[Game] Programming I Didn't Learn In School

presented by

Anton Gerdelan
Trinity College Dublin

<gerdela@scss.tcd.ie>
antongerdelan.net

asked to do a talk - i specialise in graphics programming, but i know BTH already has very good graphics programmers.
know students have group game projects to work on
have been working on a barbarian video game in spare time for ~5 years - let's do it on programming
i've been coding ~13 years now and in the last 3 or so i've been realising that all my code has been grossly over-complicated
we are taught to overcomplicate things
few dissenting voices
but i've lately seen some talks by Mike Acton on code design that were pretty neat, and some Data Oriented Design things - really the opposite of what we teach people at university
lots of debates at work at lunch time about design of programmes
let's talk about that - and i'm going to shoe-horn in some Conan the Barbarian quotes
"if i could go back in time to when i was at college - i'd tell myself …"

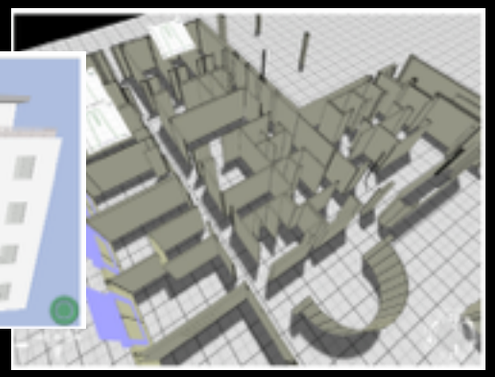
me

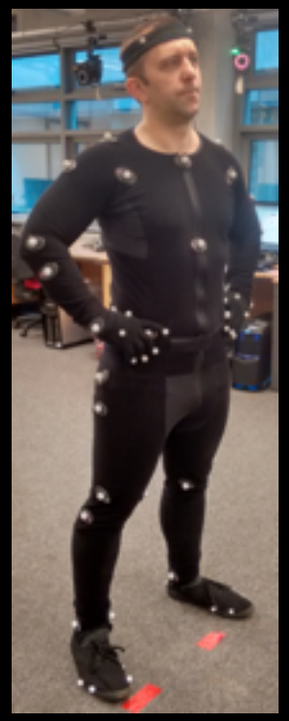
- computer graphics research, Trinity College Dublin, Ireland
- lectured graphics course at Trinity College
- learned shader programming whilst working with Stefan Petersson here
- lots of coding support for graphics courses

i worked here for a year, and really liked shader programming - very challenging. have been doing that since 2011 or so.
published an e-book of tutorials that did surprisingly well on amazon
it was based on experience i got here

research projects

- AI behaviour. Steering. Vehicles. Fuzzy/GA
- WebGL architectural models
- attempting: realistic human rendering / Unreal engine
- motion capture. animation models.
- perceptual experiments



