

Offset	Topic
00:17	<ul style="list-style-type: none"> • Intro
02:51	<ul style="list-style-type: none"> • Listener Feedback • Chris on discovery
03:59	<ul style="list-style-type: none"> • Word of the Week: de-rezz • http://catb.org/jargon/html/D/de-rezz.html
05:13	<ul style="list-style-type: none"> • Hacking 101: Complexity • I've talked about complexity many times, haven't really defined • A few relevant definitions <ul style="list-style-type: none"> • Cyclomatic complexity <ul style="list-style-type: none"> • http://en.wikipedia.org/wiki/Cyclomatic_complexity • Based on decision points in program • Draw a graph, each decision is a node • Each edge is the flow of execution to the next decision • Cyclomatic complexity is a way of computing a measurement • Roughly is the density of nodes to edges • Means a program is making a lot or a few decisions relative to just doing work • Computational complexity <ul style="list-style-type: none"> • http://en.wikipedia.org/wiki/Computational_complexity_theory • Deals with the amount of resources needed to compute an algorithm • Also with the difficulty in identifying efficient algorithms • A typical question is what is the relationship to compute time to data input size? • This leads to talking about Big O notation • This is typically something like $O(n)$ or order of n • Simplest case means resources, O, are a simple scalar function of data input, n • Leads to mathematical description <ul style="list-style-type: none"> • Linear or scalar, means direct relation • Logarithmic is a shallow but increasing curve • Polynomial is a rapid, deep curve, like exponential or worse • Exponential is something like resources required is the input times itself • Theory also deals with figuring out if a program finishes • This leads also into just figuring out if a more efficient algorithm exists • Complexity as an emergent phenomenon <ul style="list-style-type: none"> • Start with simple rules, like the syntax of a programming language • Building a system out of those rules quickly becomes non-simple

- Should be able to predict all outcomes from simple start
- May be able to do so, but not faster than just running the systems
- An emergent system reveals unexpected surprises
- Defies a human's limited ability to model and predict
- Other examples include markets, physics, animal swarms
- Why is it important?
 - Cyclomatic complexity
 - Easiest to understand measure of complexity
 - Doesn't necessarily relate directly to performance
 - In my experience, has more to do with readability
 - Humans are bad at keeping long chains of decisions straight
 - Using this measure helps keep code readable
 - Means maintainer can keep manageable amount of code clear in their head
 - Improves odds of them spotting, fixing problems
 - Learn to leverage language tricks like functions, classes, polymorphism, other ways to reduce costs of decisions
 - When source grows to a certain size, provides a good heuristic for breaking up
 - Also can guide functional decomposition
 - If a function has more than about a dozen decision points, want to think about decomposing further
 - Since it is formally computable with a decent parse tree, there are tools for measuring
 - Computational complexity gives a vocabulary
 - Can speak about relative efficiencies of algorithms
 - Also provides a vast pre-existing body of expertise of good vs. bad algorithms
 - For example, bubble sort vs. quick sort
 - Bubble involves repeatedly iterating data and comparing two successive elements, swapping if they are out of order
 - Easiest to arrive at on your own
 - A quick search for bubble sort reveals its complexity is $O(n^2)$ worst case
 - Doing some search on efficient sorts leads to quick sort
 - Relatively efficient, $O(n * \log(n))$
 - On your own, if test data is small, may not realize how poor bubble sort does
 - Conversely, if you can control input, understanding computational complexity allows you to make qualified trade offs
 - Emergence applies as you scale up
 - Simplest programs are relatively predictable
 - Given such and such input, does something expected
 - For very large systems, input sets are huge

Offset

Topic

- May be combinatorial, that is involve many variables whose combinations describe a vast set
- Again, runs up against limits of human mind to retain, parse through
- Being able to address system at different scales helps
- Whole system, versus modules, versus classes, versus functions
- Reader is better able to ignore things that are not pertinent to a given scope
- Does mean you need a more sophisticated language
- Explains why OO is popular, regardless of other questions like dynamic vs. static
- I like to think of complexity as a budget
 - Spending complexity is unavoidable
 - Programs need to make decisions to do work
 - However, if you thinking about it as something you spend or accrue, gives you context for implementation decisions
 - This often flows up to user interface, features
 - Have to bear in mind that a system's complexity is also limited by the user's ability to deal with complexity
 - It is often hard to remove complexity at all levels once it is in
 - Complexity is often a consequence of design decisions, amplified by the implementation decisions that follow
 - Use the least complex design, there is room to add complexity later
 - Start with a complex design, adding more complexity can make a system unmaintainable
 - For example, a customer wants a granular permission system
 - If you choose a simpler, say role based design, easier to implement and understand
 - Can defined, add new roles later
 - If you choose to expose fine grained, per use privileges
 - Harder to implement, more complex UI
 - Calculating who can do what also becomes harder to understand and implement
 - Each addition of a new privilege is a combinatorial addition
- Complexity is single biggest factor at odds with our limits as humans
 - Machines don't care about complexity
 - If you have an infinitely complex program, one that never stops
 - Computer will keep grinding until the heat death of the universe
 - When trying to find, fix a bug, complexity is the most direct driver of cost
 - Simple systems are easiest to understand
 - That makes them easiest to fix
 - Complexity is guaranteed to rise over time in successful programs

<u>Offset</u>	<u>Topic</u>
	<ul style="list-style-type: none">• If you are conservative, you keep your code as maintainable as possible, as long as possible
28:14	<ul style="list-style-type: none">• Outro<ul style="list-style-type: none">• Contact me<ul style="list-style-type: none">• Email to feedback@thecommandline.net• Web site at http://thecommandline.net/• IM to command.line@skype• Listener comment line is 240-949-2638• del.icio.us tag is "for:cmdln"• http://twitter.com/cmdln• I'd like to thank libsyn.com for AAC hosting and Wouter de Bie for MP3 hosting• These notes and the show audio and music are covered by a Creative Commons license<ul style="list-style-type: none">• http://creativecommons.org/licenses/by-nc-sa/3.0/us/• Attribution, non-commercial, share alike