Project Assignment 3

Steve Mazza

November 18, 2011

1 High priority software suitability characteristics

1.1 Correctness

It is fundamentally important in almost every system that the developed software implement and support the system requirements correctly. The importance is elevated in a system such as ACIDS that supports mission critical activities like IED clearing.

1.2 Reliability

Having established *correctness*, the software must also be reliable. That is to say, it must perform the *correct* action consistently, predictably, and without fail.

1.3 Usability

In an effort to reduce potentially catastrophic human errors, the software must be usable. This also has the desired side-effects of reducing training time and increasing overall system adoption.

2 Requirements Statement

2.1 Correctness

Each system requirement shall be supported directly by one or more calls, methods, procedures, components, classes, or libraries of the METAL-V operational software which shall map directly to and support system requirements.

2.2 Reliability

The METAL-V operational software shall perform consistently as written under normal operating conditions with a reliability greater than 0.98.

2.3 Usability

The METAL-V operational software shall present an interface that closely resembles the physical operation of the system.

I considered this at some length and I believe that it is important to illustrate a couple related aspects to *usability*. There is overall system usability that may be measured in reduced training time and overall adoption of the system. But there are lower level usability considerations that, in aggregate, may be just as important and are possibly more easily tested.[1]

The METAL-V operational software shall prevent over rotation of the number six servo.[2]

So while this requirement does not fall into the traditional lane of *Usability* it does address and illustrate the principle that software should make the user's job easier wherever possible by reducing complexity[3] and mitigating against user-induced fault.

3 Test and evaluation activities

3.1 Correctness

The principle aim is to ensure complete coverage of all of the system requirements by the METAL-V operational software. By providing traceability back to the requirements of the software components we can assure that coverage is complete. Often the biggest difficulties arise either when requirements change or when new requirements are added. Maintaining a complete mapping requires diligence and commitment on the part of the software development team as well as management.

3.2 Reliability

Determining software reliability is often a matter of mapping and exercising every path of execution. This method falls apart, though, for *object* and *event-driven* models. Careful design emphasizing encapsulation and abstraction facilitates unit testing where it is frequently more tractable to *prove* code by identifying pre- and post-conditions[4].

3.3 Usability

For this, the more specific the requirements are the easier they are to test. In the two examples above, the constrained rotation is highly objective and easily testable whereas the former example may require the use of user juries and user acceptance testing. In cases like this it is often useful to get interface prototypes in front of users as early as possible to verify the proper capture of work flows as well as gather essential user feedback.

[1]Requirements that are not testable are not good requirements.

[2]This is a fictitious element of the hardware that we are supposing could cause severe problems if over rotated.

[3]Complexity in this case refers to the amount and nature of extraneous details that the user must attend to.

[4]See Dijkstra

4 Software component suitability

4.1 Prototype MRC and METAL-V firmware

While this is COTS software, it is not suitable for acceptance and lifecycle use because it is only applicable in the context of the prototype hardware (Lego Mindstorms).

4.2 Prototype MRC and METAL-V operational software

While it is difficult to be certain, it is likely that the prototype operational software could be engineered in such a way that it is (mostly) ready for acceptance and lifecycle use. This pre-supposes, however, that the operational software is sufficiently abstracted from the prototype system.

4.3 Prototype operational software generator

Assuming that these are currently available components it is possible that this section might be suitable for acceptance and lifecycle use. It is worth mentioning, though, that Borland has transformed itself and has abandoned Delphi and that Object Pascal may not be as easy to support throughout the system lifecycle.

4.4 ACIDS diagnostic software

It is very difficult to tell here but it seems that BricxCC is a Lego Mindstorm specific tool and would not be suitable for acceptance and lifecycle use.

5 Executive summary

Given the critical nature of the mission and the hostile environment for which the ACIDS system is targeted, an emphasis on *usability*, *reliability*, and *correctness* is suggested. Although it is fundamentally important in almost every system that the developed software implement and support the system requirements *correctly*, the importance is elevated in a system such as ACIDS that supports mission critical activities like IED clearing. Failure to correctly and adequately cover the system requirements could unduly place warfighters in harm's way.

Having established correctness, the software must also be *reliable*. That is to say, it must perform the correct actions consistently, predictably, and without fail. If the software is correct and defect-free then it will also be reliable. Rechtin & Maier suggest that the number of defects remaining in a peice of software are proportional to the number of defects found. Where lives are at stake the goal should always be to produce the highest quality error-free software.

Usability in the field is of great concern. Software that is difficult to use increases the potential for catastrophic human errors. Addressing issues of usability also has the desired side-effect of reducing training time and increasing overall system adoption. The standard for usability has been set by commercial giants Microsoft and Apple.

