"OK, we've defined the project objectives. Now it's time to establish the stakeholders."

SENG2011 Project Vampire

Term 3 2019

You are asked to build and verify a management system for a company called *Vampire Pty Ltd* that manages blood supplies. What the company currently uses is paper-based and problems continually arise such as blood going out-of-date before it can be used, running out of blood, poor record keeping and poor donor service.

The company receives blood from medical facilities such as pathology, hospitals and clinics, and also from the public through an organised mobile blood service, popularly known as the bat-mobile, which visits public places such as shopping centres. Donors can register on the system and provide their personal details so that they can give blood when the mobile service is next in the donor's area. Blood collected in this way must be sent to pathology (there may be just one, or many) for screening and testing.

Vampire delivers blood to registered organisations, which include hospitals of course. The company must ensure that there are sufficient supplies to cover all requests. It must also ensure that its supplies are 'fresh': blood that has passed its use-by date must be disposed of.

The service offered by Vampire Pty Ltd is government funded, so you do not need to handle any financial transactions. The company does not deal directly with the public except through the bat-mobile.

You are asked to build a prototype of system that Vampire Pty Ltd can evaluate. The prototype may be stand-alone. The aim is to demonstrate the functionality to a board of executives who, if they approve, would fund full scale development. Most of the executives on the board are not convinced they can trust a software system to manage a critical resource like blood because software has a bad reputation for reliability and lives can be lost if mistakes are made. Verifying that the software is safe will be an important (maybe the most important) aspect of the software development.

First some general comments about what is important.

- **The core business** of Vampire P/L is the handling of blood. System administration facilities such as logging in, handling passwords and general 'admin' functionality are unimportant.
- **Quality of code** is more important than quantity of code. The prototype must demonstrate that a highly reliable and verifiably correct system can be built. One of the aims of your work is to convince the executives that your system is safe. A system that is unreliable, or one that cannot guarantee that 'fresh' blood will be provided, could be a disaster for patients and the business.
  - Focus on the back-end (not the front-end or user-interface). The user-interface may be command line.
  - The algorithms (e.g. searching, sorting, filtering) that you use should be verified.
  - It is critical that out-of-date blood does not get sent to customers.
  - There are also privacy concerns: results of testing blood may be negative, and require feedback to donors.
- **Functional (behavioural) requirements** only need be considered in this work. Performance is a non-functional requirement that is not relevant at this stage, nor are hardware or architectural considerations.

There are various aspects to the system:

- **queries**: what kind of queries should the system handle? A medical facility may enquire what the current levels of a certain blood type are. A member of the public may enquire when and where he/she may give blood, or general enquiries such as what the health requirements are for giving blood.
- **testing** of donated blood occurs al pathology or a hospital. (Vampire does not do testing.)
- **requests** for blood come only from registered medical facilities.
- **deposits** of blood come from pathology. You can assume that the origins of blood (name of donor, when and where it was taken) are available.
- **inventory of supplies** provides information such as type of blood, arrival date, use-by-date, where it came from, and from whom. Other information may be necessary. The level of supplies must always be known of course. If the level falls too low of some type of blood, then a 'warning' should be sent to the company to take action. You may assume that it is not the responsibility of the system itself to take action.
- **donor database** that stores the personal details of donors.

Making a start

The system is too big to model everything, and is presented here in its entirety to give you some space for creativity. Focus on a 'sub-system' where verification (of algorithms) can play an important role. Quite obviously, the more that can go wrong, the greater the role for verification. The sub-system may be a component of the system described above. For example, you do not have to have a bat-mobile in your system, or you may choose to build a system for the bat-mobile alone.

Do not focus on interface features such as the registration of donors or medical institutes, or on data in general, or the GUI. The intial steps you need to make are:

One: Identify 'core business' for your (sub-)system, and describe its functionality in general terms (less than half a page). This is sometimes called an executive summary. The users (you may or may not call them stakeholders) of the system should be identified here.

Two: Write the business and technical requirements. These requirements form the backbone of the project and should be referred to throughout the development and report. The requirments must be prioritised. Keeping track of the progress on each requirement helps immensely in organising the work, in deciding which tasks need to be performed, and in measuring progress.

Most groups attempt to implement too many requirements at first, and are overwhelmed by the amount of work and the complexity. It is better to be 'incremental'. Take a small set of requirements that you consider absolutely essential and build an initial system to implement them first. Then slowly expand the system by adding more requirements, one or two at a time. You can use priorities on the requirements to enforce this approach: first implement the highest priority level requirements, then the next level etc.

Three: Show the interactions between the stakeholders and the system in use-case diagrams.

Four: Draw the Work Breakdown Structure (WBS): a graph showing how the project can be broken down into component builds. The coding is planned and managed using the WBS.

There will be lecture slides on requirements, use-cases and WBSs.

A *template document* (a document skeleton) will be provided to help with reporting. The reporting requirements will be explained in this document.

You may implement your system using any language, but you should attempt to verify the important parts of the code (in Dafny) using pre- and post-conditions, invariants, variants and assertions for example. The verified Dafny code may be translated by hand to your implementation language. At the end of the course you should submit your Dafny code, and it should be compatable with CSE Dafny 1.9.7 for assessment purposes.