An XP inspired test-oriented life-cycle production strategy
for building embedded biomedical applications

James Miller¹, Michael Smith²,³, Steve Daenick⁴, Jingwen Chen², Juan Qiao², Fang Huang²,
Andrew Kwan² and Marc Roper⁵

¹Dept. Electrical and
Computer Engineering,
University of Alberta,
Edmonton, Alberta,
Canada
²Dept. Electrical and
Computer Engineering,
³Dept. of Radiology,
University of Calgary,
⁴NovAtel Inc.,
Calgary, Alberta, Canada
⁵Dept. Computer and
Information Sciences,
University of Strathclyde,
Glasgow, Scotland

jm@ece.ualberta.ca, Mike.Smith@ucalgary.ca, Steven.Daenick@novatel.ca, {JinChen, Jqiao,
Fhuang, AkcKwan}@ucalgary.ca, marc@cis.strath.ac.uk

Abstract

The construction of embedded biomedical applications is an under explored topic. The status quo is for practitioners to utilize a production process which possesses no specific focus; meanwhile, the marketplace requires highly demanding characteristics from these products. The principal requirement is that most of these products need to be effectively defect free. This demands that the production process be directed towards this objective; and hence the focus of this paper is our initial attempts at designing and implementing such a process. Our new process is developed around transforming a subset of extreme programming from the world of desktop applications into a methodology for this new domain. The paper also discusses our experiences in developing test frameworks to support the domain and our objectives. Finally, the paper provides some pointers on our future plans for tackling the many unresolved issues that still exist in attempting to fully realize and support this new development process.

1. Introduction

Computerized systems are increasingly exploring new application domains, resulting in requirements for new viewpoints on production processes. While this area is perhaps most noticeable within the domain of web-based systems, new areas of embedded systems are also emerging. For example, biomedical systems allow pregnant women to be monitored from home, instead of being admitted to the hospital; Rheumatoid Arthritis patients can be monitored remotely during rehabilitation exercises; pacemakers can be adjusted remotely; and blood pressure and heart rate measurements can be transmitted via the telephone or Internet to health professionals. In addition, many new technologies are under development, including wearable and wireless versions of these medical sensors. These sensors can be integrated into clothing or worn like a watch; measuring vital signs for the onward transmission to the health-care professional.

The development of such Small Micro-systems for Increasing Life Expectancy (S.M.I.L.E. ©) raises some important issues.

(1) There are the concerns of how to manage expectations of, and communication within, the multi-disciplinary team constituted by the medical end-users, clinical researchers and the engineers involved during the design and development processes.

(2) At the “plumbing” end of the spectrum, these systems possess highly demanding characteristics, especially with regard to many non-functional requirements of the embedded systems. However, of
paramount importance is that many of these systems have a near zero tolerance with regard to defects! Hence, the authors believe that it is necessary to move to production processes that explicitly target defect prevention. Such a recommendation is not without precedence as there are strong parallels with recent directions in desktop software production processes. For example, Mead and McGraw [1] outline a waterfall-life-cycle based “security-aware” development process for handling issues arising as desktop applications become increasingly interconnected. Within this process, security assurance now exists at every juncture throughout the development process: from the use of Abuse Cases during requirements analysis, to the use of Penetration Testing as the final activity before product release. It is believed that a similar re-casting of production processes for this emerging sector of embedded biomedical applications is necessary if the domain is to become successful and flourish.

While recognizing these needs, the new process must still meet the requirements of the company with regard to providing a competitive platform and a process to which their existing staff can adapt. We agree with the viewpoint expressed in [2] that the classical standpoint of “pushing formal methods to industry” is unlikely to be productive; and instead take a more realistic viewpoint of “starting from established practices” and gently leading towards more rigorous practices and processes. Indeed even this “steady as she goes” approach outlined in this paper is highly likely to be considered as “too aggressive” for many companies; especially when considered in light of recent adverse industrial findings regarding UML usage patterns [3]. In addition, to remain competitive, modern development processes need to be adaptive and highly flexible; especially given the increasing time pressures on software development [4]. These development objectives almost mandate the extensive use of a COTS centric, or asset-based, production methodology. Due to economic necessity, many mainstream products are also being driven down this path; and hence this picture is highly likely to generalize beyond this domain.

