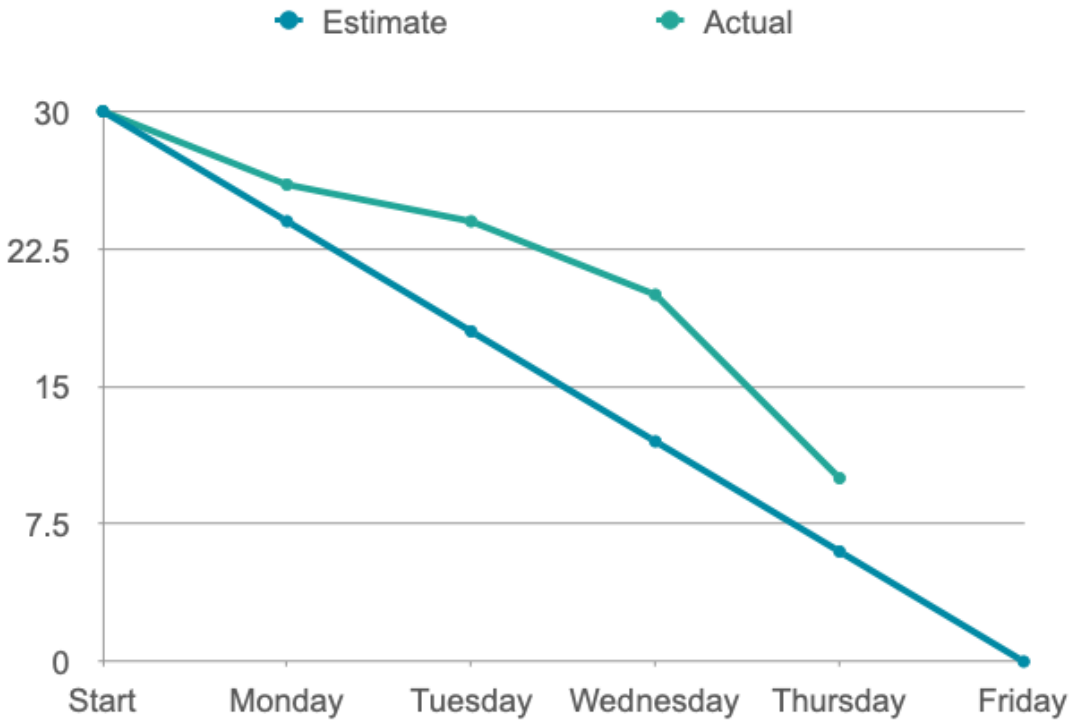




Team Velocity

- Number of (estimated) story points implemented per sprint.
- Can be derived from previous sprints.
    - e.g., Average points implemented from previous *x* sprints.
- Can be used to estimate:
    - Time required to complete project.
    - Target number of stories that can be completed in a sprint.

Burn Charts


Estimate
Actual
30
22.5
15
7.5
0
Start
Monday
Tuesday
Wednesday
Thursday
Friday

Software Laws: Patents, Copyright, Contract, Privacy

Patent Law

A government license giving a right for a set period, especially to exclude others from making, using, or selling an invention

- Granted by the government
- to stop others exploiting **your** invention
- Lasts 20 Year

Inventions Must

- be **new**
- be an **inventive** step (not an obvious improvement)
- capable of industrial application

The “Social Network”

Did Mark Zuckerberg infringe a patent?

- No patent was granted
- The idea was not new, social networks existed before this

Copyright

- Creator has **exclusive rights** to perform, copy, adapt their work.
- Everyone else must get **Permission** (and possibly pay)
- "literary, dramatic, musical and artistic works" **includes software**
- Automatically owned (not granted)
- Lasts **70 years** after authors death (lots of exceptions)

This affects software in 2 different ways:

- Illegal Copies of Applications (Piracy) !
- Using someone else's code/UI design/etc. in your application

(Not the "idea" but the actual "stuff" (code, design, documents) created by someone else)

Copyright Theft?

NO:

- Get permission (obtain a licence)
- Be within "fair use" (e.g. for study or review)
- Use "open source" software
- Create something similar yourself, independently
- "Obvious" code can't be copywrited

YES:

- Displaying an image from another page
- Using code found on the internet
- Copying Windows 95 for your friends

The “Social Network”

Did Mark Zuckerberg infringe copyright?

Maybe

- but there is no evidence he copied
- it it's not fair use
- it wasn't OSS
- he saw the code so didn't invent it himself

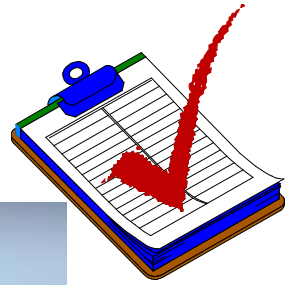
Contract Law

Employer contracts usually force an employee to:

- Not work for anyone else
- Hand over any ideas (Intellectual Property)
- Not disclose company secrets (Non-disclosure-agreements)

  (even after you stop working for them)

The “Social Network”



Did Mark Zuckerberg break contract?

Probably Not

- there was no written contract
- he did not disclose any secrets about the other project

Data Protection

- **UK** : Data Protection Act
- **EU** : Data Protection Directive
- **US** : a "patchwork" of state and national laws

8 Principles of Data Protection:

Any company storing "personal data" must make sure it is:

- fairly and lawfully processed (consent, contractual and legal obligations, public interest, ...)
- processed for **limited purposes**;
- adequate, relevant and **not excessive**;
- accurate and, where necessary, kept up to date;
- not kept longer than necessary;
- processed in accordance with the data subject's rights;
- secure;

not transferred to countries without adequate protection

Review

- How can we measure complexity?
- Why do we use black box options?
- What is a patent
- What is the difference between patent and copyright?
- What do we learn about contract from Social Network?

HCI Evaluation
Part One

Dr Jon Bird
jon.bird@bristol.ac.uk

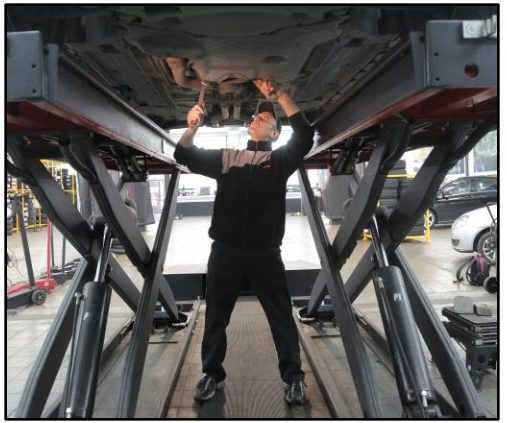
Thanks to Stuart Gray, Pete Bennett, Simon Lock, Thomas Bale, Harry Field who developed some of these slides

Today's Lecture

- What is HCI evaluation?
- Why is it important
- The Think Aloud evaluation technique
- Heuristic evaluation

HCI Evaluation

- Evaluation is a crucial part of the user-centred development process – we want to ensure our software meets our users' requirements
- The focus of this lecture is on Think Aloud technique and Heuristic Evaluation, which are two of the most widely used evaluation methods in industry
- They are methods that we recommend you carry out on your game as part of your group project – you can write up the results in your report



Why is evaluation important?

- *“Iterative design, with its repeating cycle of design and testing, is the only validated methodology in existence that will consistently produce successful results. If you don’t have user-testing as an integral part of your design process you are going to throw buckets of money down the drain.”*

Bruce Tognazzini (we’ll meet him later in the lecture)

The Think Aloud evaluation technique

- Users are asked to verbalise what they are thinking and doing as they perform a task using your software
- The Think Aloud technique provides insights into the user experience of using your software
- It can identify issues with the software e.g. navigation problems or content that can be improved
- It can be used as part of the software development process to iteratively improve software or used with a finished product

Benefits of Think Aloud

- Cheap
- Relatively easy
- It provides insight into people's experiences as they interact with your product
- It can be carried out with low numbers of participants
- Fits in with most software development processes

Drawbacks of Think Aloud

- it relies on people verbalising thoughts and impressions, rather than objective measures
- Participants may say what they believe to be the right answer rather than what they really think (social desirability). This can distort your results and conclusions

Planning a Think Aloud evaluation

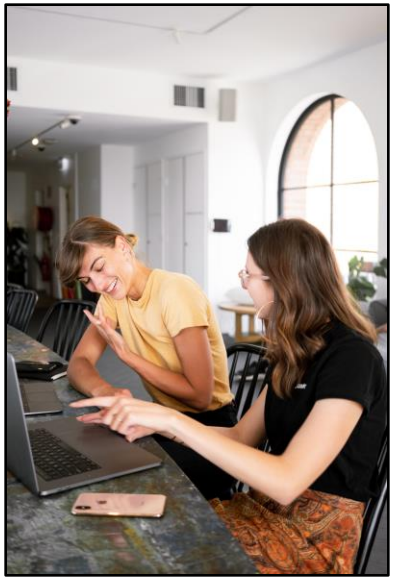
- Decide what questions you want your study to answer. For example, whether users can find particular content or what their understanding is of the information presented.
- Write down the tasks you want the user to complete while using your software
- Decide how many participants you want to recruit and how long you want the sessions to last (45 to 90 minutes works well)

Carrying out a Think Aloud evaluation 1

- Have a **facilitator** to run the evaluation and one or two observers to take notes on what the user says
- Explain to the participants how a think aloud works: they should tell you their thoughts, reactions and emotions as they occur while they are performing the task
- Explain that there is no right answer and it's fine to be critical

Carrying out a Think Aloud evaluation 2

- Ask the participants to complete the tasks you have planned. This should be **uninterrupted** as far as possible, although the facilitator will probably need to give some prompts.
- If the user goes silent then prompt them to verbalise their thoughts by saying “what are you thinking”



