*General Safety Overview for Software*

Two primary sides: hardware safety and software safety. Obviously we’re more concerned about software safety, but that’s reliant on interacting with the hardware, receiving safety signals from it, and also relinquishing control to hardware if the software fails.

**Software**

**User security**

* Need some kind of logon system (username, password) to prevent unauthorised access
* Only one logon can be “active” at a time; maybe let others view data, but only one can actually control the hardware?
* Auto log-off if idle for a certain period of time (and experiment isn’t running).

**GUI safety**

* Need to GUI to have safety controls built in
* Make sure there are clear confirmation dialogs for “dangerous” actions
* Make sure information is clearly labelled and presented (i.e. pressure signals)
* Need status of various safety measures & interlock to be clearly displayed

**Network security**

* Obviously need some sort of security on the webserver to prevent any external tampering
* As long as a relatively secure platform is chosen initially, the specifics of this can be implemented later
* Also need to be able to handle network disconnections and interrupts, bad requests, etc

**Input checking**

* Need to have basic argument checking (i.e. don’t let the user input stupid pressure values)
* Only allow inputs & advanced system control if all safety switches are OK and interlock is activated
* Argument checking needs to occur on all levels of hardware (Arduino, Pi, etc)

**Software error handling**

* Would be good to have a decent level of error reporting/error handling from the system
* GUI errors, webserver errors, data processing errors, communications/network errors, timing errors, errors in low-level hardware code, general crashes and freezes…
* During the coding and design of each specific system level (GUI, webserver, low-level code) we can list and define the safety considerations of each – for example, what happens if the server crashes and how the other systems will deal with it. It’s difficult to list specific things without knowing exactly what hardware/software systems we’re working with
* At least the webserver and GUI stuff can be considered early, with low-level interactions added once we know exactly what hardware has been decided upon
* Ensuring communications between systems work well will be key
* Ideally the software system itself shouldn’t generate any errors, but we need to be able to deal with this if it arises (and for debugging).

**Hardware error handling**

* Also need to be able to deal with hardware errors – i.e. if sensors are not reporting proper values, safety catches aren’t showing as activated, etc. To do this the program should know roughly what to expect in terms of the range of sensor inputs.
* May also have to deal with hardware errors where signals are not being reported correctly, but I think this is more for the hardware team to deal with (like if a valve somewhere stops working). Basically, if signals are anomalous then software should stop execution.
* Issues with camera data, network cables and things also come under this

**Emergency response**

* If something does go wrong in hardware the software needs to have some kind of response
* Venting the air and deactivating the pneumatics will probably be required, plus maybe resetting some of the electronic components? There will also be a manual switch to do this.
* Can freeze the experimental controls until everything is OK again

**Data security**

* Need to make sure data is stored safety and can be recovered if there is an accidental shutdown, network interruption or other failure
* Can be stored on external hard drive, with backups downloaded by the user if requested?
* Need to make sure data is transferred to the webserver and client correctly

**Code standards**

* Make sure code is understandable, generally simple and efficient
* Make sure code is well-formatted and commented
* Basically, take the standard precautions so that if multiple people work on the same code they can do it easily (and not stuff it up) and can understand what everything does

**Hardware**

**Software failure**

* If software fails it needs to relinquish control to hardware somehow
* The system must have manual overrides that can be used in place of software (maybe not for complex manipulation, but at least things like venting air and so on).
* There could be some kind of detection mechanism in the hardware that takes over if the software goes mega-bad – like if everything freezes and stops sending signals, a switch could be activated in the hardware
* This is more the job of the hardware team, but we need to communicate with them on exactly how the software controls the system and where they can place hardware overrides, and how they want to deal with software failure
* All we need to worry about is making sure the software can give up control if it breaks, that software can’t do anything too dangerous, and that hardware overrides are in place

**Safeguards**

* All software-controlled systems need their own limiters or locks (such as pressure regulators that won’t go above a certain value)
* Safeguards such as the interlock and pressure valves need to limit software control if they are not activated
* Ideally valves and some electronics in the system could be operated manually as well – valves can be adjusted, electronics can be reset, etc. There should at least be an on-off switch for the system as a whole and some kind of ‘emergency stop’ button which again overrides the software