These forces demand that any realistic development strategy for this sector needs to incorporate these, often competing, requirements. The central objectives of the approach are: defect-prevention focus, incremental change from current approaches, highly flexible and adaptive methodology; supporting multi-disciplinary team interaction, and the integration of COTS, assets, or legacy components into the production methodology.

2. Problem Description – the existing life-cycle

Given these objectives, it is essential to understand the current development approach deployed within the domain before attempting any modifications. At a very high level, the process consists of four stages:

PRODUCT ENVISAGEMENT: The development of the initial concept of the clinical embedded solution to the medical problem. This stage effectively produces the requirements and hence the acceptance testing criteria. It will commonly involve interactions (communicating) with a clinical end-user (probably a medical doctor) and a clinical researcher. While both are highly educated people, their levels of I.T. expertise should be considered suspect (a risk factor), especially for the clinical end-user. A similar comment can be made of the medical domain knowledge available within a typical development team.

PROOF OF CONCEPT OR PROTOTYPING PHASE: Commonly, the proposed solution is analyzed by producing an initial prototype to help the researcher prove that the solution works beyond the theoretical level. The solution typically revolves around numerical algorithms and hence is implemented in a specialized mathematically-oriented scripting language such as MATLAB or IDL. The clinical researcher, who normally has limited skills in terms of producing the “final production system”, performs this service. This stage effectively defines the system’s functionality and should define the corresponding set of test objectives.

INITIAL PRODUCTION SYSTEM: The MATLAB / IDL system only provides an “off-line” simulation vehicle to explore the validity of the approach. Hence, the next stage would commonly require the system to be re-written in a higher-level compiled language such as the standard C / C++ or perhaps specialized versions of C (e.g. nesC running under TinyOS [5]). While this stage may use tools to transform the MATLAB into C, it normally involves a significant amount of manual coding and is performed by an IT professional who tries to ensure that the two systems are functionally equivalent. The biggest issues here surround the movement of the “off-line” MATLAB calculation (double floating-point 64-bit precision) onto the high-
speed embedded system (integer 32-bit precision) with issues of determining adequate levels of signals scaling and providing appropriate numbers of guard bits to ensure algorithmic accuracy [6]. Typically, this third stage might take place on a host environment. However simulation times surrounding the modeling of low level, extremely optimized code running on processors with highly parallel instructions can be extremely time consuming. It is therefore likely that the development would occur through a method where such code is actually prototyped on a subset of the embedded target, i.e. manufacturer’s evaluation kit.

**FULL PRODUCTION SYSTEM:** In the last phase, the “C-code” is adapted and evolves to meet the needs of the final product. Although some of the third stage might take place in a desktop environment, at this fourth stage we are fully involved with the target embedded environment itself. The system now needs to adapt to its new hardware environment, interact with device specific peripherals and ensure that its generalized I/O structures take advantage of the communication facilities on the target system. The system also goes through further modification to tune for the required performance by re-writing inefficient programming constructs, using #pragmas to direct architecturally aware compilers and rewriting critical components in machine code. Again, an embedded professional undertakes this stage; and again its output must be functionally equivalent with the previous stage.

The work performed within each stage could well be iterative in nature; but that will depend on the experience of the development team. However, the entire life-cycle tends to be relatively sequential with the majority of the iteration occurring as a consequence of missing or erroneous details in earlier phases rather than as a planned event. However, there is one anticipated exception to this waterfall-like overall process. Since the use of S.M.I.L.E.s for improved patient care is a recent addition to Medicare, it is likely that new clinical best practices will be identified during the initial prototyping, so that the development process must be flexible enough to accommodate any associated changes. Although within many projects the various phases will be merged through a variety of adopted styles, the over-all process will remain faithful to this general sequential pattern.

