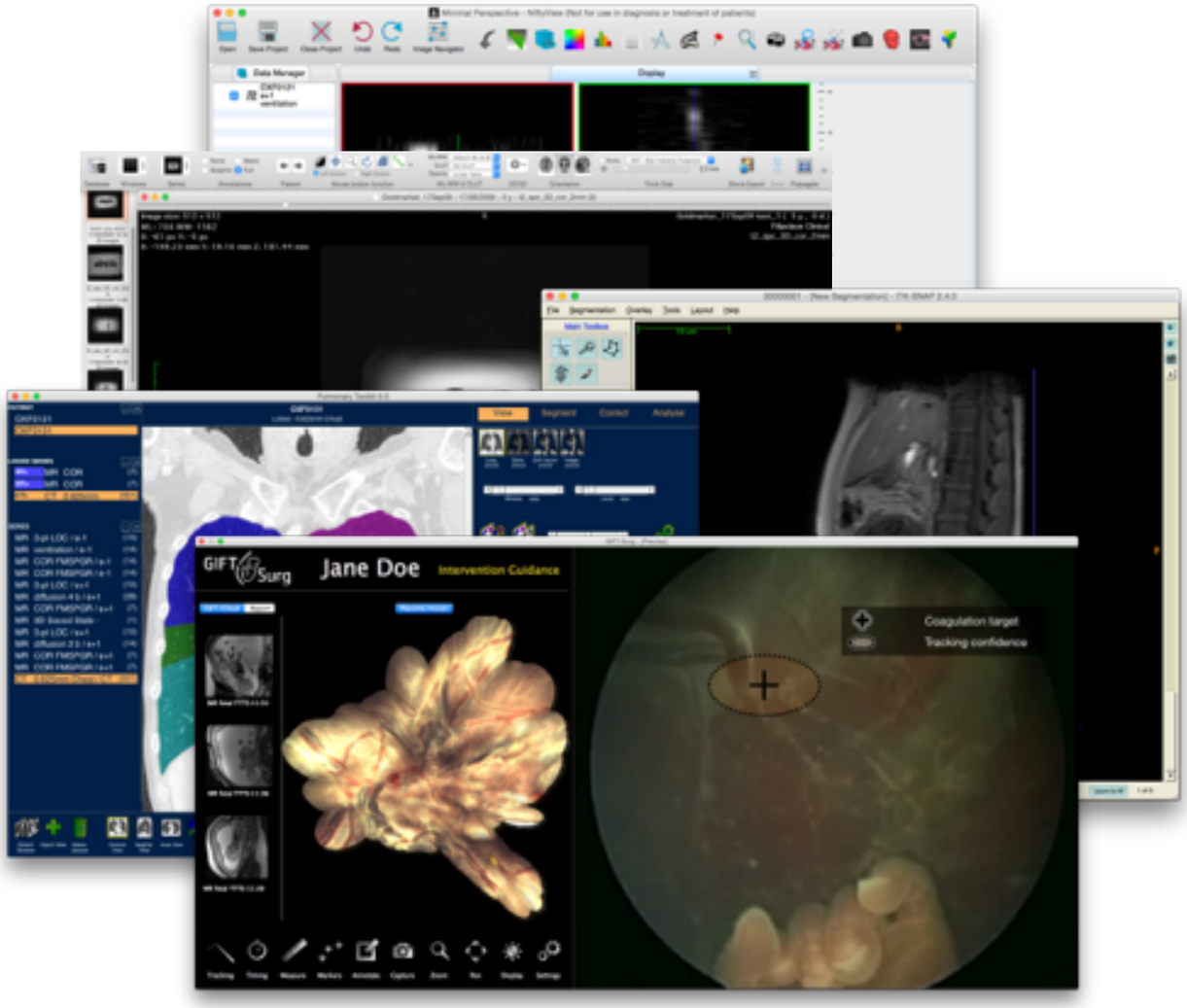
Clinical software design



Tom Doel

Why good design?

- Design is fundamental to the usability and reusability of software.
- Much of the software we develop will be used by others, whether researchers or clinicians.
- They will usually have a very specific goal in mind.
- Good design can help them achieve this goal with more efficiency and greater accuracy.

User experience (UX)