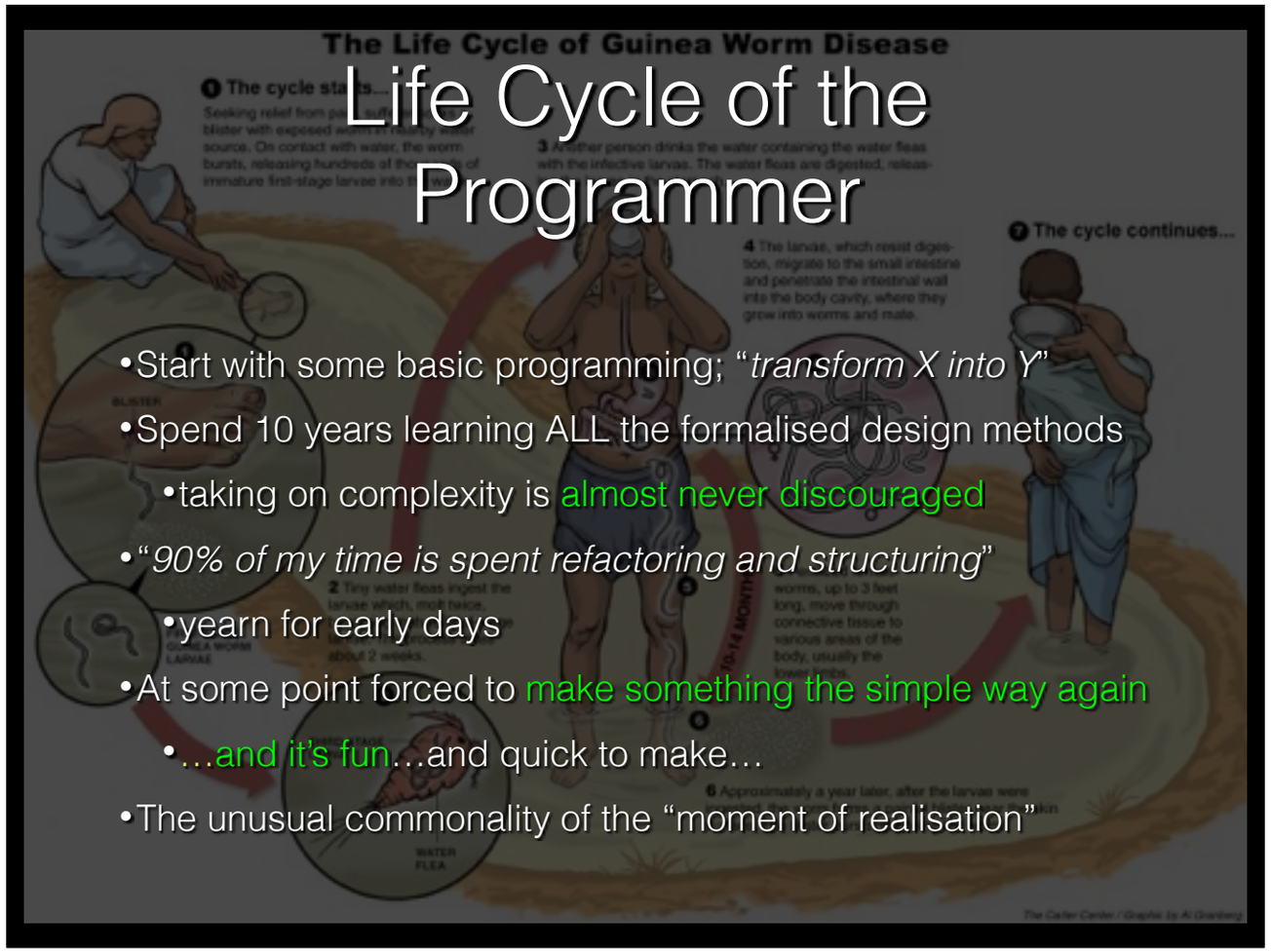


i did my PhD thesis on AI topics - mostly steering behaviour for vehicles, characters in crowds etc. fuzzy logic and some evolutionary algorithms. utterly sick of that!

since working at BTH i’ve optimised my career to work on graphics - back at Trinity College working with WebGL architectural models for 2 years. lots of spare time to develop skills i wanted.

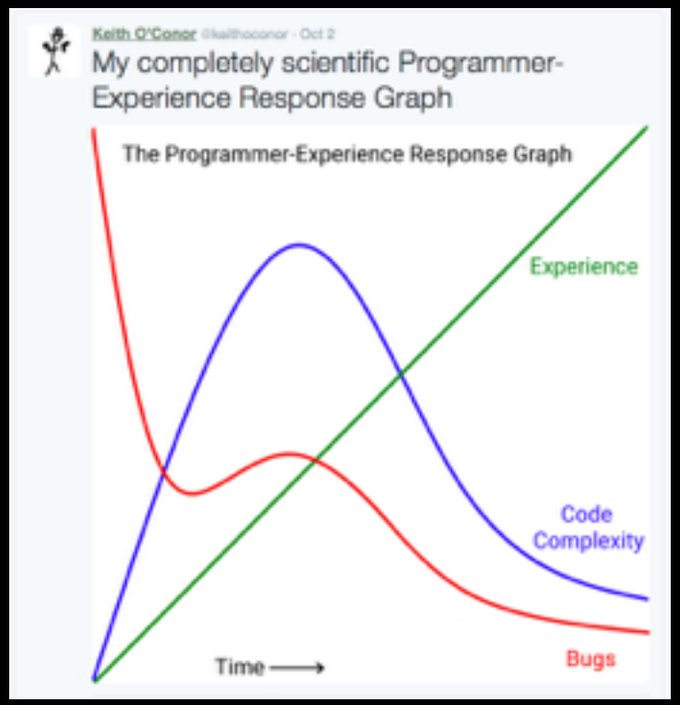
currently with the graphics group “GV2” (graphics/vision/visualisation)
where they have a motion capture lab. i was volunteered to do animations for an experiment here. let me know if you’re visiting dublin and i’ll show you around.

Life Cycle of the Programmer



- Start with some basic programming; "*transform X into Y*"
- Spend 10 years learning ALL the formalised design methods
  - taking on complexity is almost never discouraged
- "*90% of my time is spent refactoring and structuring*"
  - yearn for early days
- At some point forced to make something the simple way again
  - …and it's fun…and quick to make…
- The unusual commonality of the "moment of realisation"