3. **The new test-oriented life-cycle**

Given, the large array of requirements and restrictions outlined in the previous sections, only a limited number of avenues seem possible. But, perhaps surprisingly, the authors believe that this problem can be solved by adapting an “agile approach”; XP to be specific [7]. While XP is best viewed as a “generic approach” to software production, the authors believe that a sub-set taken from the concept [7] can be adapted to define the core of a specialized (fault-prevention) approach to the production of embedded systems software. Having said this, the adaptation is quite significant and the originators of the XP approach may well not view it as being faithful to their original concept. For example, Test Driven Development [8] is normally considered a process to promote the active exploration of the “design space”; however several researchers (e.g. [9-11]) have shown, via industrial-based empirical studies, that it can also be viewed as an extremely effective defect reduction strategy. Furthermore, in fitting with the pre-existing embedded system process structure, our approach is currently much more sequential than XP; and while we believe that, in the long-term, parts of the process may evolve to become more iterative; in the short term, this is not a priority from a testing perspective.

Our “XP inspired” (XPI) process seeks to transform the pre-existing process by imposing additional test-oriented structure upon this model:

**XPI PRODUCT ENVISAGEMENT:** In this phase, we get the clinical customer to specify their ideas as a set of acceptance tests; specifically we use FIT / Fitnesse test frameworks [12,13] to produce executable test statements (as input and expected output pairs) which allows us to directly ensure that the acceptance level test objectives are met. Here we seek to deviate from the traditional viewpoint in three respects:

(A) Coverage: Current Agile methodologies provide limited guideline to the customer with regard to test case construction, volume and diversity. Clearly, this is a difficult topic as we are dealing with a non-I.T. specialist – but the production of comprehensive set of tests (i.e. with adequate coverage) from the clinical customer is paramount to the successful production of the health-care system.

(B) Non-functional Tests: Agile methodologies tend to separate functional and non-functional test production. For example, in [8], Beck describes the functional tests driving the production process while the non-functional tests suddenly appear at the end.
almost as an afterthought. We believe that the production of non-functional acceptance tests is paramount and are currently re-developing acceptance testing frameworks to support these essential components.

**C** The acceptance tests now “drive” the prototyping stage, not the final production stage.

**XPI PROOF OF CONCEPT OR PROTOTYPING PHASE:** The “acceptance tests” now provide the initial “skeleton” for the MATLAB / IDL prototype. The prototype will clearly add “flesh” onto this outline structure; and hence we need to introduce further testing support at this level. Again, following a XPI model, the support at this level is two-fold:

- **A** The introduction of a unit testing framework (e.g. the MATLABUnits found in [14, 15]). These frameworks have similar functionality to the well known JUnit.

- **B** Use of Test Driven Development (TDD): Despite the clinical researcher being a non-I.T. specialist, they will be potentially highly skilled in writing MATLAB – however, our experience suggests that “formalized testing” tends to be an important but overlooked skill in their production process. We have conducted a number of informal trials, and find that this more formal process does seem to help clinical researchers in structuring their thoughts around the objectives and requirements of the system. This “result” is perhaps not too surprising. We agree with Mugridge [16] that TDD has strong parallels with the “scientific method”, and hence the establishment of a “hypothesis” (test) followed by its experimental realization (solution code) is a common idea amongst both groups of practitioners.