- The visual design - look and feel of the application

- Workflow - the procedures a user follows to achieve their goal

- Visual design is more than just a marketing and PR exercise

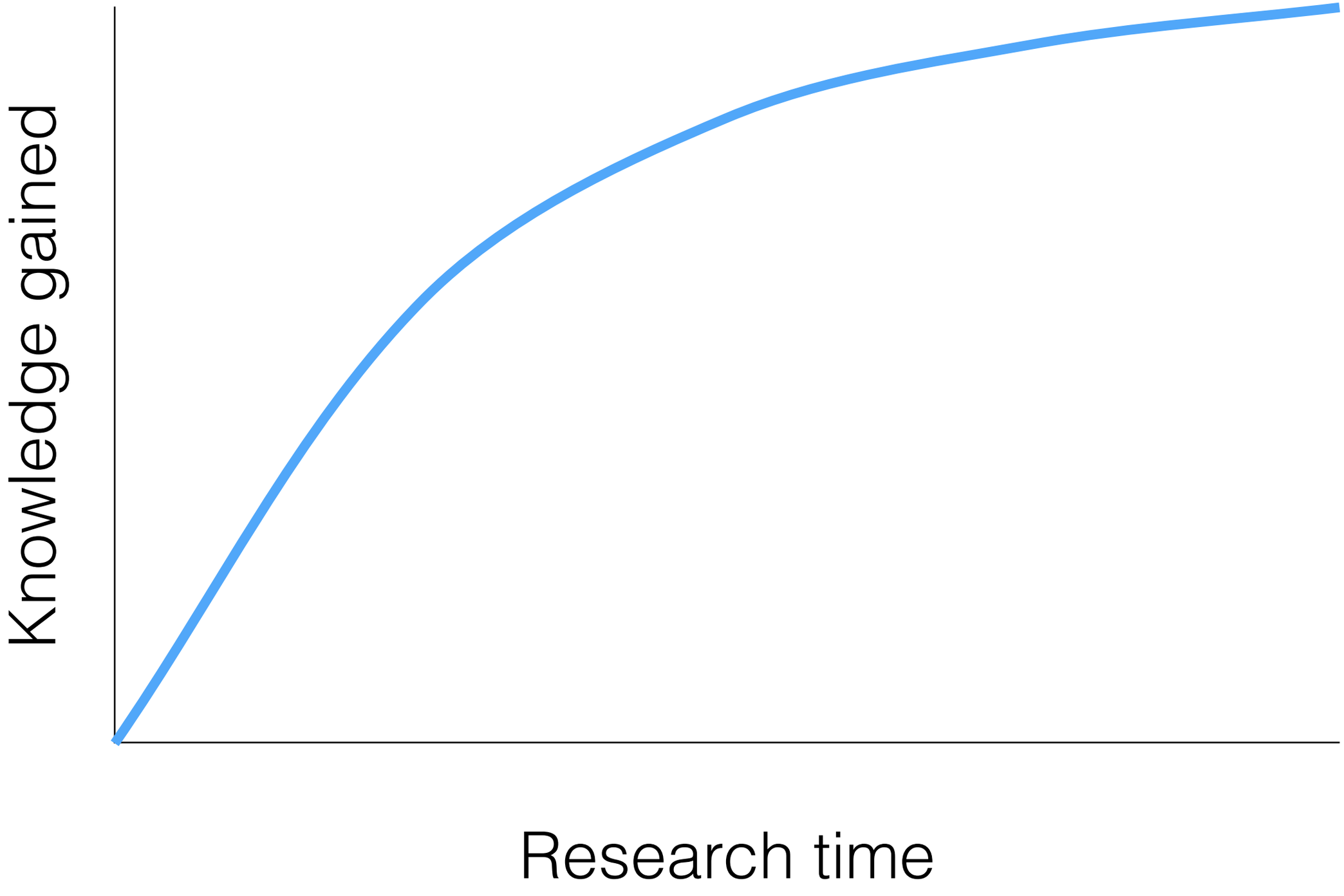
- Example: the London Underground“roundel” was created to help passengers distinguish station names from advertising

- The 1933 “electrical circuit” London Underground map by Harry Beck has less information than a geographic map, but makes it is easier to navigate. It is the basis of most metro system maps used today