Analysing a Think Aloud evaluation

- Put the written notes together from both observes in to one document
- Organise the notes into meaningful categories e.g. what features helped users; what features led to problems; any additional features that users wanted.
- You can make your own meangingul categories
- Count the number of times users comment about different categories to identify the biggest issues

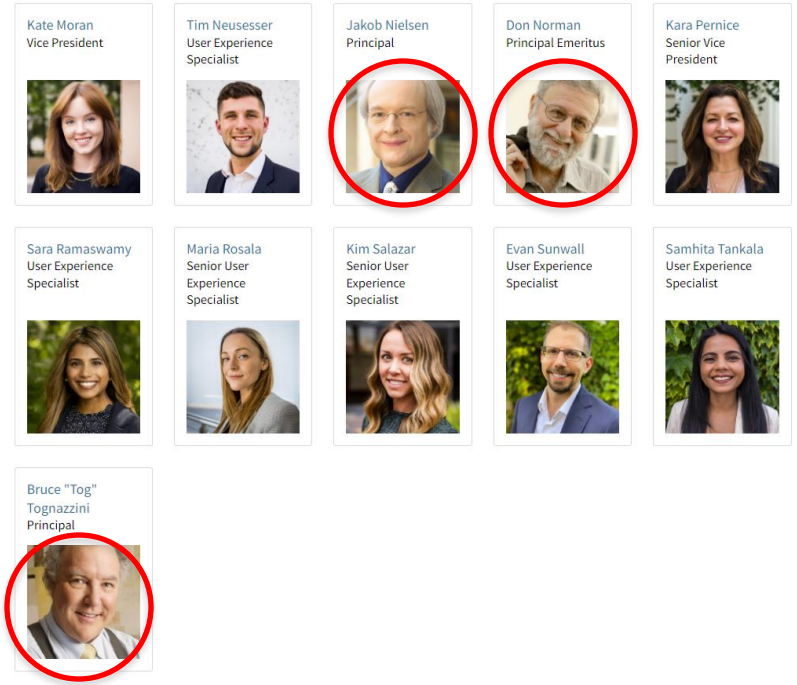
Jakob Nielsen – heuristic evaluation

Nielsen, J., and Molich, R. (1990). Heuristic evaluation of user interfaces, *CHI'90*, 249-256.

https://www.nngroup.com/articles/how-to-conduct-a-heuristic-evaluation/

Nielsen Norman group http://www.nngroup.com/

- The Nielsen Norman group is a UX research and consulting firm
- It was founded by two big figures in the HCI world:
  - Don Norman coined the term “user experience” and developed a set of design heuristics
  - Jakob Nielsen also developed a set of usability heuristics and was a pioneer of heuristic evaluation

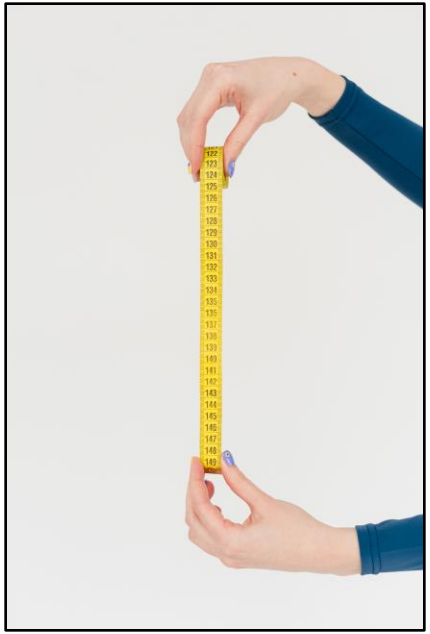


What is a heuristic?

- A rule of thumb
- Experienced-based strategies
- E.g. if you're doing some DIY then 'measure twice, cut once' is a useful heuristic

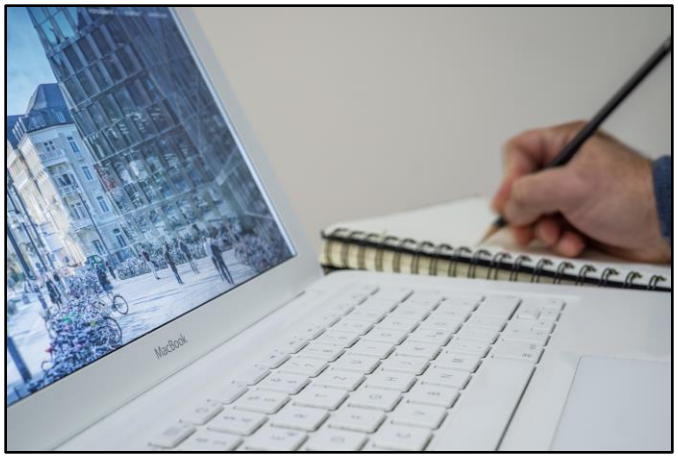
Heuristic evaluation 1

- An evaluation technique conducted **without** users
- Also known as **expert** evaluation as it's sometimes carried out by external experts (sometimes by the development team) aka evaluators
- It's a type of **analytical** evaluation, that is, based on a set of principles or a model…
- …rather than by observing users (which is known as **empirical** evaluation)



Heuristic evaluation 2

- It's an **inspection** method – it involves inspecting a design to find usability problems
- This involves asking whether the design complies with **usability principles** (a set of heuristics)



Heuristic evaluation is widely used because…

- It's **cheap** (only needs a small number of evaluators and no specialist equipment or labs)
- Relatively **easy** to carry out (can do it after a few hours of training)
- **Instant gratification** – lists of problems are **available immediately** after the inspection
- It **fits in** with most software development processes used in industry
- It's a very **cost effective**: benefit-cost ratio of 48: cost of $10,500; expected benefits $500,000 (Nielsen 1994).

Where are the users?

- Heuristic evaluation is based on HCI researchers' extensive experience of designing and evaluating interfaces
- By focusing on users, HCI researchers learned what works and what doesn't
- Their experience is distilled into **usability principles** (a set of heuristics)
- The principles represent the findings from thousands of user studies
- They have been used for over 30 years

What are Nielsen's 10 principles of heuristic evaluation?

- visibility of system status
- match between system and real world
- user control and freedom
- consistency and standards
- error prevention
- recognition rather than recall
- flexibility and efficiency of use
- aesthetic and minimalist design
- help users recognise, diagnose and recover from errors
- help and documentation

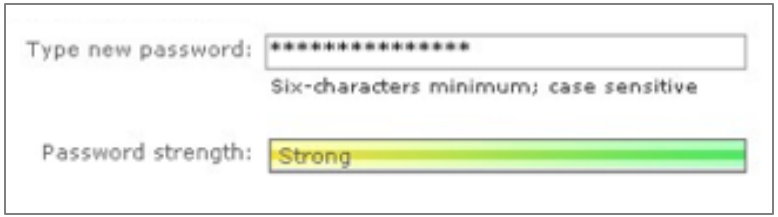
Nielsen's 10 principles of heuristic evaluation (minimal information)

- feedback
- metaphor
- user control and freedom
- consistency
- error prevention
- recognition not recall
- flexible use
- minimal information
- error recognition and recovery
- help

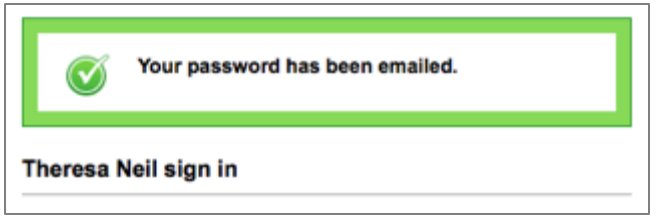
Visibility of system status - feedback

- Inform the user about what's going on:
  - show appropriate feedback and progress
  - do not show blank screens
  - do not show static "load" or progress messages

Visibility of system status: examples



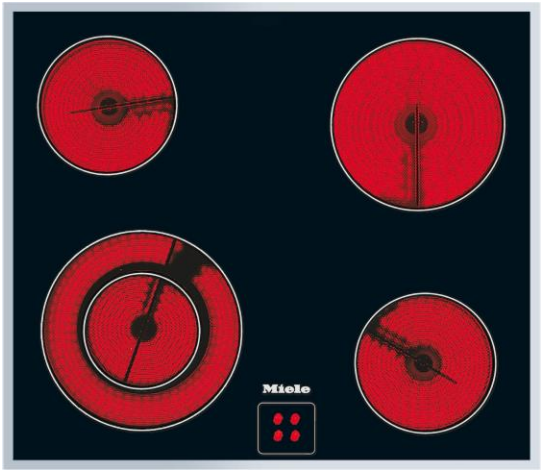
Microsoft Live
Password strength is shown as the password is entered. Colors are used to augment the message.



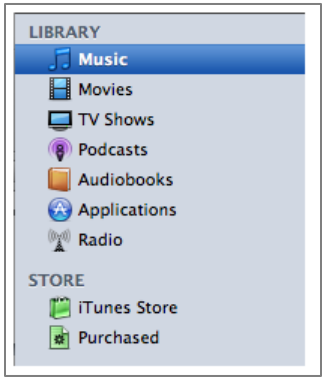
Tick
A feedback message is displayed when an action is performed

Match between system and real world - metaphor

- There must be a match between the system's interface controls and the real world
- The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms
- Follow real-world conventions, making information appear in a natural and logical order



Match between system and real world - examples

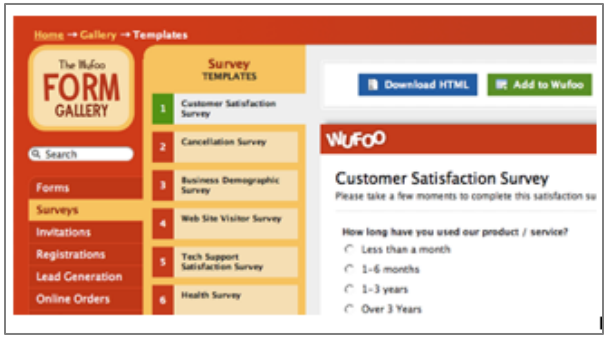


iTunes
Organized as a library that contains your media library: music, movies, TV shows, audiobooks. Beneath the Library is the Store where you can buy more media to put in your Library.