The first step to enforcing these criteria is to state each as a requirement or set of requirements. In the case of *correctness* we can state a high level requirement to ensure that all of the system requirements are supported in code by mapping each part of the code back to system requirements.

> "Each system requirement shall be supported directly by one or more calls, methods, procedures, components, classes, or libraries of the METAL-V operational software which shall map directly to and support system requirements."

For *reliability* we can spell out a measure of overall system performance.

> "The METAL-V operational software shall perform consistently as written under normal operating conditions with a reliability greater than 0.98."

Lastly, a high level usability requirement will not only assist in the design and implementation of the system software but will also help convey a common understanding of the problem by showing how the algorithmic solution models the physical system.

> "The METAL-V operational software shall present an interface that closely resembles the physical operation of the system."

Further consideration illuminates a couple related aspects of usability. There is overall system usability that may be measured in reduced training time and overall adoption of the system. But there are lower level usability considerations that, in aggregate, may be just as important and are possibly more easily tested.

> "The METAL-V operational software shall prevent over rotation of the number six servo[5]."

So while this requirement does not fall into the traditional lane of Usability it does address and illustrate the principle that software should make the user's job easier wherever possible by reducing complexity and mitigating against user-induced fault.

By providing traceability back to the requirements of the software components we can assure that coverage is complete. Often the biggest difficulties arise either when requirements change or when new requirements are added. Maintaining a complete mapping requires diligence and commitment on the part of the software development team as well as management.

Determining software reliability is often a matter of mapping and exercising every path of execution. This method falls apart, though, for *object* and *event-driven* models. Careful design emphasizing encapsulation and abstraction facilitates unit testing where it is frequently more tractable to *prove* code by identifying pre- and post-conditions[6].

For usability testing, the more specific the requirements are the easier they are to test. In the two examples above, the constrained rotation is highly objective and easily testable whereas the former example may require the use of user juries and user acceptance testing. In cases like this it is often useful to get interface prototypes in front of users as early as possible to verify the proper capture of work flows as well as gather essential user feedback.

While prototype MRC and METAL-V firmware is COTS software, it is not suitable for acceptance and lifecycle use because it is only applicable in the context of the prototype hardware (Lego Mindstorms). On the other hand, it is possible that the prototype operational software could be engineered in such a way that it is (mostly) ready for acceptance and lifecycle use. This pre-supposes, however, that the operational software is sufficiently abstracted from the prototype system.

Assuming current availability, it is possible that the Prototype operational software generator might be suitable for acceptance and lifecycle use. It is worth mentioning, though, that Borland has transformed itself and has abandoned Delphi and that Object Pascal may not be as easy to support throughout the system lifecycle as some other languages.

[5]This is a fictitious element of the hardware that we are supposing could cause severe problems if over rotated.

[6]See Dijkstra

With regard to the ACIDS diagnostic software it is very difficult to tell. It seems that while BricxCC supports a growing list of programmable microcontroller platforms it is still very tightly tied to hardware and so may not be suitable for acceptance and lifecycle use. The following is from the BricxCC SourceForge project page.

> ”Bricx Command Center (BricxCC) is a Windows (95, 98, ME, NT, W2K, XP, Vista) program commonly known as an integrated development environment (IDE) for programming the RCX (all versions), Scout, Cybermaster, and Spybot programmable bricks using Dave Baum’s Not Quite C (NQC) language. And it supports programming the Scout, RCX2, and Spybot using The LEGO Company’s MindScript(tm) and LASM(tm) languages via the Mindstorms 2.5 SDK. It supports programming RCX bricks in C, C++, Pascal, Forth, and Java using the brickOS, pbForth, and leJOS alternate firmwares. BricxCC now also supports programming the new LEGO Mindstorms NXT brick using Not eXactly C (NXC), Next Byte Codes (NBC), and a simple on-brick programming language called NPG.”

The overall state of the ACIDS prototype software is not suitable for acceptance and lifecycle use. Too much relies on the specific implementation which uses the Lego Mindstorm NXT brick. Sufficient software engineering by way of abstraction and encapsulation may result in the ability to re-use (or at least leverage) some of the prototype operational software. But the fielded implementation may well be similarly tailored to the specifics of the production hardware. This is due to the hardware specific nature of programmable microcontrollers and the lack of portable and generalized routines and libraries for sensor and actuator access. Nonetheless, the prototype system is valuable in that it provides early feedback on system viability, user adoption, and importatnt lessons learned.

The success of the system may ride on careful selection of key criteria. Emphasizing *correctness*, *reliability*, and *usability* will place proper emphasis on the strengths that will be most imporant to safety, performance, and sdoption in the field.