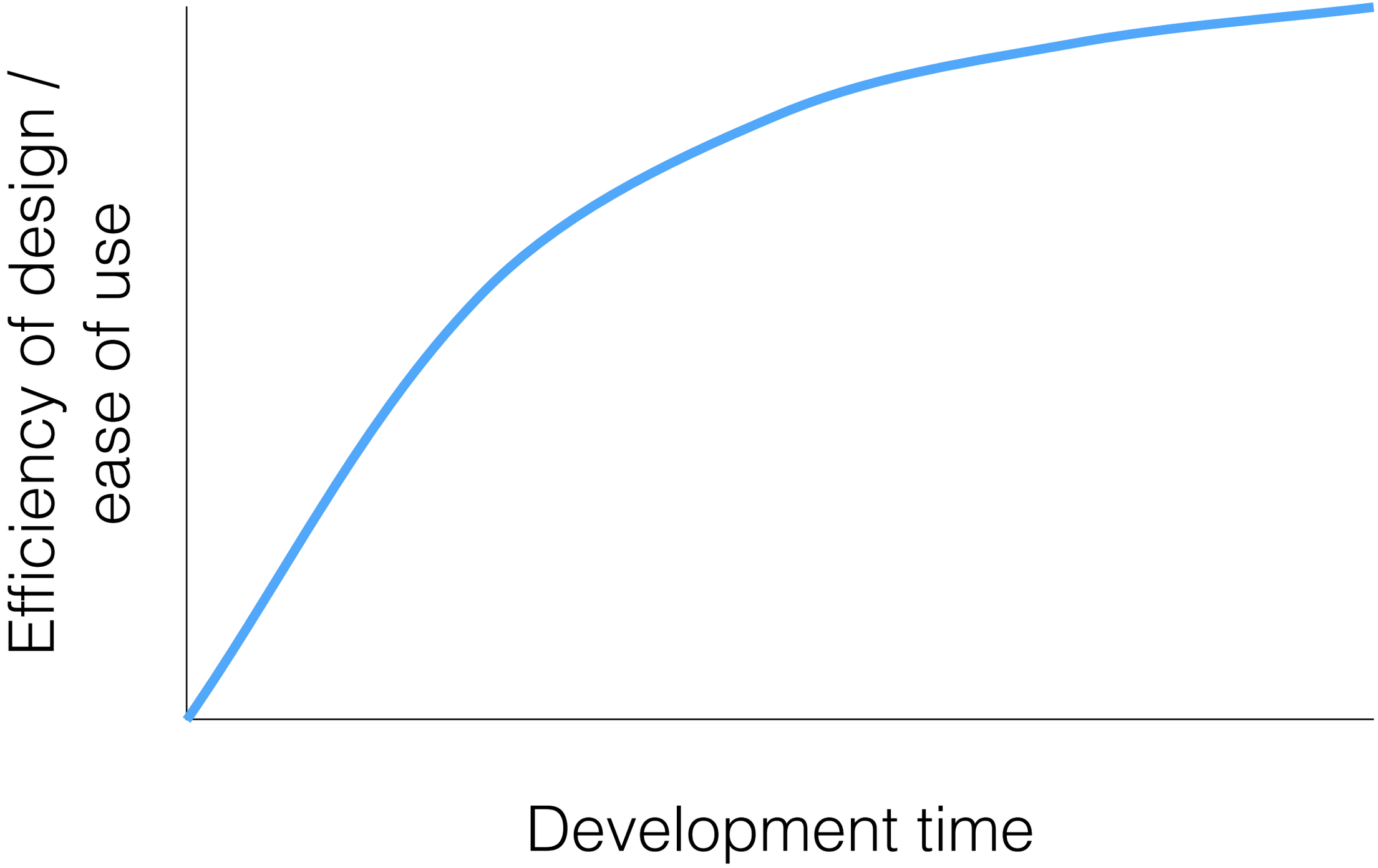
Do I need to think about design

- Yes - up to a point

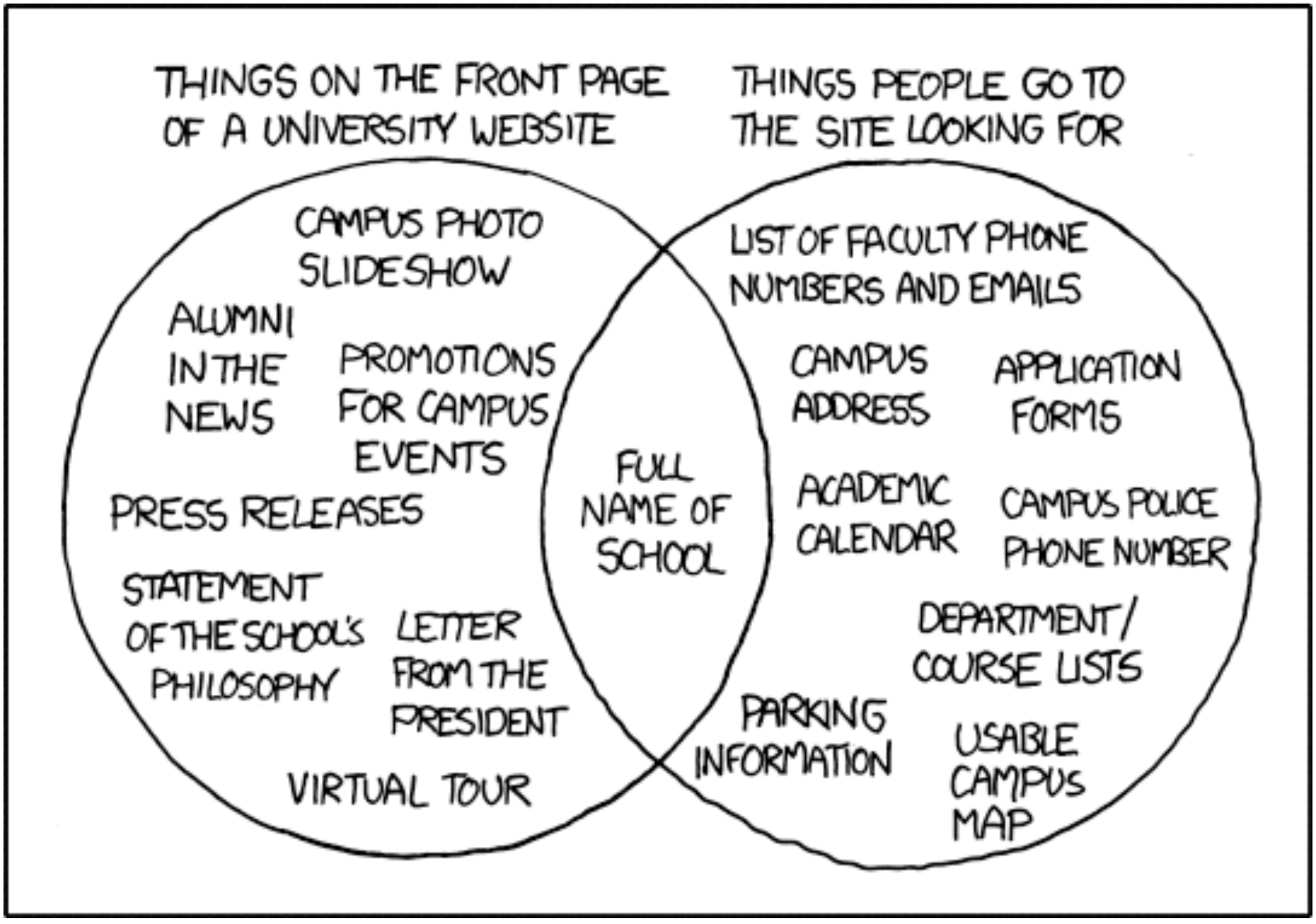
Academic research



Software design



What does your user want?


THINGS ON THE FRONT PAGE OF A UNIVERSITY WEBSITE
THINGS PEOPLE GO TO THE SITE LOOKING FOR
CAMPUS PHOTO SLIDESHOW
ALUMNI IN THE NEWS
PROMOTIONS FOR CAMPUS EVENTS
PRESS RELEASES
STATEMENT OF THE SCHOOL'S PHILOSOPHY
LETTER FROM THE PRESIDENT
VIRTUAL TOUR
FULL NAME OF SCHOOL
LIST OF FACULTY PHONE NUMBERS AND EMAILS
CAMPUS ADDRESS
APPLICATION FORMS
ACADEMIC CALENDAR
CAMPUS POLICE PHONE NUMBER
DEPARTMENT/ COURSE LISTS
PARKING INFORMATION
USABLE CAMPUS MAP

xkcd

80/20

- Other people will use only some of your software's features
    - "80%" used less often
        - often the most interesting and challenging areas
        - should be the focus of your **research** effort
    - "20%" used most often
        - often operations that are "routine" or "solved"
        - should be the focus of your **design** effort
- It's not worth spending much design effort on little-used features, even if they are the most interesting

Example - design for researchers

How to design a GUI

- Think about existing designs - they will be familiar to users
- …but don't feel constrained to follow the same design
- GUI designers (e.g. QT Designer, Guide, …) are useful for prototyping and *small, simple applications*
- … but they are difficult to maintain and produce poor code
- Better to use a good GUI toolkit and layout manager.