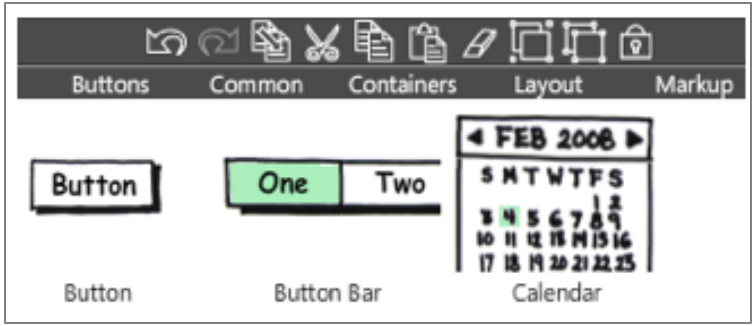
User control and freedom - navigation

- Users often choose system functions by mistake and will need a clearly marked “emergency exit” to leave the unwanted state without having to go through an extended dialog.
- Support undo and redo and a clear way to navigate.
- Provide bread crumbs to clearly show where the user is.

User control and freedom - examples



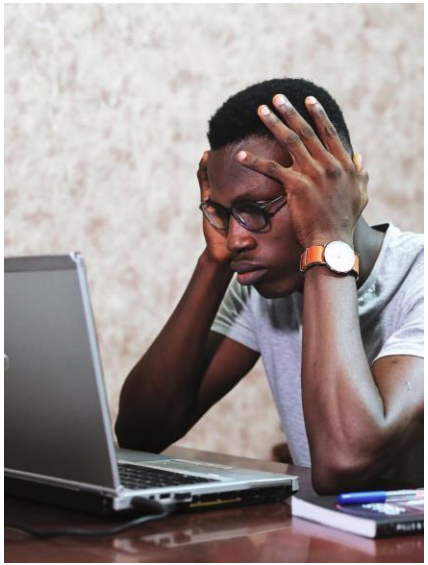
Wufoo
Clearly marks where the person is and where they can go by showing the selection in each menu



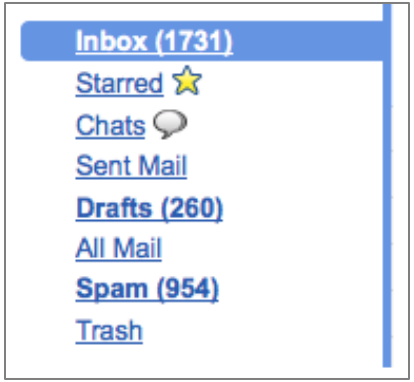
Balsamiq
Undo and Redo buttons are available in the toolbar, and can also be accessed with the standard keyboard shortcuts

Consistency and standards

- Users should not have to wonder whether different words, situations, or actions mean the same thing.
- Follow platform conventions.



Consistency: examples



Gmail
When Gmail was designed, they based the organizational folders on the same ones used in other client email applications: Inbox, Drafts, Sent Mail.

Microsoft Office
Word, Excel, and PowerPoint all use the same style toolbar with the same primary menu options: Home, Insert, Page Layout.

Error prevention

- Even better than good error messages is a careful design which prevents a problem from occurring in the first place.

- Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.

Error prevention: examples

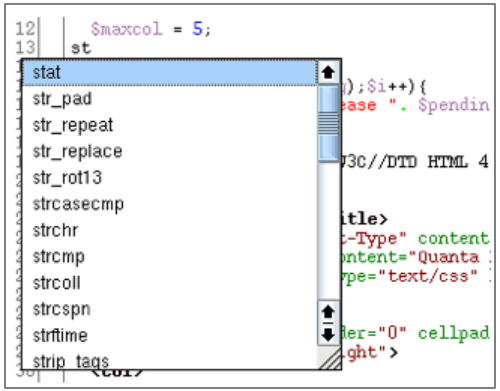
Yammer
Disables the update button after it is clicked, so the person cannot update the post twice by accident

Example from "Web form Design: Filling in the Blanks" by Luke W.
Make the primary action prominent with a larger click area. Cancel and other secondary actions are just shown as links

Recognition rather than recall

- Minimize the user's memory load.
- Make objects, actions, and options visible.
- The user should not have to remember information from one part of the dialogue to another.
- Instructions for use of the system should be visible or easily retrievable whenever appropriate.

Recognition: examples

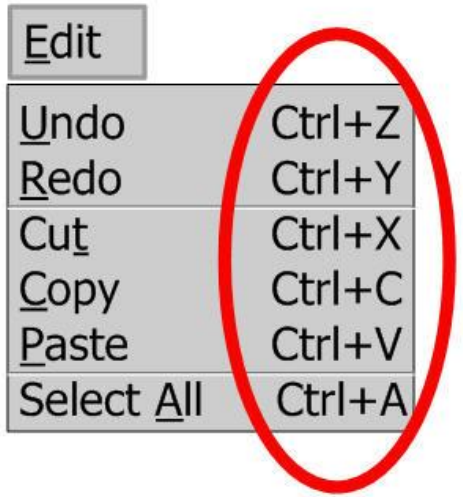


Quanta IDE
Auto completion for coding in a development environment

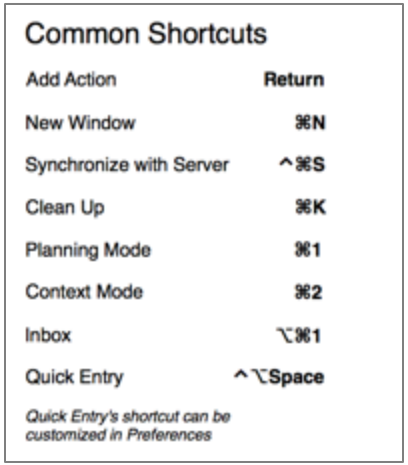
Keynote
Previews the fonts you can pick from, instead of just the font name

Flexibility and efficiency of use

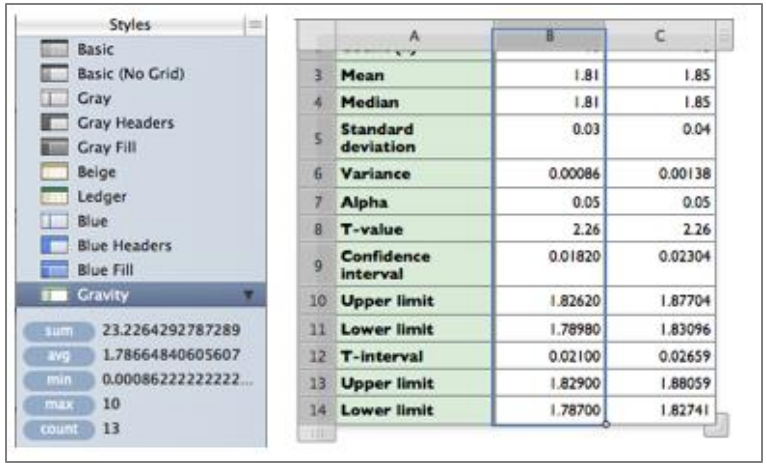
- **Accelerators** — unseen by the novice user — may often speed up the interaction for the expert user so that the system can cater to both inexperienced and experienced users
- Allow users to tailor frequent actions



Flexibility and efficiency: examples



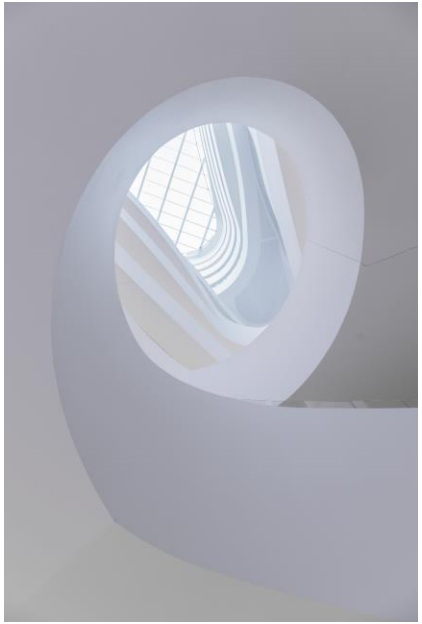
OmniFocus
List of keyboard shortcuts and accelerators



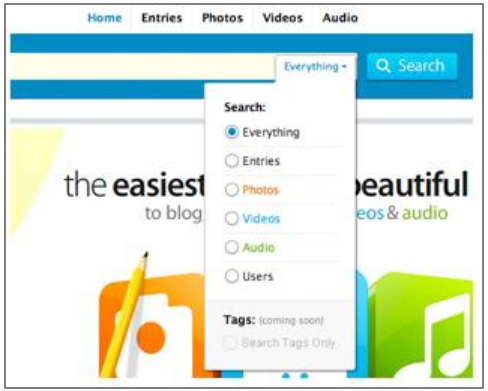
Numbers by Apple
Previews common function results on the left when a column is selected, more efficient than clicking on an action in the toolbar

Aesthetic and minimalist design

- Dialogues should not contain information which is irrelevant or rarely needed
- Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility
- Visual layout should respect the principles of contrast, repetition, alignment, and proximity.