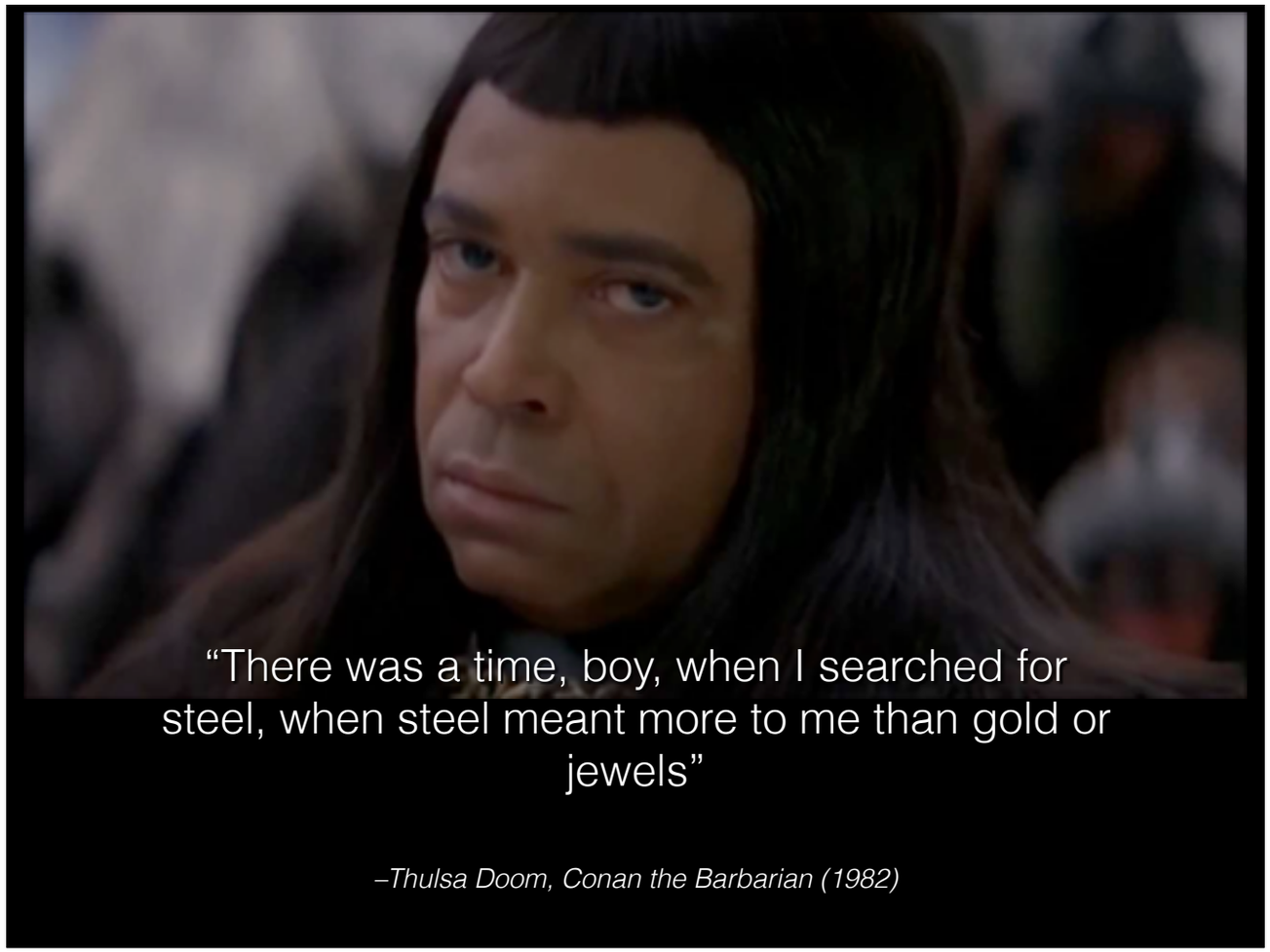
so i was happy being the crazy guy that rejects all the complicated stuff and goes back to basics but it turns out there's loads of us - it seems to be pretty common actually
have been watching
teaching academic dogma versus real world practical ideas
taxonomies, ontologies, abstractions, relations to calculus - interesting academic stuff!
time constraints, performance constraints



(tech. lead Ubisoft Montreal)

has a great blog with lots of advice for aspiring game programmers fragmentbuffer.com

saw this after i made my slides and thought it was spot-on!
keith is a TCD graduate who has come to talk to the students several times about optimisation and writing code that fits neatly into cache lines from his experiences in the game industry - great website/blog



so i’m wearing my Thulsa Doom t-shirt and i have a very tenuous link to the talk theme [recite scene in james earl jones voice]

this idea of an **obsession with a philosophical idea** - and he’s condescending to Conan that he’s surpassed the Riddle of Steel that “Ah, it must have been when I was younger”

Been there, done that, seen the elephant.
In Ireland they would call this being “**given to notions**”

Why is complex bad?