**XPI INITIAL PRODUCTION SYSTEM:** The above pattern is now repeated in the new programming language with the new unit testing framework. However due to the existence of Stage 2, the Test Driven Development activity is now transformed. Test Driven Development envisages a highly iterative development cycle: you conduct a small test then produce code to satisfy the test; and repeat this cycle until completed. However, we now start with the “majority” of tests already constructed via the previous stages and hence this third stage is perhaps better described as a “design by contract” [17] process; albeit potentially in an imperative rather than object-oriented style. This form of producing “contract” statements will “hopefully” lead to practitioners producing more comprehensive contractual statements. In previous low-level projects, e.g. constructing a system shell, it was found to be extremely difficult to construct a comprehensive set of contracts. Similarly, if one looks at the Eiffel libraries, most functions have a very limited set of post-conditions. We believe that this more gradual approach to these contractual objectives will lower the cognitive overhead of the task and allow it to become a reasonable possibility for the average practitioner.

**XPI FULL PRODUCTION SYSTEM:** This final stage repeats the pattern but now the unit test framework becomes a significant issue. Several comments are present in the literature concerning the lack of tool support for low-level development [18] or the equivalent of an “assembly” xUnit testing framework [19]. The approach cannot hope to have genuine traction without appropriate test framework support; although we know of a couple of commercial projects that have attempted to proceed without one [19, 20].

In summary, we believe that these new processes, although not fully developed, have the ability to fulfill the objectives outlined in the opening section. We view the XPI approach as a “mongrel”; we have unashamedly “stolen” the best test-oriented ideas that we can find and adapted them in an effort to solve our embedded production problem.

In the remainder of the paper, we will examine these issues in terms of our experiences in attempting to place Agile tools into the embedded system design world. Although some of these examples were not, per-se, designed as bio-medical engineering projects, they encompass components that could be used in that context.

1. Verification and validation of cerebral blood flow (stroke) algorithms [21] written in MATLAB to identify why discrepancies occur between two implementations of the well-known singular value decomposition (SVD) deconvolution algorithm [22]. One implementation provided results that were interpreted as changes (overly optimistic) in a medical condition of the patient’s brain.

2. Design of the hardware and software components of an Analog Devices BF533 (Blackfin) based embedded system interface for an oxi-meter (oxygen level) and thermo-sensors. This problem was examined from several different points of view; the customer (EmbeddedFIT) [23], the developer (EmbeddedUnit) [24] and from experiences of developing prototypical systems as a laboratory exercise with a third year Engineering class [25, 26].
A video-surveillance project [27] used to investigate the use of EmbeddedUnit [24] in a more sophisticated environment. This could be considered a prototype adaptable for more complex bio-medically related surveillance (intruder in a hospital pharmacy, Alzheimer patient wandering); and included the use of tests developed during the design phase of a Analog Devices Blackfin (ADSP-BF533) processor based system to validate operation on a different processor.

Two projects involving components of complicated image processing algorithms and telecommunications combined with simple timing operations of relays. The concepts could be considered as relevant to many elements found in robotic and remote surgery.

4. XPI Product Envisagement

EmbeddedFitNesse can be use as a communication tool between clinical end-user and embedded developer in the S.M.I.L.E. product envisagement. It is modified version of FitNesse to fit into an embedded system development environment. Our testing framework (Figure 1) connects with both an embedded system’s development environment (VisualDSP) and MATLAB’s API. The DSPInterface class is built to provide a VisualDSP service to FitNesse. Through this interface, FitNesse can access the embedded memory, control the embedded target, build and run the embedded program on the board and analyse the results produced during execution. A test Runner is added and loaded into target board and is used as the VisualDSP test engine. The EmbeddedFitNesse work flow is: (1) Both sides of the multi-disciplinary clinical and engineering teams work together to produce test tables with the corresponding Fixture code written by the ‘I.T.’ savvy developers. (2) The FitNesse server reads all the table information from a Wiki page; which is sent to the Fit server for execution. The fixture code provides the Fit server with an execution guide for the test table. (3) The Fit Server extracts all the test data which is then sent to VisualDSP. (4) The Visual DSP Test Runner executes all the tests, directed by the test information, and sends back the result of the embedded system’s behavior to the Fit Server; which in turn, returns the test result back to FitNesse to report to the customer.