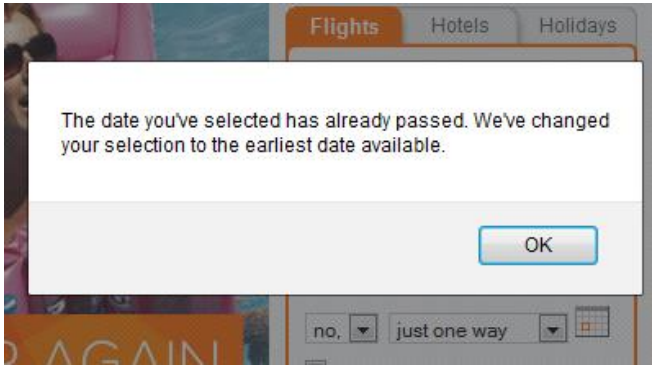
Aesthetics: example



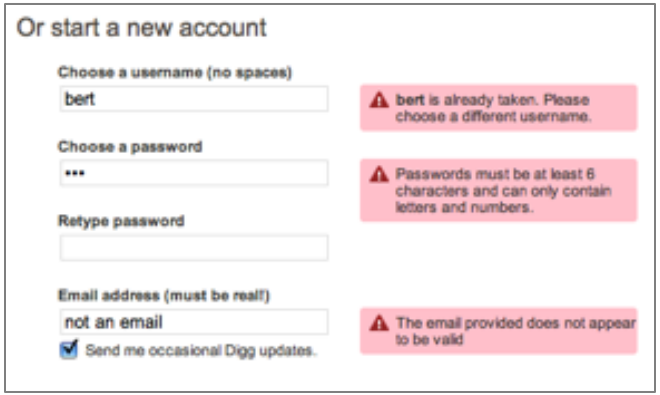
Kontain's search menu exemplifies the four principles of visual design:
1.Contrast: bold text is used for the two labels in the search
2.Repetition: the orange, blue, and green text match the media types
3.Alignment : strong left alignment of text, right aligned drop down
4.Proximity: a light rule is used to separate tags from the other options

Help users recognise, diagnose and recover from errors

- Help users recognize, diagnose, and recover from errors.
- Error messages should be expressed in plain language (no jargon), precisely indicate the problem, and constructively suggest a solution.



Error recognition and recovery: examples

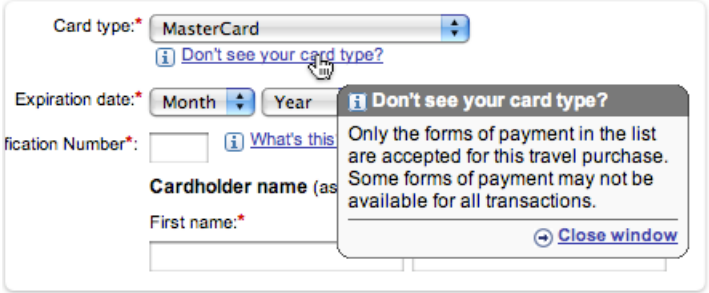


Digg
Provides immediate feedback with specific instructions

Humorous ‘ Page Not Found’ Error
Uses a funny image and text, but provides viable alternatives (article listings and blog link) and a course of action (report it)

Help and documentation

- Even though it is better if software can be used without documentation, it may be necessary to provide help and documentation.
- Any such information should be contextual, easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.

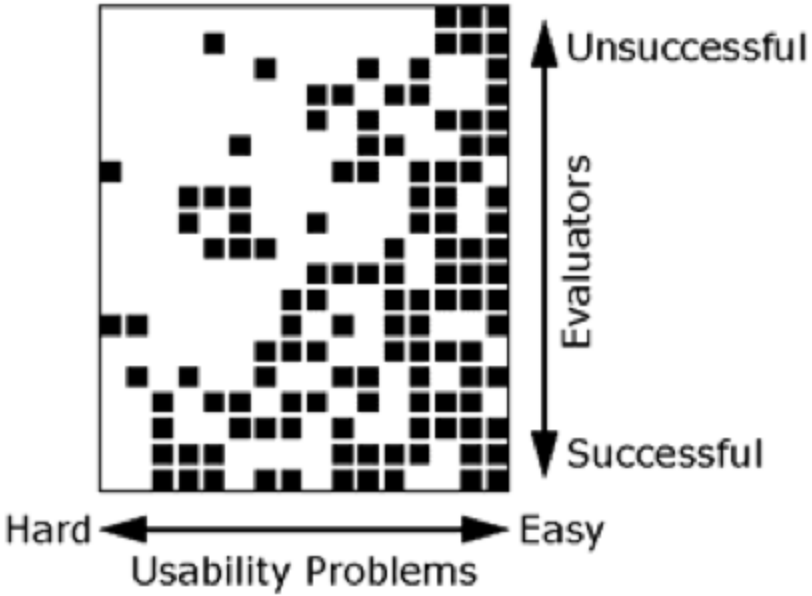


Help and documentation: examples

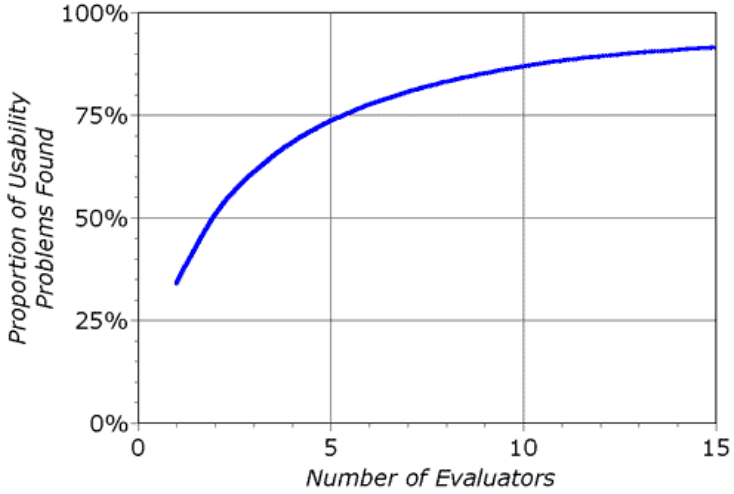
Picnik
Contextual tips in Picnik are clear and easy to navigate

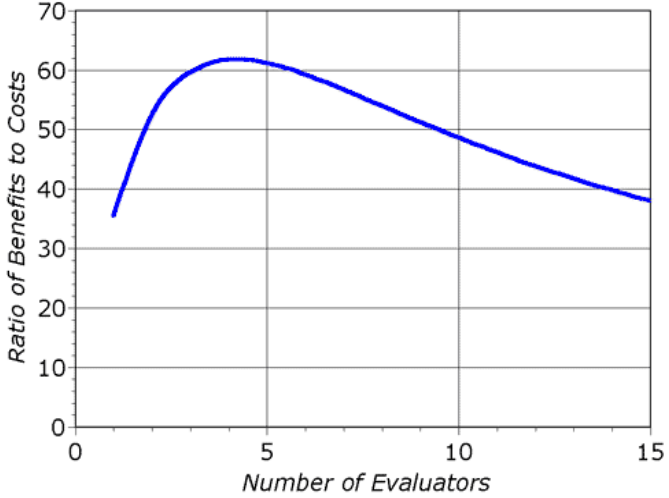
GoodBarry
Embedded videos can be used to showcase features as well as get people started using the product

How many evaluators are needed for heuristic evaluation?



Practical considerations





How to run a heuristic evaluation 1

- Each of the 3 – 5 evaluators does a heuristic evaluation of an interface alone
- Sometimes a facilitator can record the evaluator's comments, sometimes the evaluator does it
- A facilitator **can** answer evaluators' questions, in contrast to traditional user testing, particularly if it's not a walk up and use system
- Heuristic evaluation can be done on paper prototypes

How to run a heuristic evaluation 2

- Heuristic evaluations typically last 1 – 2 hours, but it does depend on the complexity of the software
- The expert goes through the interface several times – first time to get a feel for the system, second time to focus on specific elements
- Evaluators can be given scenarios that describe typical usage scenarios (built from a task analysis of users)
- Evaluators produce a list of usability problems: the usability principle and the design feature that violated it

Benefits of heuristic evaluation

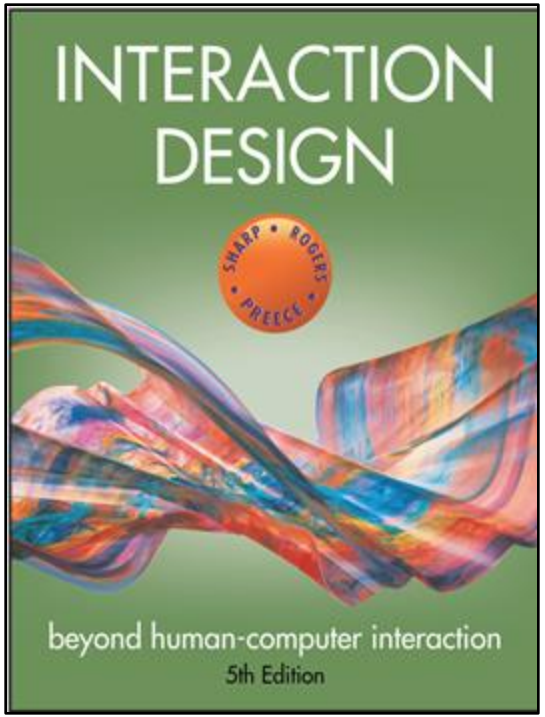
- Cheap
- Relatively easy
- Instant gratification lists of problems are available immediately after the inspection
- It can be carried out with low numbers of participants
- Fits in with most software development processes
- Cost effective

Drawbacks of heuristic evaluation

- Important issues may get missed
- Might identify false issues
- Many trivial issues are often identified, making it seem overly critical
- Experts have biases

Reading

- *Interaction Design: Beyond Human-Computer Interaction* covers all HCI evaluation techniques. It's available through the university library as an eBook. Read about the evaluation techniques covered in this lecture to deepen your understanding
- Read the original Nielsen paper on heuristic evaluation: https://www.nngroup.com/articles/how-to-conduct-a-heuristic-evaluation/