Pulmonary Toolkit 0.5

PATIENT

OXF0121

OXF0131

LINKED SERIES

XE> MR COR (7)

MR> MR COR (7)

CT> CT 0.625mm (551)

SERIES

MR 3-pl LOC / e-1 (15)
MR ventilation / e-1 (14)
MR COR FMSPGR / e-1 (14)
MR COR FMSPGR / e-1 (14)
MR 3-pl LOC / e+1 (15)
MR diffusion 4 b / e+1 (28)
MR COR FMSPGR / e+1 (7)
MR 3D Saved State - (1)
MR 3-pl LOC / e+1 (15)
MR diffusion 2 b / e+1 (14)
MR COR FMSPGR / e+1 (7)
MR COR FMSPGR / e+1 (7)
CT 0.625mm Chest / CT (551)

OXF0131

Lobes - 0.625mm Chest

View | Segment | Correct | Analyse

Lung preset | Bone preset | Soft tissue preset | Image preset

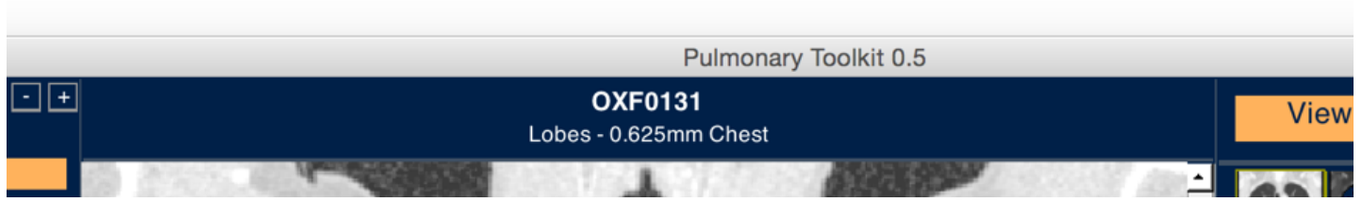
Window: 1600 | Level: -600

Hide Image | Hide Overlay | Overlay Transparency: 50 | Developer tools

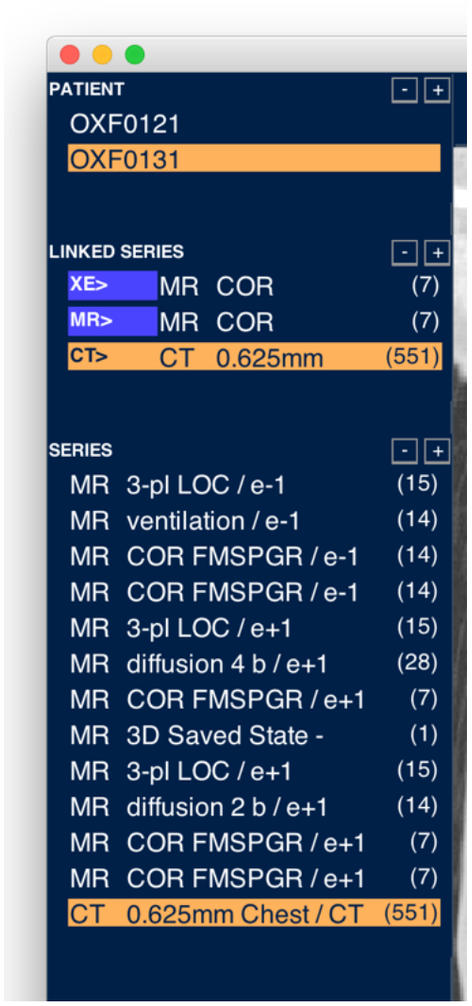
X:360 Y:185 Z:56 I:1089 HU:65 O:0

Patient Browser | Import Data | Delete dataset | Coronal View | Sagittal View | Axial View | 3D | Show Markers | Window / Level | Cine | Pan | Zoom | Correct | Capture | Export Volume | Export Overlay