- Takes too long to write (and read)
- Refactoring and the fallacy of the ducks
- Does not respect the computer
- Unhelpful abstraction
- We get very carried away with notions

worst transgression/sin: games take a **very** long time to write - must be very very quick producing working code
or — **expensive** and **demoralising** and **unfinished**
psychological shortcomings - **feedback** from **ordering things for order's sake** alone "to put one's ducks in a row"
actually part of why we enjoy coding at all is this feedback from organising and ordering and sorting
shader programming at BTH re-introduced me to code that respects hardware reality. and boy does graphics code take a long time to write - no room at all for wasting time on conventions
cookie-cutter design patterns and UML layouts that often don't fit the problem well - very few specialised cases where these things enhance productivity at all. must know from experience. but we find ourselves, like good little soldiers, dutifully applying them all, to everything we make anyway.

Quandary

- We change our style ideas every year or two
- I *could* present my “this year” ideas
- Another opinionated rant from an academic about programming games?

what can I do to give this more value?

Interview Prompts

"What would you tell yourself about programming if you could go back in time?"

"What advice can you give about keeping code simple or working expediently?"

"What are typical mistakes that new graduates make, that they would benefit from hearing now?"

"Is there one reference/book/person that you recommend students read or follow?"

hey why not interview some really experienced game programmers?

Mini Interview 1

Niklas Lundberg

Game engine programmer
Avalanche Studios
Stockholm

just finished the Mad Max game of the movie
Just Cause series

“Don’t bother with object oriented programming, it's not helpful”

on going back in time and giving yourself advice

“Try to keep functions having 1 task,
not multiple
(depending on extra passed in
booleans etc.)”

on keeping it simple

“Learn to program a real machine, not a virtual machine.
e.g. use C, not java”

mistakes — hardware

“Don’t keep global state.
Functions should have all
inputs they need”

opengl…ahem. anyone who’s programmed with a global binding model will know how painful this is

- “watch all videos in this series:

  https://handmadehero.org

- and don't proceed to the next until you understand each episode 100%

- requires basic C knowledge

- has a mini course for that too, but you will need some more”