Reading 2

- Explore the materials (papers, articles and videos) on heuristic evaluation on the Nielsen Norman group website:

https://www.nngroup.com/articles/ten-usability-heuristics/

Before the workshop today

- Please review the lecture materials on the Think Aloud and Heuristic Evaluation techniques
- Your workshop activities will involve evaluating your games using these two techniques

HCI Evaluation Part Two

Dr Jon Bird
jon.bird@bristol.ac.uk

Thanks to Stuart Gray, Pete Bennett, Simon Lock, Thomas Bale, Harry Field who developed some of these slides

Today's Lecture

- Questionnaires
- NASA Task Load Index (NASA TLX)
- System Usability Scale (SUS)
- Statistical tests to determine if the perceived workload or system usabiity score has changed significantly

Questionnaires - defined

- Questionnaires involve asking people to answer questions either on paper or digitally e.g. on a webpage or app
- They can be used at scale with low resource requirements
- They generate a collection of demographic data and user opinions
- They can be used to evaluate designs and for understanding user requirements

Questionnaires - tips

- Ensure that you are asking a feasible number of questions (question fatigue is a thing)
- Watch out for leading questions e.g. “Why did you have difficulty with the navigation?”
- It is difficult to produce your own questionnaires
- It is best to use existing questionnaires that have been validated i.e. they measure what they claim to be measuring
- I’ll now introduce you to two widely used questionnaires

NASA TLX

- The NASA Task Load Index (TLX) is a questionnaire that estimates a user's perceived workload when using a system.
- Workload is a complex construct but essentially means the amount of effort people have to exert, both mentally and physically, to use a system.
- It was developed by Sandra Hart of NASA's human performance group and Lowell Staveland of San Jose University.
- The focus is on measuring the "immediate often unverbalized impressions that occur spontaneously" (Hart and Staveland, 1988). These are difficult or impossible to observe objectively.

NASA TLX 2

- Originally the NASA TLX questionnaire was developed for use in aviation but it's since been used in many different domains, including air traffic control, robotics, the automative industry, healthcare, website design and other technology fields.
- Since it was introduced in 1988, it has had over 8000 citations.
- It is viewed as the gold standard for measuring subjective workload.

NASA TLX 3

- Originally it was developed as a paper and pencil questionnaire but there are also free apps for iOS and Android
- The official website is here: https://humansystems.arc.nasa.gov/groups/TLX/index.php

NASA TLX 4

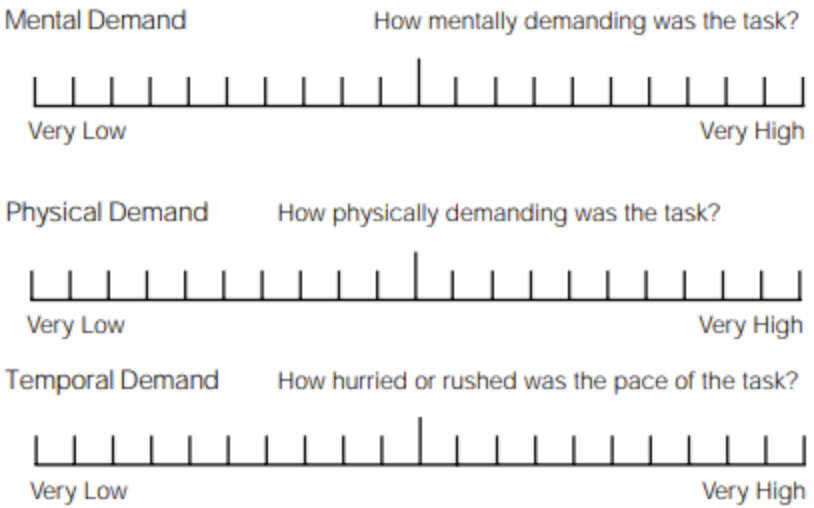
- The NASA TLX uses a multi-dimensional rating procedure that derives an overall workload score based on a weighted average of ratings on six subscales:

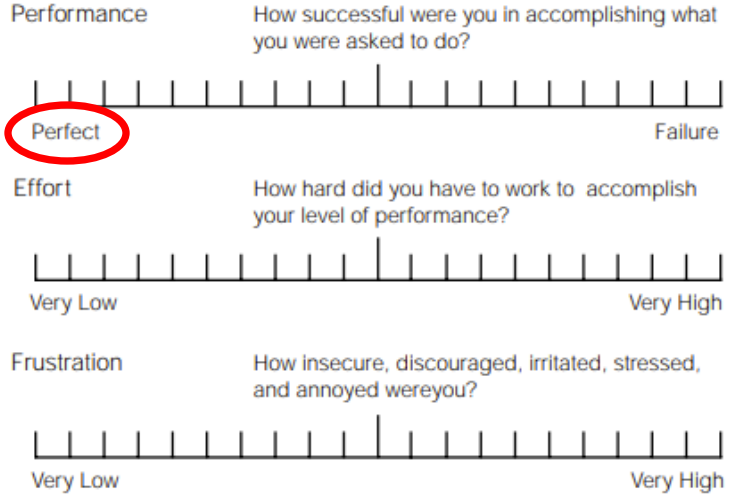
  - Mental Demand
  - Physical Demand
  - Temporal Demand
  - Performance
  - Effort
  - Frustration

NASA TLX 5

- Mental demand – how much mental and perceptual activity was required?
- Physical demand – how much physical activity was required?
- Temporal demand – how much time pressure did the user feel due to the rate at which tasks occurred?
- Frustration – how insecure, discouraged or irritated did the user feel in the task?
- Effort – how hard did the user have to work (mentally and physically) to accomplish their level of performance?
- Performance – how successfully did the user think they accomplished the task?

NASA TLX 6


Mental Demand
How mentally demanding was the task?
Very Low
Very High
Physical Demand
How physically demanding was the task?
Very Low
Very High
Temporal Demand
How hurried or rushed was the pace of the task?
Very Low
Very High


Performance
How successful were you in accomplishing what you were asked to do?
Perfect
Failure
Effort
How hard did you have to work to accomplish your level of performance?
Very Low
Very High
Frustration
How insecure, discouraged, irritated, stressed, and annoyed wereyou?
Very Low
Very High

NASA TLX Scoring 1

- Users answer the NASA TLX after they have completed a task. This is necessary as asking them to complete it during task is typically not possible. However, it may mean that users forget details of the perceived workload.
- The questionnaire is scored in a two step process:

1. Identifying the relative importance of the 6 dimensions on a user's perceived workload
2. Rating each of the 6 dimensions on a scale

NASA TLX Relative weighting of dimensions 1

- A user reflects on the task they've been asked to perform and is shown each paired combination of the six dimensions to decide which is more related to their personal definition of workload as related to the task.
- This means a user considers 15 paired comparisons. For example, they need to decide whether Performance or Frustration "represents the more important contributor to the workload for the specific task you recently performed."
- Each time a dimension is selected as more important it receives a score of 1. The total score is the weight of the dimension and ranges from 0 to 5.
- The sum of the weights should be 15.

NASA TLX Relative weighting of dimensions 2

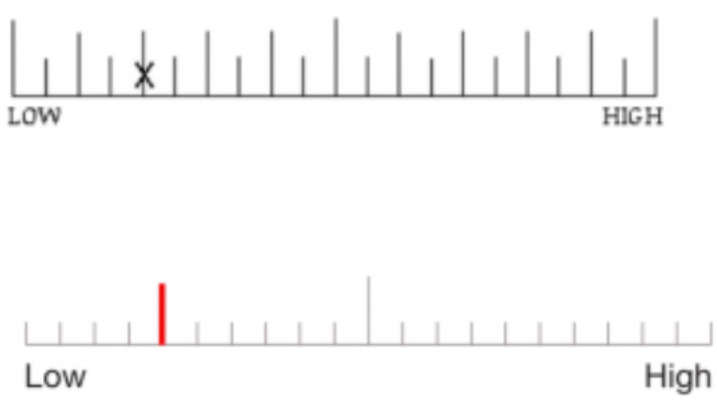
- The relative weighting of the six dimensions is often **not** measured or used.
- Not measuring the relative weighting makes the NASA TLX simpler to administer.
- Several studies have compared raw TLX scores to weighted TLX scores and have found mixed results (some showing better sensitivity when removing weights, others showing no difference, and others showing less sensitivity).
- When the dimensions are not rated the method is called the ‘raw TLX score’

NASA TLX Rating the dimensions 1

- Users mark their score on each of the six dimensions.
- Each dimension consists of a line with 21 equally spaced tick marks, which divide the line from 0 to 100 in increments of 5. If a user marks between two ticks then the value of the right tick is used.
- The score on a dimension is calculated as the tick number (1, 21) – 1 multiplied by 5.

NASA TLX Rating the dimensions 2

- For example, the images show the rating on a paper questionnaire (top) and on a mobile app (bottom)
- The fifth tick mark is selected, so the rating score is: (5 – 1) * 5 = 20



NASA TLX What do the scores tell us?

- If the weights are used then the individual ratings on each of the dimensions are multiplied by their respective weights, summed and divided by 15, resulting in an aggregate perceived workload score for a task ranging from 0 – 100.
- If the weights are not used then the individual ratings on each of the dimensions can be summed and divided by 6, resulting in an aggregate perceived workload score ranging from 0 – 100.
- The individual ratings on the 6 dimensions also give some insight in to where the workload is coming from. This can be helpful for developers hoping to improve their design.

NASA TLX Validity

- Hart and Staveland validated that the sub-scales measure different sources of workload.
- Subsequent independent studies have also found that the NASA TLX is a valid measure of subjective workload (Rubio et al, 2004; Xiao et al, 2005).