Simple design elements



Use of whitespace to highlight patient name

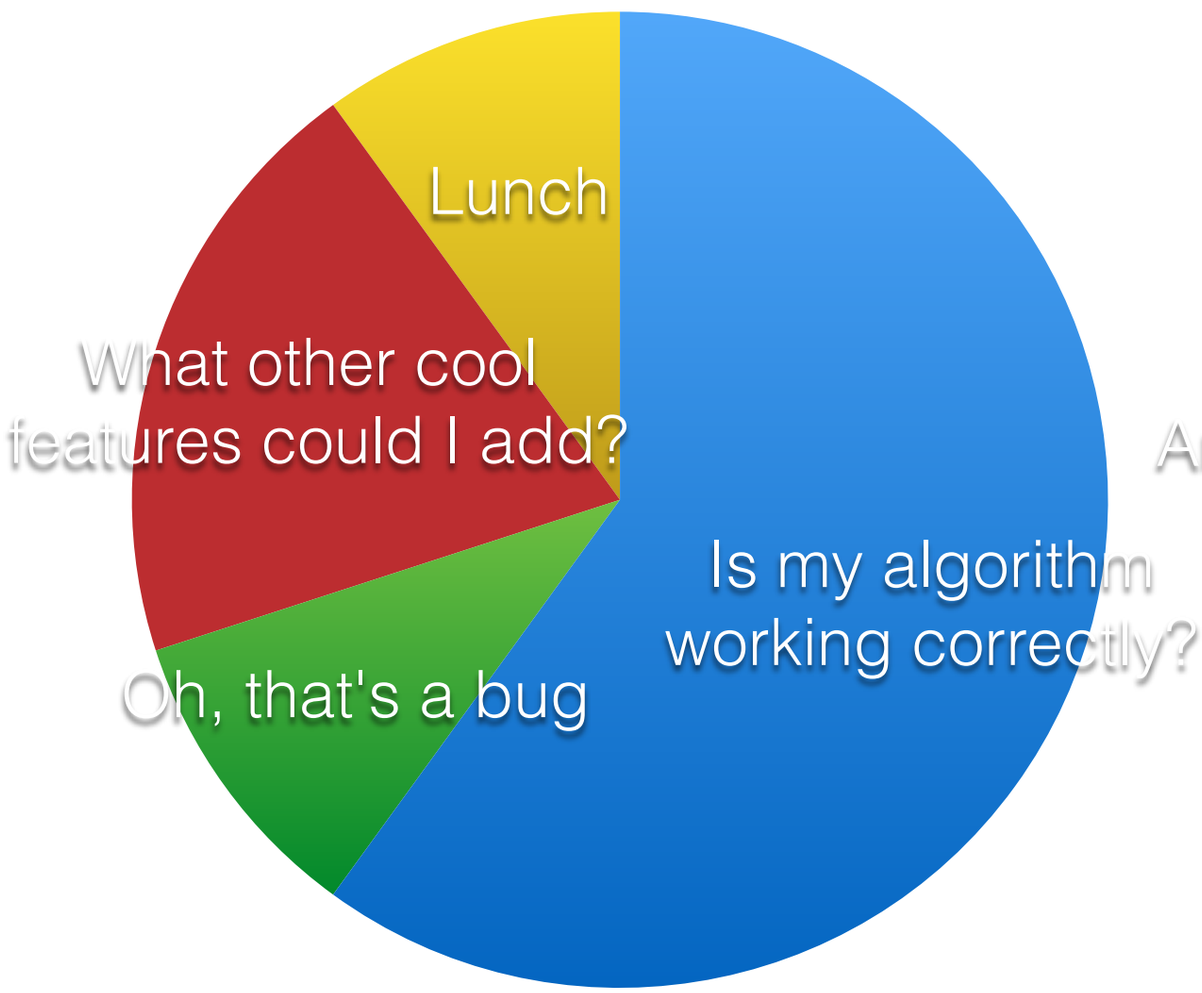


Quick load of different datasets

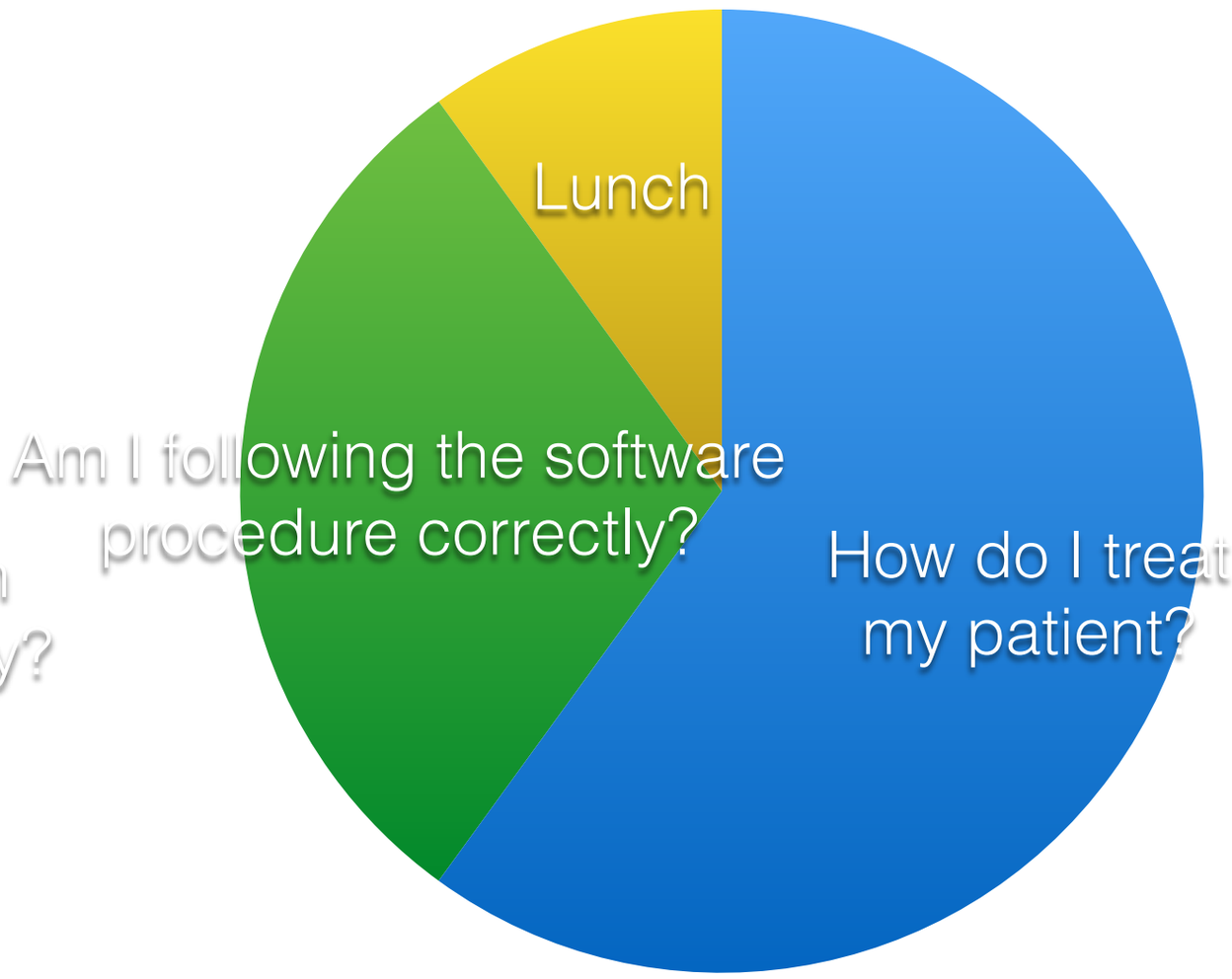
Labeled icons, current tool highlighted

Thoughts when using medical software

What software engineers are thinking about



What clinicians are thinking about



- What someone else wants from your software will be very different from what you want from it.
- They will usually have a very specific purpose in mind and will focus their attention on that.