Mini Interview 2

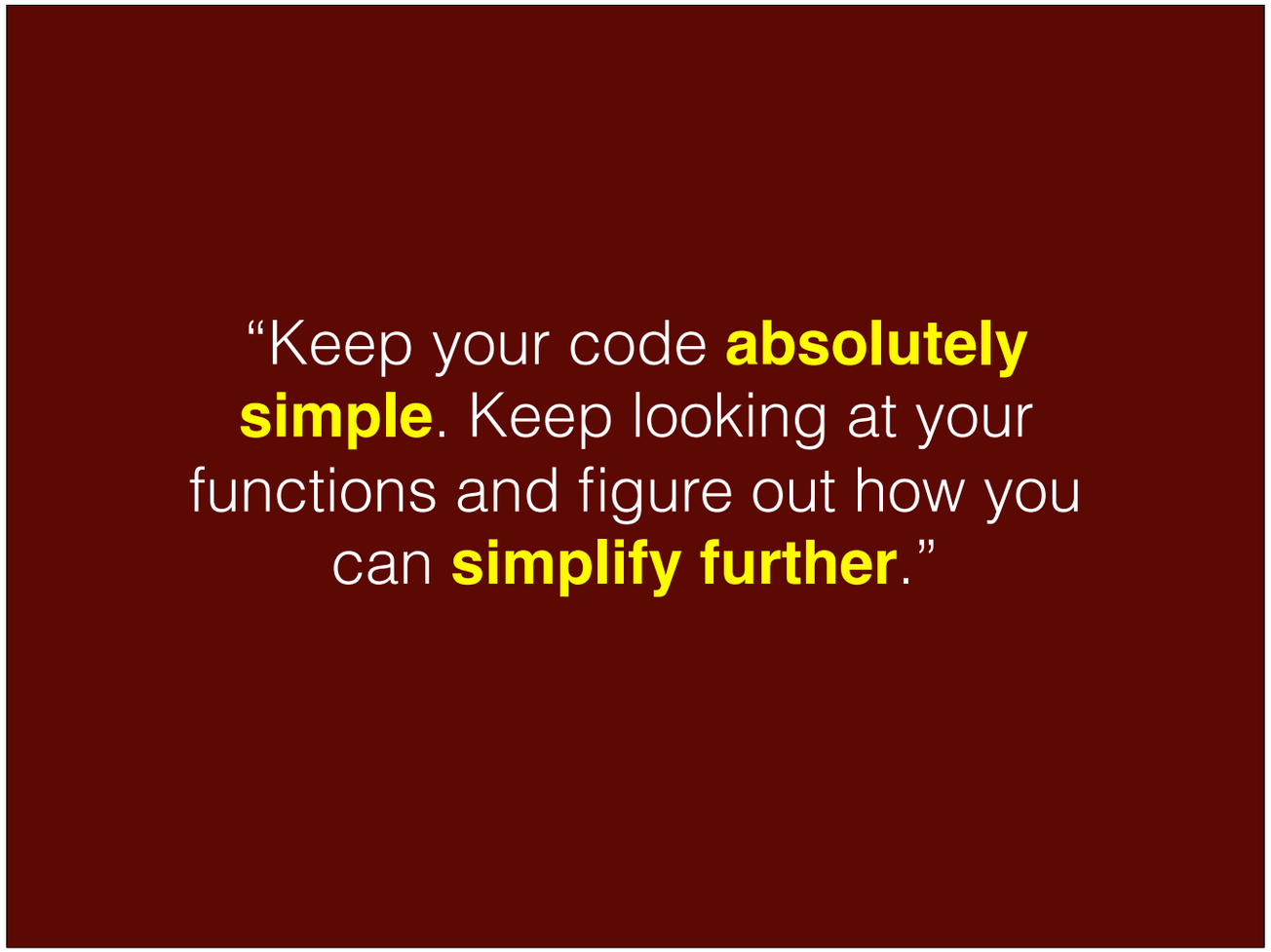
John Romero

i saw brenda and john romero do a talk/panel at Dublin Institute of Technology. very willing to share their experience to students.
fan pic

some different remarks to previous interview:

“Object-oriented programming is really great, but only if you use the bare basics of what it’s great at: encapsulation and inheritance. Stay away from polymorphism, multiple inheritance, and other advanced features because they will get you in big trouble when you try to put your game on multiple platforms – not all compilers treat your code the same.”

“Learning to program in C is really important and it’s also tedious. If you’re going to be an engine programmer, you better know C and C++. If you’re a gameplay programmer, you can get away with a high-level language like Lua, Python, Java, etc. – just know that you’re limiting your career options by not knowing how things work under the hood. You will make more money if you know more, it’s a simple fact.”