Our preliminary results with EmbeddedFitNesse [21, 23] indicate that certain classes of embedded tests are easier to express within a Fit table than others. For example the relationship between the blood-oxygen levels and the light-sensor output related to those levels is easy to discover. These results are essentially the scientific equivalent of the business relationship surrounding discount and number of items; and hence easy to express in a Fit table. A slightly more difficult test-writing example surrounds the use of Fit tables to express the timing relationships between relay signals to accept, reject and pack fruit in a Fruit Sorting and Packing machine (an embedded but not a biomedical application). This latter example is pushing Fit’s capability, but its relationship to existing Fit usage is easy to see.

The biggest conceptual problem so far experienced with using EmbeddedFit was with trying to express the concept of Fruit Blemishes. This series of Fit entries was required to permit testing of the image processing algorithms present in a Fruit Sorting embedded system. This is the equivalent of generating a Fit table to ensure that a robotic surgeon can handle the wide individuality of each patient’s organ.

Figure1. Block diagram of a test environment demonstrating how Fit fixtures can be used to run customer tests on an external embedded target platform or within the MTLAB API.
The discussions surrounding trying to build these tables has so far proved more useful in uncovering system issues than actual attempts to express these items in the form of Fit tables! However, this probably indicates either the team’s lack of experience (and mentorship) with the new tool; or that traditional FIT training does not transfer well into this new domain.

The biggest practical limit to the use of *EmbeddedFitnesse* is the communication time associated with moving the tests and results between the host machine and the embedded system. It is very time consuming to essentially “set-up” and “tear-down” the embedded environment for each *FITnesse* table entry. We have re-configured the system so that set-up / tear-down is only necessary once for all the table entries on a given *Wiki* page. More research is necessary to determine if this approach is sufficient, or whether a *Fitnesse* compiler will need to be constructed.

**5. XPI Proof of Concept or Prototyping Phase**

Our initial goal was to develop a C++ based *MATLABUnit* where it was possible to have a single series of tests that could be used directly on both the original *MATLAB* functions running within the *MATLAB* engine and the derived C++ code running directly on the embedded system [28]. The tests and code would be compiled to run on the host when testing the *MATLAB* code; and later compiled to run on the actual prototype / production system when development reached that stage.

With this one-set-of-tests two-environments approach, we were able to validate cerebral blood flow algorithms used to determine perfusion levels in the brain (location and extent of brain tissues damaged by stroke) [21]. This method of implementing a *MATLABUnit* was successful; but totally unusable. The direct C++ interface to the *MATLAB* engine proved to be too primitive for everyday use; you could if you had to, but you would not want to make a habit of it!

We have examined a number of different approaches by which one single set of tests could be used in all stages of the embedded life cycle. In [29] we were able to demonstrate *MATLABFit* applied to the embedded environment. This, combined with *EmbeddedFit*, would potentially allow one set of acceptance tests for validating both *MATLAB* and embedded C++ code. However this *EmbeddedFit / MATLABFit* combination does not provide a satisfactory solution for a common unit test framework. We are currently exploring two different approaches to solve this issue. It is already common practice to test *MATLAB* algorithms using Simulink [30]. This is a platform for simulation and Model-Based Design of dynamic systems in which the solution (model) is stated by sequential blocks of numerical calculations. It is true that various inputs and environment factors can be applied to the model and the output examined; and it is also true that this approach does give more of a direct view of the solution for clinical researchers and even non-clinical people to understand. However, these two facts do not make *Simulink* a testing framework! Given the characteristics of *Simulink* we are working towards an approach to develop *SimulinkFIT* rather than *SimulinkUnit*.