System Usability Survey (SUS)

- The System Usability Scale (SUS) provides a “quick and dirty”, reliable tool for measuring usability.
- It was created by John Brooke in 1986.
- It consists of a 10 item questionnaire with five response options for each item ranging from Strongly agree to Strongly disagree.
- It enables the evaluation of a wide variety of products and services, including hardware, software, mobile devices, websites and applications.

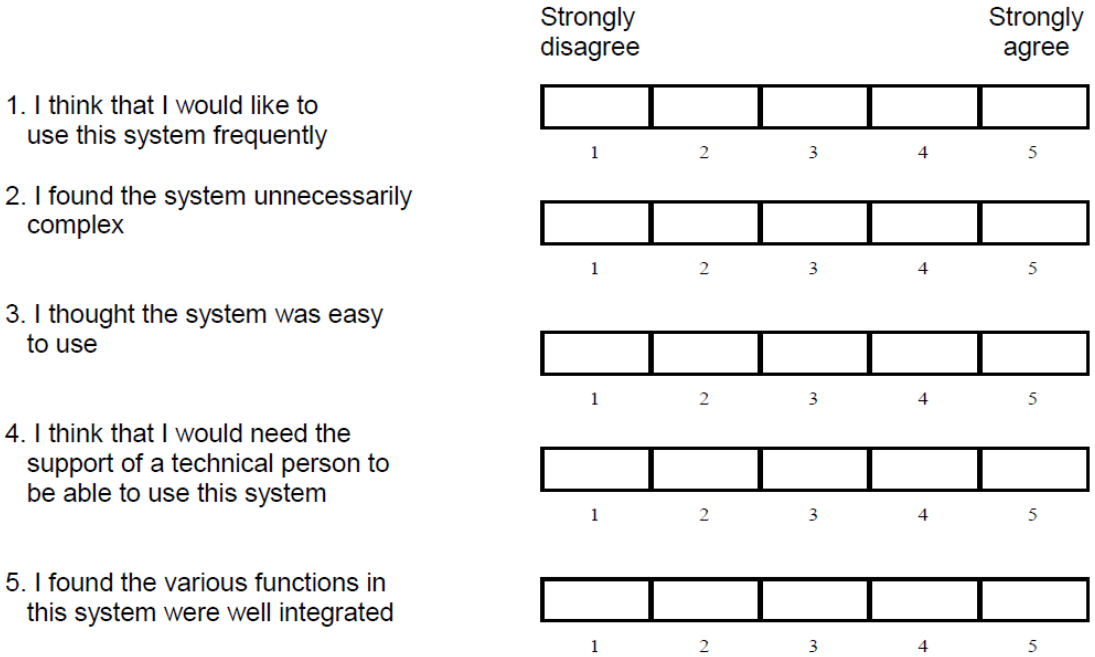
System Usability Survey (SUS) - benefits

- SUS has become an industry standard, with references in over 1300 articles and publications.
- The noted benefits of using SUS include:
- It is a very easy scale to administer to participants
- It can be used on small sample sizes with reliable results
- The SUS has been validated and shown to effectively differentiate between usable and unusable systems

System Usability Survey (SUS) - scale

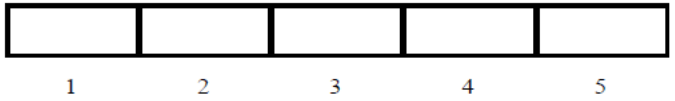
- When an SUS is used, participants are asked to score the 10 items with one of five responses that range from Strongly Agree to Strongly disagree i.e. using a five point Likert scale

System Usability Survey (SUS) – scale 2

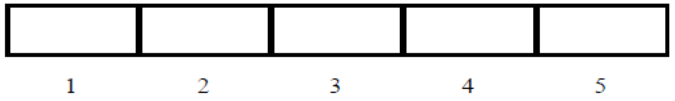


System Usability Survey (SUS) – scale 3

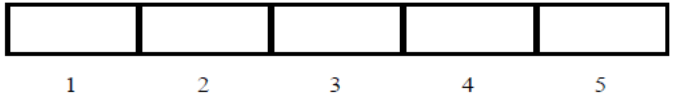
6. I thought there was too much inconsistency in this system



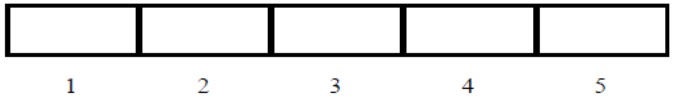
7. I would imagine that most people would learn to use this system very quickly



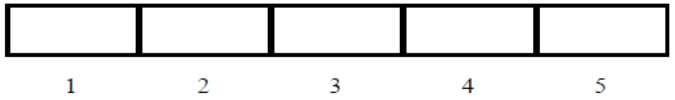
8. I found the system very cumbersome to use



9. I felt very confident using the system



10. I needed to learn a lot of things before I could get going with this system



System Usability Survey (SUS) – scoring 1

- The SUS is given to users when they have completed using the system which is being evaluated
- They score each of the 10 items by marking one of the five boxes
- The SUS yields a single number representing a composite measure of the overall usability of the system being studied. Note that scores for individual items are **not** meaningful on their own.

System Usability Survey (SUS) – scoring 2

- To calculate the SUS score, first sum the score contributions from each item. Each item's score contribution will range from 0 to 4.
- For items 1,3,5,7,and 9 (the odd numbered items) the score contribution is the scale position minus 1. For items 2,4,6,8 and 10 (the even numbered items) the contribution is 5 minus the scale position.
- Multiply the sum of the scores by 2.5 to obtain the overall score.
- SUS scores have a range of 0 to 100.
- Based on research, a SUS score above **68** would be considered above average and anything below 68 is below average.

Statistical testing

- You might get a user to rate the SUS of two different designs and want to know if one design is significantly better than the other.
- Similarly, you might want to know if two levels of difficulty in your game are significantly different so you get a user to rate the workload of both levels.
- To determine whether the differences in scores are significantly different we can use a statistical test

Statistical testing 2

- There are many statistical tests but I am going to show you two that will be useful for your project.
- The first is the Wilcoxon Signed Rank Test and it is ideal for analysing data from Likert and other scales e.g. the NASA TLX and SUS.
- It is used when **one user carries out two evaluations** e.g. rates the workload of your game at two different difficulty levels.
- It is a good test when you have small numbers of users – the minimum is 5; however, it's better at identifying significant differences when you have larger numbers of users.

Statistical testing 3

- Make a table where each row represents a user's scores and each column a separate evaluation score.
- I've shown the results of three users evaluating the workload of a game at two difficulty levels using the NASA TLX.
- You need a minimum of 5 and ideally more

User ID	Workload level 1	Workload level 2
U1	25	67
U2	32	56
U3	18	43

Statistical testing 4

- Enter the data into the online calculator: https://www.statology.org/wilcoxon-signed-rank-test-calculator/
- Look up the calculated **W test statistic** in the table of critical values
- To do this you need to know N, which is the number of users, and the significance level, which we will set at 0.05
- This means that if a significant difference is found then it is 95% certain that this is a real difference rather than due to randomness

Statistical testing 5

- We use an alpha value aka significance level of 0.05
- We find the row that corresponds to our number of users aka n.
- If we have 10 users then the W test statistic generated by the online calculator **needs to be less than** 8 otherwise there is no significant difference.

	Alpha value				
n	**0.005**	**0.01**	**0.025**	**0.05**	**0.10**
5	-	-	-	-	0
6	-	-	-	0	2
7	-	-	0	2	3
8	-	0	2	3	5
9	0	1	3	5	8
10	1	3	5	8	10
11	3	5	8	10	13
12	5	7	10	13	17
13	7	9	13	17	21
14	9	12	17	21	25
15	12	15	20	25	30
16	15	19	25	29	35
17	19	23	29	34	41
18	23	27	34	40	47
19	27	32	39	46	53
20	32	37	45	52	60

Statistical testing 6

- If we are comparing two sets of values generated by two different groups e.g. experienced gamers and novice gamers then we use a different test to see if they are significantly different
- This is known as the Mann-Whitney U test. There is also an online calculator and you can read about the test here:

https://www.statology.org/mann-whitney-u-test/

Reading

- Read the original paper on the NASA TLX:

  Hart, S. G., & Staveland, L. E. (1988). Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. In *Advances in psychology* (Vol. 52, pp. 139-183). North-Holland.
- Read the original SUS paper
- Read more about the Wilcoxon signed rank test

Before the workshop next week

- Please review the lecture materials on the NASA TLX and SUS
- Your workshop activities will involve evaluating your games using these two techniques

Software Quality

Lecture 8

Ruzanna Chitchyan, Jon Bird, Pete Bennett
TAs: Alex Elwood, Alex Cockrean, Casper Wang

Overview

- Software quality and how to get to it
- Test-driven development
  - White box testing
  - Black box testing

Software Quality

Why is Software Quality relevant: Case of Bard (Gemini)

Google Bard AI mistake just cost Google over $100 billion

By Philip Michaels last updated 8 days ago

AI-powered chatbot makes costly error in early demo

Comments (0)

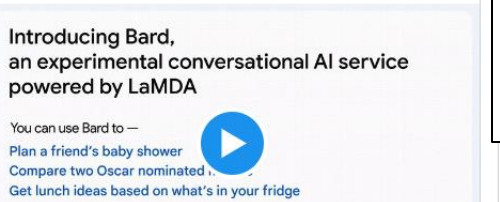
https://www.tomsguide.com/news/google-bard-ai-is-off-to-an-embarrassing-start

Isabel Angelo
@IsabelNAngelo · **Follow**

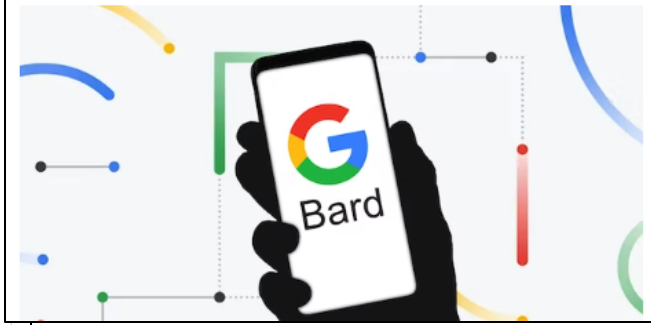
Unfortunately a simple google search would tell us that JWST actually did not "take the very first picture of a planet outside of our own solar system" and this is literally in the ad for Bard so I wouldn't trust it yet

Google @Google

Bard is an experimental conversational AI service, powered by LaMDA. Built using our large language models and drawing on information from the web, it's a launchpad for curiosity and can help simplify complex topics → goo.gle/3HBZQtu



The chatbot generated an incorrect fact about the James Webb Space Telescope in its very first public demo. The incident has dramatically highlighted one of the most pertinent dangers for marketers using AI: it doesn't always tell the truth.



https://www.thedrum.com/news/2023/02/09/attention-marketers-google-s-100bn-bard-blunder-underscores-current-dangers-using-ai

Why is Software Quality relevant?