Drew T, Võ ML, Wolfe JM,, Psychol Sci. 24(9):1848-53, 2013.

- Don’t expect your users to validate their results in the same way you do!

- Clinical users are happy to follow complex workflows but they are not going to think much about how your algorithm works and how to check it is doing the right thing.

- If you need the user to perform some kind of validation (e.g. visual checks on a segmentation result), you need to make this absolutely explicit.

Icons

Use Water

Still Use Water

What

No don't put your hands in

Don't wash ever

Just use 30

Why

Don't wash Tennis balls

Stop it

NO TENNIS BALLS

Triangle

No Triangle

This thing

Less of this

No Idea

The myth of intuition

- Visual elements such as icons rarely help to explain the function of software…

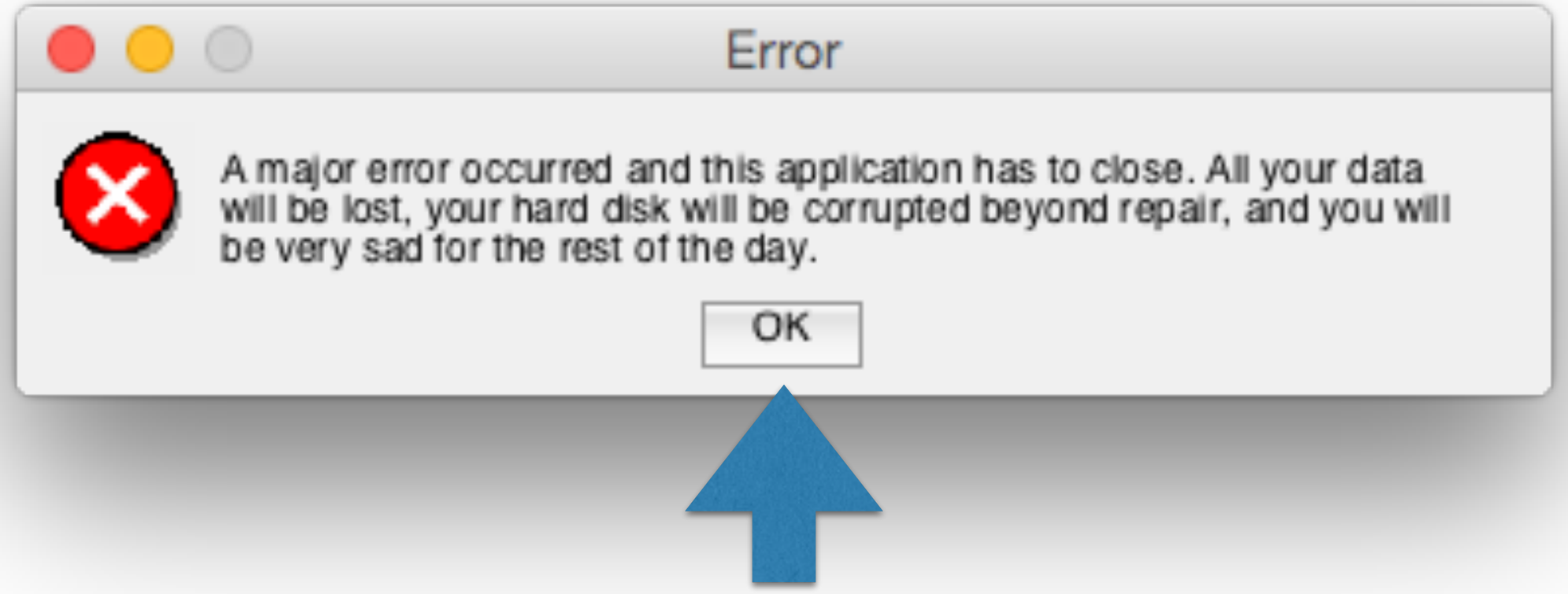
- …but they do provide useful visual clues for performing tasks that **the user has already learned.**

- Most software is not intuitive. It only feels intuitive because you have already learned how to use it.

- Design can’t remove the learning element, but it can make it easier.

- Even with good design, you still need to teach someone how to use your software. The best was to do this is in person.

Don’t annoy your users



This is not OK!

At the very least, change the button to “Close”

Error messages

- Only include information that is *useful to the user.*
- Tell them what they need to do to fix the error.
- Don’t include any details of the error. Debugging information and stack traces should be written to a log file

Do you want to exit the program?

Yes

Are you absolutely sure you want to exit the program?

Seriously, do you really REALLY want to exit? Are you sure you know what you're doing?

Yes

No

Cancel

Don’t ask questions!

- Do you really need to ask the user a question?
- In most cases the answer is no. It is usually possible to “design away” the question. For example:
    - Automatically save results instead of asking the user;
    - Store state between sessions so closing the application won’t lose any data;
    - Save results to a sensible, consistent location instead of prompting the user for a directory.
- If you can remove all questions, then your algorithm can be automated and scripted.

Optimisation

“More computing sins are committed in the name of efficiency (without necessarily achieving it) than for any other single reason - including blind stupidity.”

