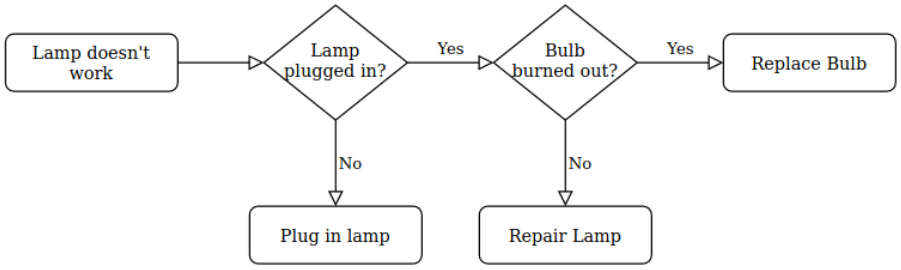
Software Behavioural Design Techniques

Flowcharts



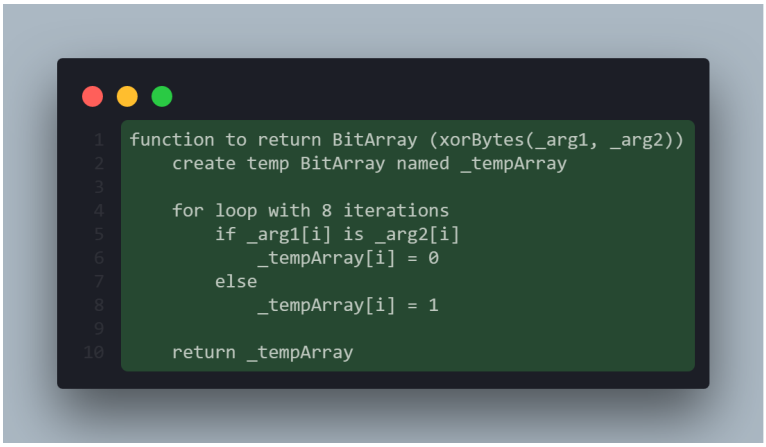
https://www.draw.io/

Flowcharts are useful for software development, showing iteration, conditional statements and branches. There are many types of flowcharts for different purposes, each outlining processes with various amounts of extra information included (Smartdraw.com, n.d.). They always involve having decisions, inputs and outputs, start/stop states, processes and arrows (Tutorialspoint.com, 2020). Decisions and processes are most significant for software, as conditional statements and mathematical or type operations are central to programming as a whole.

SDL diagrams are a form of flowchart that are more specialised for algorithms, using an extra set of symbols to better describe external system references, among other more complex features. These can be useful in describing the interactions between multiple processes using SDL diagrams, along with further describing software processes, although they still come with some of the pitfalls of regular flowcharts (Smartdraw.com, n.d.).

Flowcharts and SDL diagrams are useful for algorithms and methods, however for larger scale or more complex problems they may not be ideal for design and development, as flowcharts can quickly become too large to read and understand easily. While they may be simple, logical and effective for communication, the added complexity that comes from scale quickly reduces the impacts of the benefits afforded by this system (techievp, 2018).

Pseudocode



Pseudocode is another tool that helps to simplify the implementation of software, outlining specific segments of code using simple, universal terminology not specific to any given language. This allows for pseudocode to be written closer to plain English, simplifying understanding and complexity (Techterms.com, 2008).

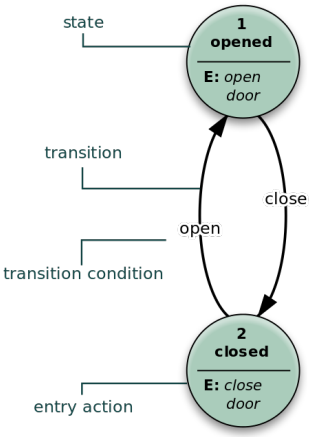
Pseudocode also suffers from some of the same issues as flowcharts, being useful for small code solutions and algorithms, while complicating larger solutions and projects. The lack of a writing standard also complicates larger pseudocode examples, with potential inconsistencies causing potential issues with translation and implementations. This being said, pseudocode is still very useful due to the similarity to real code making development easier overall (Kumar, 2019).

Formal Specification Methods

Formal specifications can be used in software design and development to produce software based on mathematically based techniques. This can be incredibly useful for certain software projects that require mathematical proof in order to be reliable, affording better system stability, robustness and safety (Hierons et al., 2009). This is vital in applications such as software controlling turbine PLCs or aerospace computers, as the robustness and accuracy of the system is key for operation.

This type of design technique can be very costly, with formal specifications being expensive compared to the other options available. As well as this, they can be difficult to understand and apply, requiring analytical skills and mathematical experience to utilise (Nummenmaa et al., 2011). While they may be key for development of certain software, for the majority of development this process is not required, with other design techniques being preferable in terms of cost, time, complexity and scope.