**Visual safety**

* Would be good to have pressure readouts, LEDs for the safety switches, and other visual aids so a person in the room knows roughly the state of the system
* Could be controlled by software stuff or by hardware switches

**Safety stuff that software doesn’t really have to worry about**

* Electrical safety in terms of wiring, voltages, circuit connections etc
* The box exploding and killing everyone
* The can exploding and killing everyone
* Pneumatics leaking air everywhere
* Structural failures
* Pumps and valves breaking
* Etc. Basically, software should be able to deal with some hardware failure, but a lot of it we can’t really control – as long as software is accurate, safe, deals with errors and can let hardware take over manually if failure occurs, we should be fine

This flowchart is basically nonsense but at least it looks cool.

**Overall Checklist:**

* **User security**
  + Is the software only accessible by authorised people?
  + Is the system safe if operated by two conflicting people at once?
  + Can the system safely be left unattended for a period of time?
* **GUI safety**
  + Do dangerous actions like pressurising require confirmation?
  + Is all information clearly presented and labelled?
  + Is status of safety measure displayed at all times?
* **Network security**
  + Is the webserver secure when accessed by unauthorised sources?
  + Are the server and associated pages secure?
  + Can the webserver deal with communications dropouts?
* **Input checking**
  + Does the GUI check for out-of-range inputs?
  + Does the interface prevent inputs if safety switches are deactivated?
  + Does argument checking occur in all lower levels of software?
* **Error handling**
  + Has the GUI been tested and debugged? Does it work accurately and fulfil requirements in general? Can the GUI deal with errors?
  + Webserver?
  + Data processing algorithms?
  + Lower-level sensors code?
  + Lower-level control code?
  + Can timing errors be tracked and dealt with?
  + Can communications or network errors be tracked and dealt with?
  + BASICALLY DOES EVERYTHING WORK?
  + AND IF IT DOESN’T CAN WE FIND OUT WHY?
* **Data security**
  + Is data stored safely?
  + Is there a backup available in event of failure?
  + Can the user download it if necessary?
* **Code standards**
  + Is your code well-commented and formatted?
  + Is your code a simple, efficient solution for the problem?
  + If someone else has to work on your code at a later date, will it be easy for them to pick it up?
* **Remote control**
  + Are control signals passed accurately through all system levels to all actuators?
  + Does the software’s remote control capability prevent dangerous actions?
  + Are control routines able to deal with and detect unresponsive hardware?
* **Hardware failure response**
  + Does the program know what to expect in terms of inputs? Can it respond if inputs are definitely not correct?
  + Does the program know what to expect in terms of control? Can it do something if elements are not responding?
  + Can software safely stop the experiment if a safety issue is detected with valves, sensors or the interlock?
* **Hardware emergency response**
  + Does the system have an appropriate response for hardware/software failures?
  + Is software able to vent air and deactivate pneumatics safely in an emergency?
  + Can the experimental controls be frozen until the problem passes?
* **Software failure response**
  + If software fails does it relinquish control to hardware?
  + Are manual overrides available that can be used in place of software?
  + Is there any kind of detection mechanism in hardware to check if software is acting and responding appropriately?
  + Are the manual overrides all safe and undamaging to the rest of the system?
  + BASICALLY IF SOFTWARE FAILS IS ANYONE GOING TO DIE?
  + AND IF IT FAILS CAN WE STILL CONTROL HARDWARE TO SOME DEGREE?
* **Safeguards**
  + Do all software-controlled systems have their own limiters and locks?
  + Do safety switches like the interlock prevent pressurisation if they are open?
  + Can everything necessary be operated manually? Is there an on-off switch and an emergency stop button outside of software?
* **Visual safety**
  + Can a person in the room figure out the state of the system from visual feedback?
  + Are there clearly-labelled status indicators for the interlock, electrical systems, valves and pressure?
  + Oh my god there’s a lot of questions?