We have been more successful in exploring a different approach to providing a single test suite for the unit testing across the final three stages of the embedded life-cycle. Consider tests for a function *CelsiusToFahrenheit()* to change a Celsius temperature into Fahrenheit written in *mlUnit* [15]

```matlab
function it = test_temperature(it)
    f = CelsiusToFahrenheit(30);
    assert_equals(f, 90);
end
```

and the same test written in *EmbeddedUnit* [24]

```cpp
TEST(test_temperature, CelsiusToFahrenheit)
{
    int f = CelsiusToFahrenheit(30);
    CHECK(f, 90);
}
```

The similarity of the syntax between the two examples suggests that we have one set of tests written for *mlUnit*; and a parser to translate those tests into the syntax for *EmbeddedUnit*. We suspect that more efficient test code (smaller embedded system memory requirements) will be generated by the parser approach than by simply applying a *MATLAB-to-C++* translator onto the *MATLAB* code, *mlUNIT* tests and associated support framework.

**6. XPI Initial production System**

The *EmbeddedUnit* described in [24] is based around Feather’s CPPUnitLite [31] and is the unit test framework that we utilize for this and the following
stage. A key reason for choosing CppUnitLite as a basis for the EmbeddedUnit tool was the fact that it was “non-scripted”, implying that the framework detects the tests to be run automatically, freeing the developer from working with a script running on an external development environment (PC). Such external scripts require that the embedded system be stopped in order to run, or report on, or be interrogated about, a specific test. It should be remembered that on many embedded systems simply providing the equivalent of a printf() or cout statement is a severe technical challenge!

EmbeddedUnit can be considered as consisting of two components.

(1) A generic testing unit modified for the limited memory capabilities present in an embedded system. This component is equally useful whether the test is run on a host system or an embedded platform. A typical test used to validate a temperature calculation procedure CalculateTemperature() is shown below.

```cpp
#define CELSIUS 1
typedef temperatureValue4 signed short;

TEST(CELSIUS_20, CalculateTemperature) {
    // Timed high / low thermal-sensor values for 20 C temperatureValue4 value;
    value = CalculateTemperature(400, 215, CELSIUS);
    SHORTS_EQUAL(value, 80);
}
```

(2) An embedded system specific testing unit that is designed to make use of special processor functionality through new WatchDataClass() and WATCH_MEMORY_RANGE() test macros. These make use of the ability of the Analog Devices Blackfin ADSP-BF5XX processor range to recognize, with zero overhead, operations being performed on its internal registers and internal fast L1 and L2 memory banks and the much slower external L3 bank.

These tests should be considered as an embedded extension to, rather than a strict departure from, the “oracle” style of classical TDD since a given computation is run and its output, the number of memory mapped register hardware operations performed and the results of those operations, compared to values predicted in advance. The timer and watch-data E-TDD classes are general in concept, but must be specifically implemented using processor resources, while not removing resources needed for the normal development of the system.

The following example demonstrates embedded tests made on a procedure SetCoreTimer() that initializes the processor’s internal core timer registers:

```cpp
typedef unsigned long int;

TEST(Test_SetCoreTimer, INSTRUCTOR) {
    // Setup “watching” on specific processor registers
    WatchDataClass<unsigned long> coretimer_reg(4, pTCNTL, pTPERIOD, pTSCALE, pTCOUNT);
    // Setup the expected final values of the registers
    unsigned long expected_value[] = {
        0x0, PERIOD, SCALE, COUNT};

    // “Watch” the initialization function performance
    WATCH_MEMORY_RANGE(coretimer_reg,
        (SetCoreTimerASM(COUNT, PERIOD, SCALE)),
        READ_CHECK | WRITE_CHECK);

    // Determine if the number of register read / write operations
    // and final register values match the design criteria
    CHECK(coretimer_reg.getReadsWrites() == 4);
    ARRAYS_EQUAL(expected_value, coretimer_reg.getFinalValue(), 4);
}
```

Other processor specific testing macros we have introduced include the test macro MEASURE_EXECUTION_TIME() which automatically uses the on-chip clock to measure the execution (performance) time of the Function(parameters) code. Whether this execution time meets critical performance power and time characteristics can be checked through the MAXPOWER_ASSERT() statement and through the MAXTIME_ASSERT() statement respectively. These are the first of a series of functional and non-functional tests for embedded systems planned for development.