Finite State Machines



(Perhelion, 2015)

Finite State Machines are diagrams that describe the states in a machine with a finite quantity of states, along with how each of the states relate to each other and how they can be traversed. These can be incredibly useful for segments of a program that involve states, describing how each state is reached, assisting development of scripts and modules that can have internal states (Bevilacqua, 2013).

Similar to flowcharts, FSMs can get very complicated with scale, involving many states and state transitions, which can quickly be overwhelming. Additionally, determining the differences between similar states can be difficult, with similar states with only minor differences being difficult to differentiate. FSMs are incredibly easy to convert to code, however, using conditional statements and switch/case statements that recreate the state transitions easily. As well as this, explicit state definitions can help to de-clutter software, depending on implementation (García, 2018).

Extended Finite State Machines

Similar to conventional Finite State Machines, e-FSMs involve a diagram of states and state transitions, however with e-FSMs, there's more detail placed on the operations that take place when a state transition occurs. This covers some of the information that is not shown on a traditional FSM diagram, which instead only outlines conditional statements and not any algorithm usages (Maxemchuk, 2002).

e-FSMs retain the benefits of conventional Finite State Machines, affording very easy development by having explicit definitions for states and state transitions, along with the extended operations that e-FSMs involve. The added complexity from showing the additional actions on state transitions can make e-FSMs harder to read, however the verbosity can be seen as a benefit in spite of this (García, 2018).

Bibliography

Smartdraw.com. (n.d.). *Different Types of Flowcharts and Flowchart Uses.* [online] Available at: https://www.smartdraw.com/flowchart/flowchart-types.htm [Accessed 15 Jan. 2020].

Tutorialspoint.com. (2020). *Flowchart Elements - Tutorialspoint.* [online] Available at: https://www.tutorialspoint.com/programming_methodologies/programming_methodologies_flowchart_elements.htm [Accessed 15 Jan. 2020].

techievp (2018). *Advantages and Disadvantages of Flowchart.* [online] AdvantagesNDisadvantages. Available at: https://www.advantagesndisadvantages.com/advantages-and-disadvantages-of-flowchart.html [Accessed 15 Jan. 2020].

Techterms.com. (2008). *Pseudocode Definition.* [online] Available at: https://techterms.com/definition/pseudocode [Accessed 15 Jan. 2020].

Kumar, B. (2019). *Advantages and Disadvantages of pseudo code – Buildprogrammer.com.* [online] Available at https://www.buildprogrammer.com/advantages-and-disadvantages-of-pseudo-code/ [Accessed 15 Jan. 2020].

Hierons, R., Krause, P., Lüttgen, G., Simons, A., Vilkomir, S., Woodward, M., Zedan, H., Bogdanov, K., Bowen, J., Cleaveland, R., Derrick, J., Dick, J., Gheorghe, M., Harman, M. and Kapoor, K. (2009). Using formal specifications to support testing. ACM Computing Surveys, [online] 41(2). Available at: https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.144.3320 [Accessed 15 Jan. 2020].

Nummenmaa, T., Tiensuu, A., Berki, E., Mikkonen, T., Kuittinen, J. and Kultima, A. (2011). Supporting agile development by facilitating natural user interaction with executable formal specifications. ACM SIGSOFT Software Engineering Notes, [online] 36(4). Available at: https://dl.acm.org/doi/10.1145/1988997.2003643 [Accessed 15 Jan. 2020].

Perhelion (2015). File:Finite state machine example with comments.svg - Wikimedia Commons. [online] Commons.wikimedia.org. Available at: https://commons.wikimedia.org/wiki/File:Finite_state_machine_example_with_comments.svg [Accessed 16 Jan. 2020].

Bevilacqua, F. (2013). Finite-State Machines: Theory and Implementation. [online] Game Development Envato Tuts+. Available at: https://gamedevelopment.tutsplus.com/tutorials/finite-state-machines-theory-and-implementation--gamedev-11867 [Accessed 16 Jan. 2020].

García, J. (2018). Advantages & Disadvantages of Finite State Machines: Switch-cases, C/C++ Pointers & Lookup Tables (Part II). [online] Ubidots Blog. Available at: https://ubidots.com/blog/advantages_and_disadvantages_of_finite_state_machines/ [Accessed 16 Jan. 2020].

Maxemchuk, N. (2002). Reliable multicast with delay guarantees. IEEE Communications Magazine, [online] 40(9), p.6. Available at: https://www.researchgate.net/publication/221198722_A_Reliable_Multicast_Protocol_with_Delay_Guarantees [Accessed 16 Jan. 2020].