- Reputation
- Cost of Product and Maintenance
- Software Certification
- Organizational Certification
- Legality
- Moral/ethical codes of practice

Software Quality is Multi-dimensional

- Subjective or “fitness for use”: as perceived by an individual user (e.g., aesthetics of GUI, missing functionality…)
- Objective or “conformance to requirements”: can be measured as a property of the product (e.g., detailed documentation, number of bugs, compliance with regulations ….)
- Practical: what does it mean to your team and your clients?

Quality Models: ISO/IES25010

- **Functional Suitability**
  - Functional Completeness
  - Functional Correctness
  - Functional Appropriateness
- **Performance Efficiency**
  - Time Behaviour
  - Resource Utilisation
  - Capacity
- **Compatibility**
  - Co-existence
  - Interoperability
- **Usability**
  - Appropriateness
  - Realisability
  - Learnability
  - Operability
  - User Error Protection
  - User Interface Aesthetics
  - Accessibility
- **Reliability**
  - Maturity
  - Availability
  - Fault Tolerance
  - Recoverability
- **Security**
  - Confidentiality
  - Integrity
  - Non-repudiation
  - Authenticity
  - Accountability
- **Maintainability**
  - Modularity
  - Reusability
  - Analysability
  - Modifiability
  - Testability
- **Portability**
  - Adaptability
  - Installability
  - Replaceability

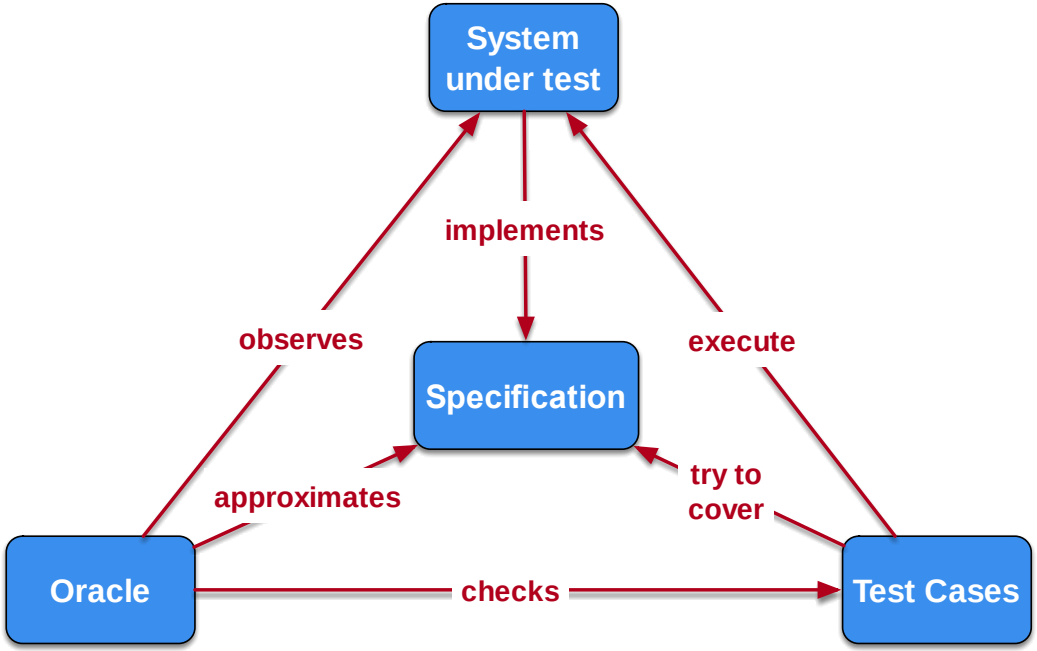
Steps Towards Software Quality:

- Use a standard development process
- Use a coding standard
    - Compliance with industry standards (e.g., ISO, Safety, etc.)
    - Consistent code quality
    - Secure from start
    - Reduce development costs and accelerate time to market
- Define and monitor metrics (defect metrics and complexity metrics)
    - High complexity leads to higher number of defects
- Identify and remove defects
    - Conduct manual reviews
    - Use Testing

Testing

Mauro Pezze and Michal Young. *Software testing and analysis - process, principles and techniques*. Wiley, 2007.

Testing process: key elements and relationships



From: M. Staats, M. W. Whalen, and M. P. E. Heimdahl. Programs, tests, and oracles: the foundations of testing revisited. In *Software Engineering (ICSE), 2011 33rd International Conference on*, pages 391–400. IEEE, 2011.

Testing: White Box

Mauro Pezze and Michal Young. *Software testing and analysis - process, principles and techniques*. Wiley, 2007.

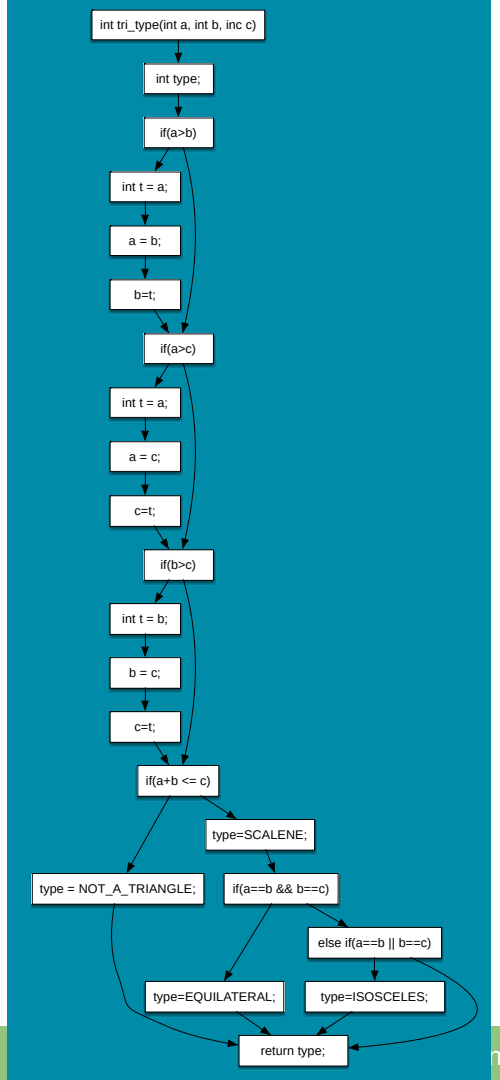
White Box Testing

- Access to software ``internals'':
  - Source code
  - Runtime state
  - Can keep track of executions.
- White box testing exploits this to
  - Use code to measure coverage
    - Many different ways
  - Drive generation of tests that maximise coverage

```
 1    int tri_type(int a, int b, int c) {
 2      int type;
 3      if (a > b)
 4        { int t = a; a = b; b = t; }
 5      if (a > c)
 6        { int t = a; a = c; c = t; }
 7      if (b > c)
 8        { int t = b; b = c; c = t; }
 9      if (a + b <= c)
10        type = NOT_A_TRIANGLE;
11      else {
12        type = SCALENE;
13        if (a == b && b == c)
14          type = EQUILATERAL;
15        else if (a == b || b == c)
16          type = ISOSCELES;
17      }
18    return type;
19    }
```

White Box Testing

- Access to software ``internals'':
  - Source code
  - Runtime state
  - Can keep track of executions.
- White box testing exploits this to
  - Use code to measure coverage
    - Many different ways
  - Drive generation of tests that maximise coverage.