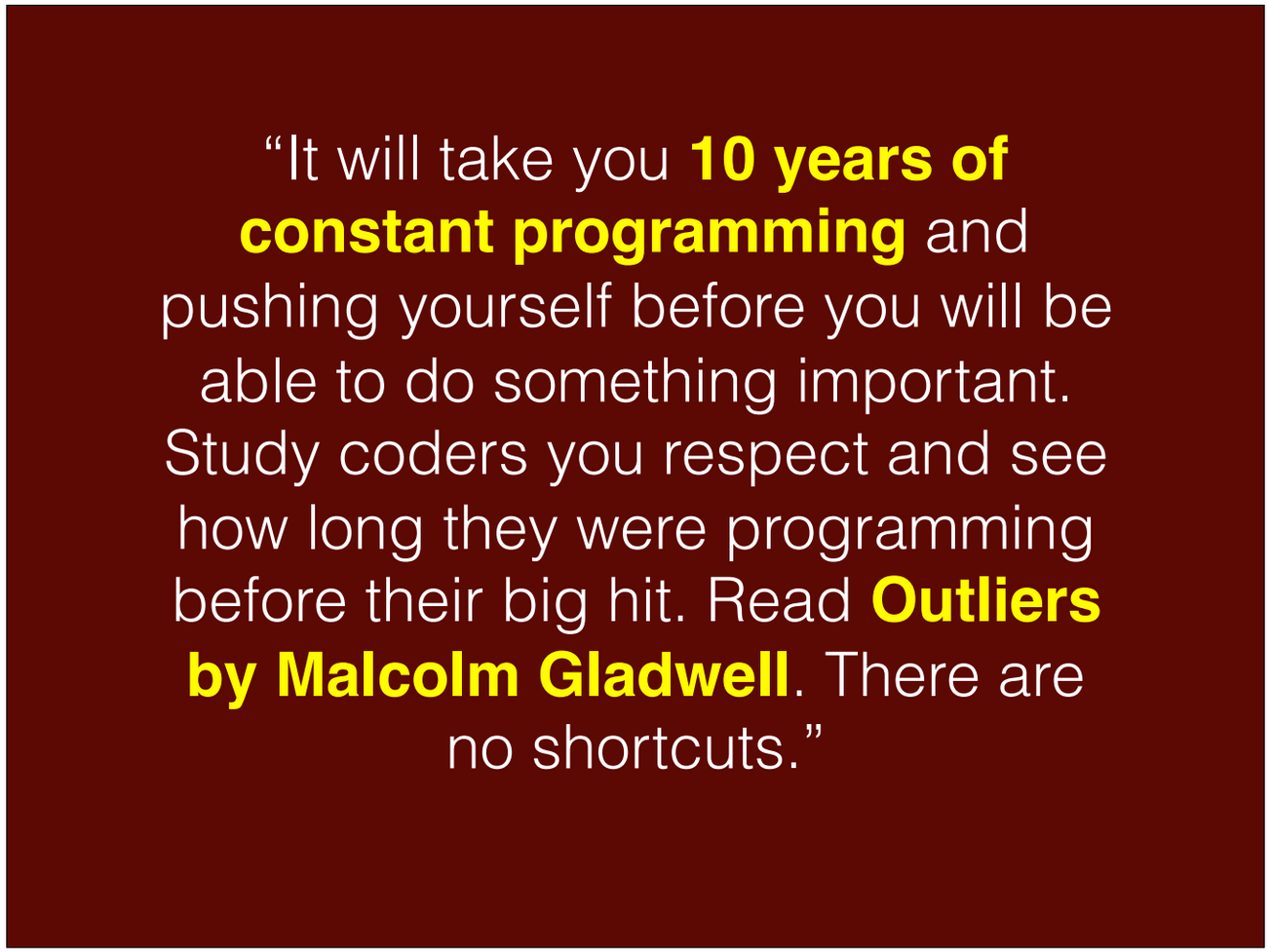


i hear similar opinions from many really great programmers - it’s satisfying like finding a mathematical function from a pattern.
i think a lot nicer/more productive kind of feedback buzz than most ‘refactoring’.

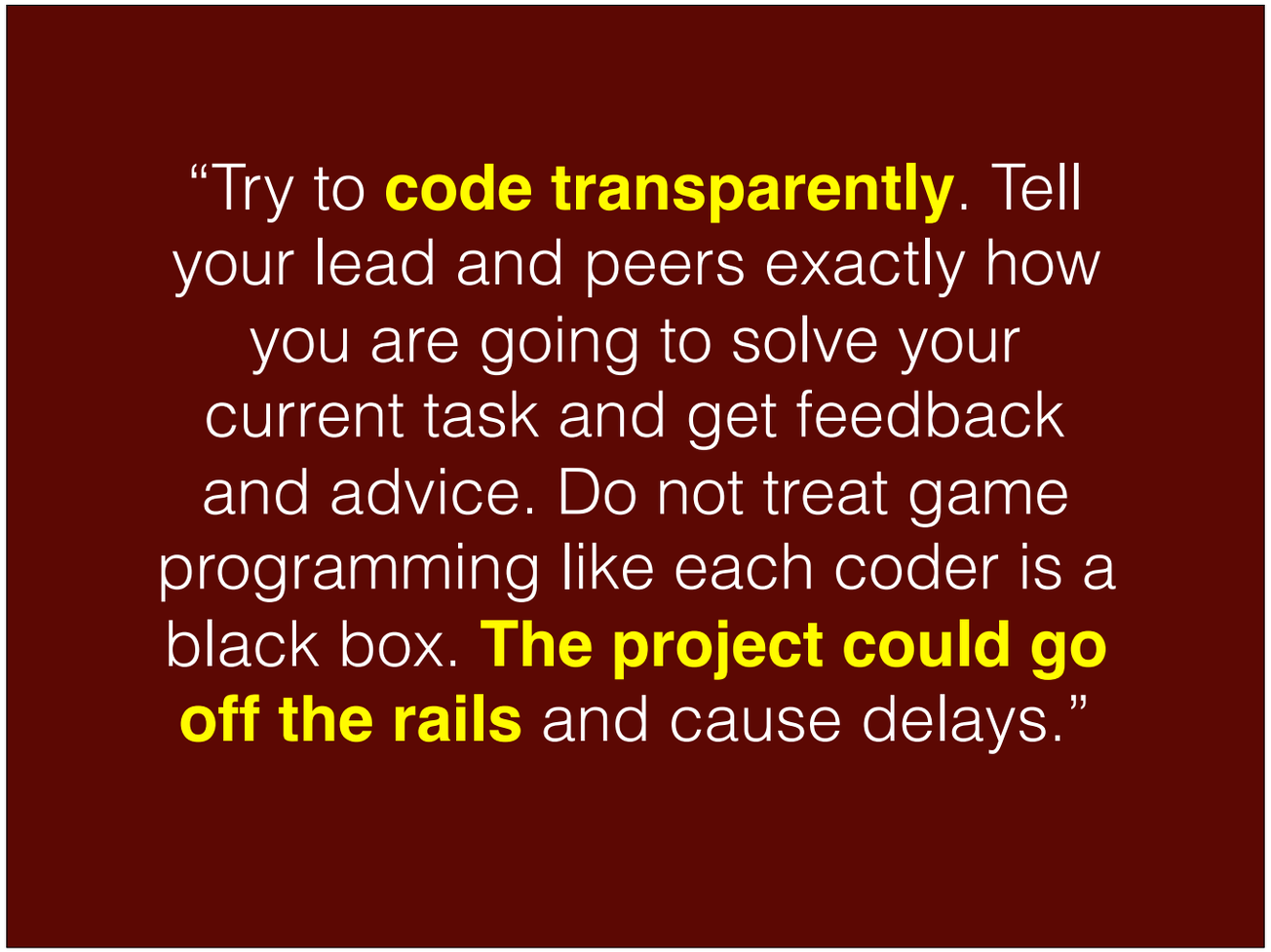
can we harness this feedback we get from ordering and turn it to simplifying instead? more productive

"Write your code **for this game only** – not for a future game. You're going to be writing new code later because you'll be smarter."