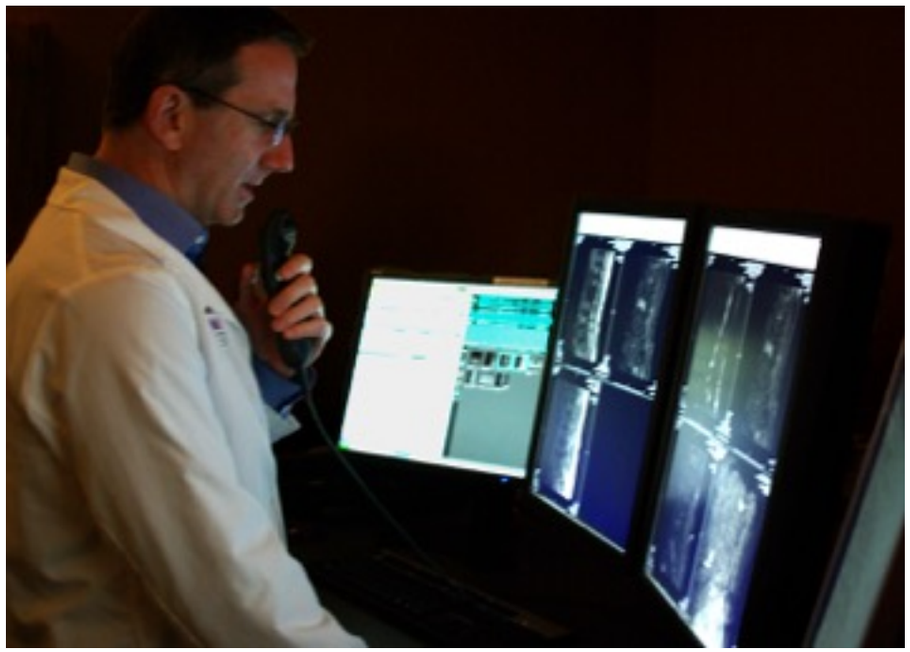
– William Wulf, Proc. 25th National ACM Conf., pp. 791-97, 1972.

“We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil.”

– Donald E. Knuth, ACM Journal Computing Surveys 6(4): 268, 1974.

Case study:

Faster loading



Radiologist launches software

Radiologist waits

Radiologist uses application

How do we reduce this wait time?

Application

Application starts (10s)

Fetch data from hospital server (30s)

Load data into application (30s)

Process and visualise data (5s)

Image credit: Radiologist in San Diego CA 2010 by Zackstarr

Look at the whole picture…

Radiologist selects next dataset from worklist (5s)

Radiologist examines dataset using other viewer (100s)

Radiologist launches software

Radiologist waits

Radiologist uses application

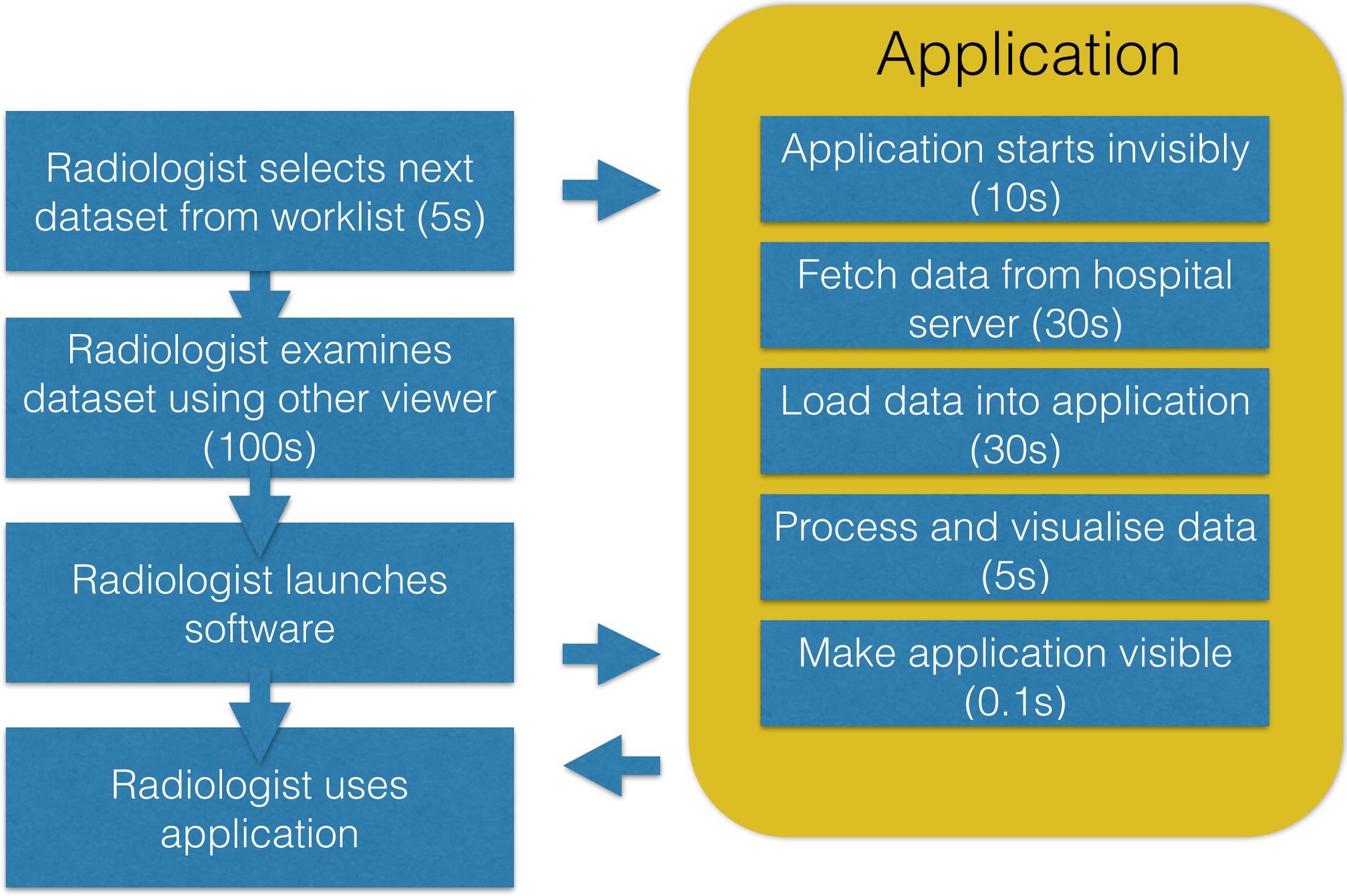
Application

Application starts (10s)