White-Box Testing

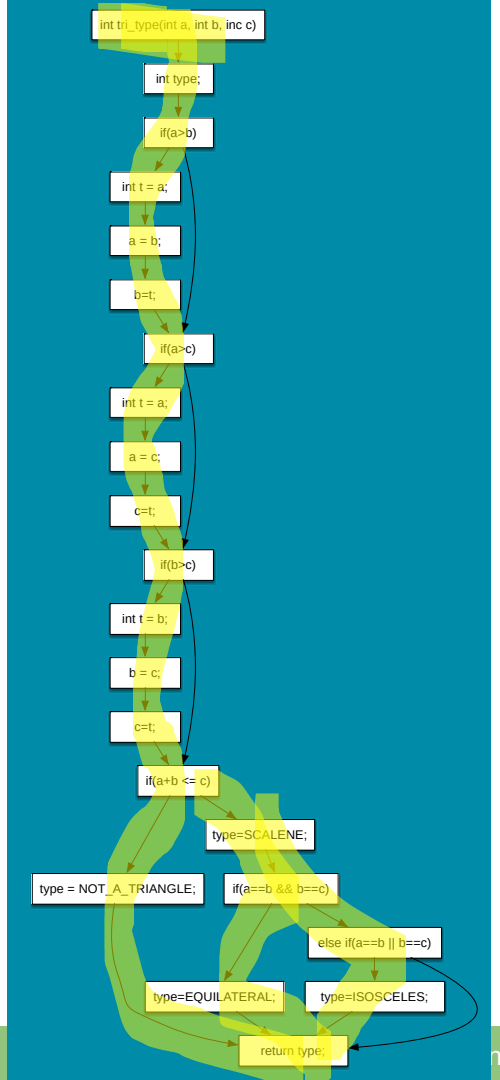
- Coverage Metrics:
  - Statement coverage
  - Branch coverage
  - Def-Use or Dataflow coverage
  - MC/DC (Modified Condition / Decision Coverage)
  - Mutation coverage…

- Prescribed metrics, e.g., DO178-B/C standard for civilian aircraft software
  - non-critical - statement coverage
  - safety-critical - MC/DC coverage

Statement Coverage

- Test inputs should collectively have executed each statement
- If a statement always exhibits a fault when executed, it will be detected
- Computed as:

$$Coverage = \frac{|Statements\ executed|}{|Total\ statements|}$$



Branch Coverage

- Test inputs should collectively have executed each branch
- Subsumes statement coverage
- Computed as:

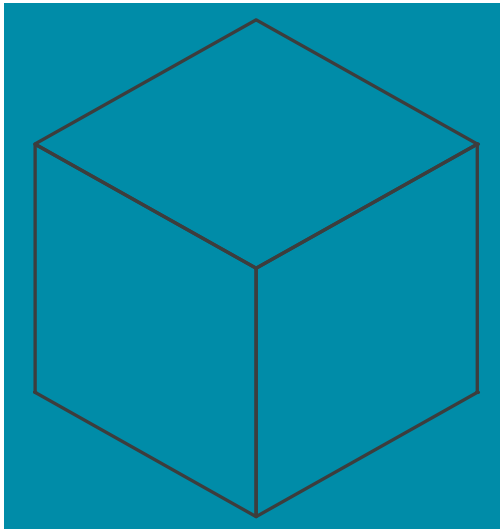
$$Coverage = \frac{|Branches\ executed|}{|Total\ branches|}$$

Testing: Black Box

Mauro Pezze and Michal Young. *Software testing and analysis - process, principles and techniques*. Wiley, 2007.

Black Box Testing

- No access to “internals”
  - May have access, but don’t want to
- We know the interface
  - Parameters
  - Possible functions / methods
- We may have some form of specification document



Testing Challenges

- Many different types of input
- Lots of different ways in which input choices can affect output
- An almost infinite number of possible inputs & combinations

Equivalence Partitioning (EP) Method

Identify tests by analysing the program interface

1. Decompose program into “functional units”
2. Identify inputs / parameters for these units
3. For each input

    a) Identify its limits and characteristics

    b) Define “partitions” - value categories

    c) Identify constraints between categories

    d) Write test specification

Example – Generate Grading Component

The component is passed an exam mark (out of 75) and a coursework (c/w) mark (out of 25), from which it generates a grade for the course in the range 'A' to 'D'. The grade is calculated from the overall mark which is calculated as the sum of the exam and c/w marks, as follows:

greater than or equal to 70 - 'A'

greater than or equal to 50, but less than 70 - 'B'

greater than or equal to 30, but less than 50 - 'C'

less than 30 - 'D'

Where a mark is outside its expected range then a fault message ('FM') is generated. All inputs are passed as integers.

EP – 1. Decompose into Functional Units

- Dividing into smaller units is good practice
  - Possible to generate more rigorous test cases.
  - Easier to debug if faults are found.
- E.g.: dividing a large Java application into its core modules / packages
- Already a functional unit for the Grading Component example

EP – 2. Identify Inputs and Outputs

- For some systems this is straightforward
  - E.g., the Triangle program:
    - Input: 3 numbers,
    - Output: 1 String
  - E.g., Grading Component
    - Input: 2 integers: exam mark and coursework mark
    - Output: 1 String for grade
- For others less so. Consider the following:
  - A phone app.
  - A web-page with a flash component.

EP – 3.a Identify Categories

Category	Description
Valid	valid exam mark
	valid coursework mark
	valid total mark
Invalid	invalid exam mark
	invalid coursework mark
	Invalid total mark

EP: 3.b Define “Partitions” - value categories

- Significant value ranges / value-characteristics of an input

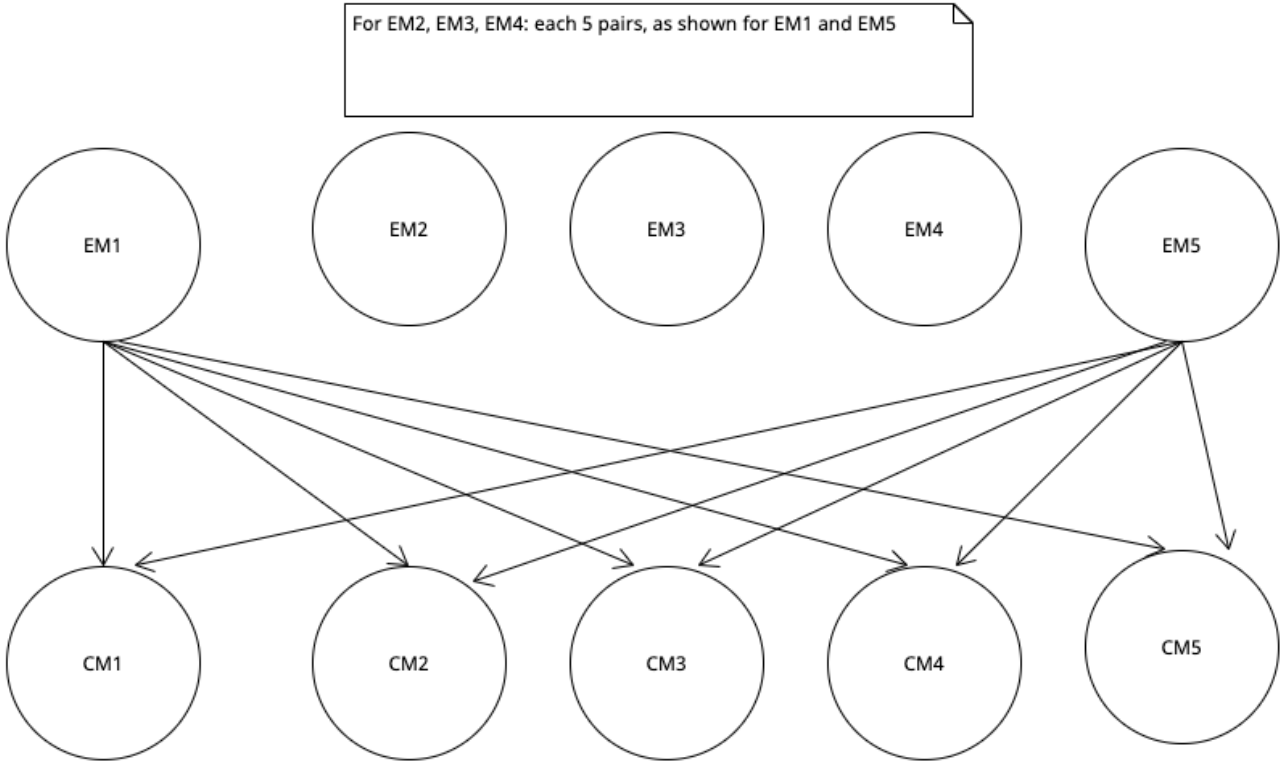
Category	Description	Partition
Valid	EM_1 valid exam mark	0 ≤ Exam mark ≤ 75
	CM_1 valid coursework mark	0 ≤ Coursework mark ≤ 25
Invalid	EM_2 invalid exam mark	Exam mark > 75
	EM_3 invalid exam mark	Exam mark < 0
	EM_4 invalid exam mark	alphabetic
	EM_5 invalid exam mark	Other real number (outside of EM_1)
	CM_2 invalid coursework mark	Coursework mark > 25
	CM_3 invalid coursework mark	Coursework mark < 0
	CM_4invalid coursework mark	alphabetic
	CM_5 invalid coursework mark	Other real number (outside of CM_1)

EP – 3. c Identify Constraints between Categories

- Not all categories can combine with each other

Category		Condition
valid exam mark	EM_1	0 ≤ Exam mark ≤ 75
invalid exam mark	EM_2	Exam mark > 75
invalid exam mark	EM_3	Exam mark < 0
invalid exam mark	EM_4	alphabetic
invalid exam mark	EM_5	Other real number
valid coursework mark	CM_1	0 ≤ Coursework mark ≤ 25
invalid coursework mark	CM_2	Coursework mark > 25
invalid coursework mark	CM_3	Coursework mark < 0
invalid coursework mark	CM_4	alphabetic
invalid coursework mark	CM_5	Other real number

EP – 3. d Write Test Specifications



Example: Inputs and Expected Outputs

The test cases corresponding to partitions derived from the input exam mark are:

Test Case	1	2	3
Input (exam mark)	44	-10	93
Input (c/w mark)	15	15	15
total mark (as calculated)	59	5	108
Partition tested (of exam mark)	$0 \leq e \leq 75$	$e < 0$	$e > 75$
Exp. Output	'B'	'FM'	'FM'

Boundary Values

- Most frequently errors occur in "edge" cases
  - Test just under boundary value
  - Test just above the boundary value
  - Test the boundary value

How do we go about using this?

- Testing applied in Java: Use JUnit
  - uses “Assertions” to test the code
  - Allow us to state what *should* be the case
  - If assertions do not hold, JUnit’s logging mechanisms reports failures
  - Various types of assertion are available, e.g., assertEquals(expected, actual); assertTrue(condition); assertFalse(condition); assertThat (value, matchingFunction)

Review

- What is Software Quality?
- What are key elements and relationships for test specifications?
- How do we carry out white-box testing?
- How do we carry out black-box testing?