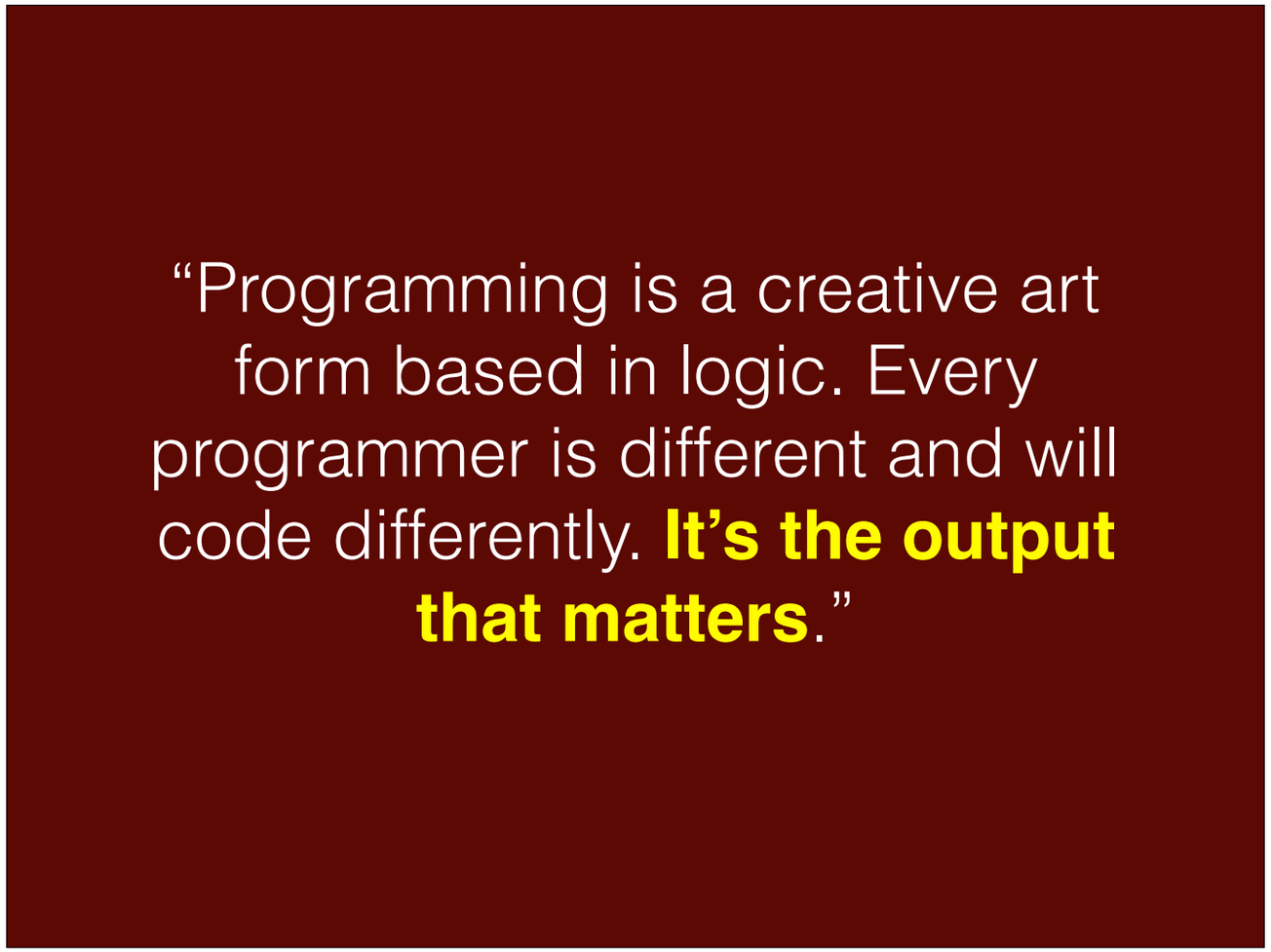
something we do really badly - making general solutions for short term specific problems
although i note it worked out okay for Id software licensing their specific-game engines to other games



probably my favourite point - john talked about this in DIT so I had to ask him more about this because it kind of shocked the game students there who you could feel were ready to go out and strike it rich immediately after a 3 year course or so
masters of doom - surprising - early 20-somethings actually had >10 years experience each and lots of unusual opportunities (made or given) to work before that
i’ve read Outliers - thesis is that 10k hours experience and makes the expert. lots of insightful examinations of software wiz-kids; bill gates etc.



i like this point because it actually mentions cooperation - something university is not that interested in, but industry certainly is. something to think about in your group projects?



thinking about how this sits with the previous point - perhaps we musn’t force everyone to conform to style
it’s very easy to start furious debates with programmers about silly things that disappear when you hit “compile”. where to put brackets etc.