Fetch data from hospital server (30s)

Load data into application (30s)

Process and visualise data (5s)



Run application invisibly in background, resulting in instant response - *preprocessing*

99.9% *effective* speedup with zero code optimisation

- Good code architecture and good design are the best ways to good performance.

- Don’t optimise unless you absolutely have to.

- If you really do need to optimise, use a profiling tool - code bottlenecks often aren’t where you expect!

- If you have to sacrifice design or code quality for performance reasons, you are probably approaching the problem in the wrong way.

Commercialisation

- Research funding will usually only take you to a prototyping or clinical trial stage.

- If you want to see your software in widespread clinical use, the typical route is creating a spinout.

Startup company

Clinical partners

Patents

Commercialisation

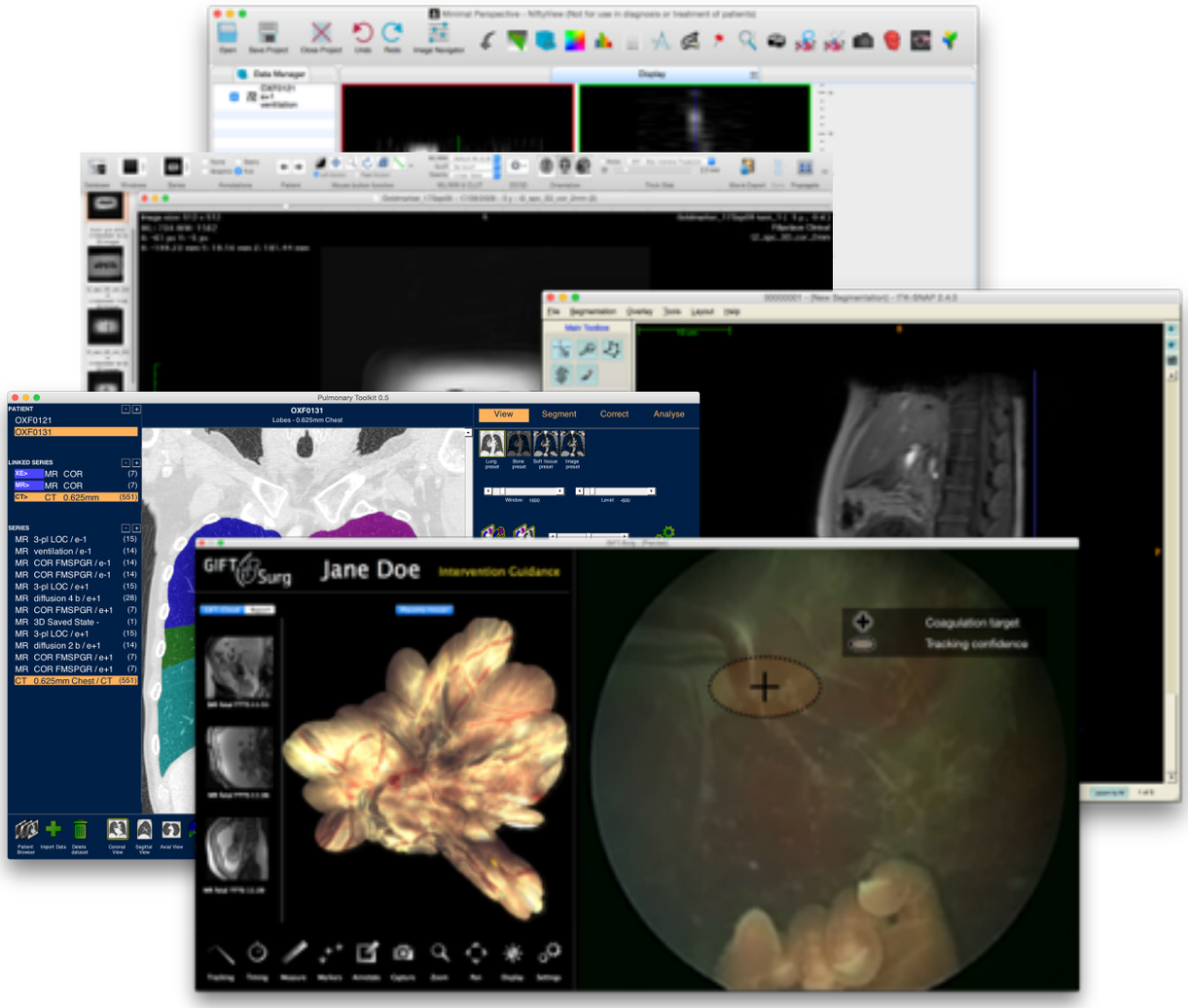
Funding

Your code

Regulatory approval

Final thoughts

- Don’t try and build an application that does everything. Focus on the “killer feature” and design it well.
- Think about who will use your software. Not hypothetical future users, but people you know and work with.
- Understand their use-case, from beginning to end.
- Watch people using your software. Think of easy ways you can make it more efficient for their task in hand.
- Don’t over-design or over-optimise.
- Keep your code clean and modular. This will pay dividends in the future.



Thanks

t.doel@ucl.ac.uk