“If you’re making a game with a small team, don’t use GIT – use SVN. Try not to branch. Always keep your code current as often as possible.
Everyone should be able to make a full build and run the game at any time.”

great thing about git/hg etc is the web services — much better than sourceforge with svn was
i feel like we still don’t have the version control system we deserve - they still feel like tools made in the ‘70s or ‘80s to me - not seamless or intuitive like google drive or dropbox, and why not?
takes far too long to learn.

“Programmers should **code as if the QA team does not exist**. When you find a bug, **fix it immediately**. Do not code further. You risk your codebase **depending on that bug**. id Software did not have a QA team before I left after Quake 1.”

quality assurance
one thing we tend to do is build up a big bug list/tracker to ‘solve later’
we have a fear that if we spend time on polish early it might get thrown out for another feature and therefore wasted time

“what would you tell yourself about programming if you could go back in time?”

> “There's a saying that making a game is like **doing major reconstruction work on your airplane while you're flying it**; I heavily recommend having an old airplane needing reconstruction in the first place!”

* what are your key structure and design principles for making good game software _expediently_ ?

"I see some sense in the classical *one class per real-world concept* OOP design style in the highest levels of gameplay code, **e.g. having one object of class "Unit" for each, uhm, unit in the game**, and where performance is the least concern."

* what is your opinion of the usefulness of OOP, design patterns, UML, other formal design conventions?

“I see some value in the idea of "design patterns" as a common language; we say *let's use the reasons pattern* and everyone knows what we're talking about here.

I don't get the **religious attitude towards the book**, and the treating of its list of patterns as end-all, be-all - something I'm sure **the authors never intended**.”

* what is your opinion of the usefulness of OOP, design patterns, UML, other formal design conventions?

“I never got the point of UML. **Pick a high-enough level language and you won't need diagrams** - you'll fit the "big picture" on a screen, then zoom as appropriate.”

* what is your opinion of the usefulness of OOP, design patterns, UML, other formal design conventions?

"A modern game contains many different subsystems, and what is true of code that is executed **once per (one of ten thousand) object per frame** is not as applicable of code that is executed **once in a few seconds per human player**.

Writing code **carefully considering the hardware** is very rewarding, and **very much necessary in the first case**; unfortunately, it's also much harder, and working via **abstractions can be forgiven in the second**."

* how much do you really need to consider the computer's hardware when writing code i.e. is the data-oriented stuff more important than abstraction?

> “**The first thing new recruits need to unlearn is the love for their own code**. New programmers love nothing more than to produce code - pages and pages of elaborate, complex code. But **code is a liability, not an asset**. The job of a programmer is to think first, and to solve problems of his customers (e.g. designers and artists in a game team) second. Not to produce code. Programmers need to learn to love deleting code more than they love writing new code. Programmers need to learn to let their code go and **mercilessly delete and simplify** when a better, or simpler solution presents itself.”

* what are some learned bad habits that you see all the time in new game programmers?

“I think an important habit to have is to resist the temptation to abstract and create "general solutions" the first time you encounter a problem. It is better to implement simple, concrete solutions to the first and even the second occurrence of a similar problem. **Only the third time you know enough about the problem to think of generalising** - and even then you can't be sure you've seen it all.”

* what are some habits or code structure principles of excellent programmers?

“Anyone with a desire to delve in engine code should get a good understanding of C, a decent understanding of C++, and a cursory understanding of the assembly language of the machine they're targeting.”

* what languages should games programming students concentrate on?

  “You won't be writing much, or any, assembler, but you'll need to look at the disassembly when debugging hairier problems more often than you'd like. Advanced mastery of C++ is best left for your first industry job, depending on the exact subset and dialect (11, 14 etc.) the particular C++ shop uses.”

“Nowadays it's hard to evade JavaScript - it's not a very good language, but it's ubiquitous in large portions of the IT industry. C# is also a good bet, being embedded in Unity and also potentially useful in the real world, if you decide games aren't for you after all.”

* what languages should games programming students concentrate on?

“I like the **Handmade Hero project by Casey Muratori**. It's a series of videos showing how a non-trivial game is made from first principles, without using _any_ middleware or "game engines". It's very instructional, and Casey is a **gifted comedian just wasting his talent as a programming superstar**.”

* what is the best way students can make sure they keep learning the _best_ of practical game programming - people to follow or particular book to read?

“Yes! You know what it is, don't you boy? Shall I tell you? It's the least I can do. Steel isn't strong, boy, flesh is stronger!”

–Thulsa Doom

[recite rest of scene]

challenging **ideals with experience** makes me feel like Thulsa Doom when he gives his answer to the riddle of steel

taught complexity vs learned simplicity