Educational use of TDD: The WatchDataClass() class was initially developed for use by an expert (e.g. an instructor with an intimate prior knowledge of the system architecture). Expert tests would be written to automatically examine whether the students developing code for the processor had properly configured the registers of a peripheral. Both register values and register access operations are evaluated, i.e. checking that registers have been specifically set to the required values rather than left with the default (reset) values which “just happen” to be the same as the required values. However, in practice, this test class proved to be unexpectedly much more utilitarian. The 3rd year undergraduate class, in the very first assignment after being trained with EmbeddedXunit for use in a hardware-software co-design laboratory, discovered unexpected (undocumented) behaviour of a new processor’s core timer resources. Such behaviour
could play (could be playing) havoc within industrial products if it remained unrecognized. The students, through the methods of the WatchDataClass test class, were able to identify, and then stabilize, the error’s behaviour (to prove that it existed) so that the problem could be reported to, and recognized as an issue by, the chip manufacturer [24-26].

7. XPI Full production system

When we move onto supporting the testing of the full production system, the unit testing framework has to accommodate a number of low-level or embedded system specific issues. In this section we will illustrate these issues with three concrete examples.

The first example concerns an industrial customer who, for their own business reasons, wishes to place video and other applications into an uClinux environment. The main design criterion of the Linux kernel is throughput not pre-emptability; and the scheduler is called at fixed (1ms) interval and not at arbitrary times. This leads to non-determinism of response times surrounding interrupt latency, interrupt duration, scheduler latency and scheduler duration. To overcome these issues the customer has explored the uClinux kernel 2.6, which is capable of meeting very stringent timing requirements; and using RTLinux (Real-time Linux [32]) and RTAI [33] (Real Time Application Interface) where the uClinux base kernel is run as the lowest priority thread within a minimalistic real-time kernel.

The customer is asking for tests to show that changes in the OS environment actually solve the known problems rather than pushing them somewhere else. We successfully moved the “software” component of our EmbeddedUnit into the uCLinux environment. However the hardware components of EmbeddedUnit make specific requests for the system registers which are religiously protected by the uClinux kernel; resulting in the Kernel Scramble and Core Dump -- the Linux equivalent of the “Blue Screen of Death” found with some desktop Windows applications. We are moving some of the hardware EmbeddedXUnit features into the kernel as drivers; but that opens up many other implementation concerns.

The second full production example concerns the implementation of a video-surveillance program running as a prototype on an Analog Devices BF533 evaluation board (See Fig. 2). Intruder alerts were initially sent to the customer through a telephone interface. However, after completion of the prototype work, the application (and associated tests) were migrated to the newly released Analog Devices BF537 board which supports an Ethernet connection; and can be reconfigured to run a uClinux kernel. It required considerable extension to the existing EmbeddedUnit to solve the issues raised when developing / testing the code for the surveillance program using our XP-inspired life-cycle. Some of those extensions are discussed in the following paragraphs.

MEMORY ALLOCATION ISSUES: As mentioned earlier a key element that distinguishes embedded system development from desk top application development is memory – size, type and location. There are three types of embedded system memory. The CppUNITLite macros were modified to allow the embedded extensions present in the development environment to place the code so that the tests and associated asserts would run in L2 or L3 memory while allowing the code under test to run at full speed in L1 memory. The exact syntax for achieving this goal depends on the embedded system extensions for a given compiler.

![Fig. 2 The schematic of the thread-blocks needed to support a real-time security system providing video surveillance and entry point motion detection is shown [27].](image-url)
TESTING CLASS METHODS: Since all test classes are instantiated by the compiler, a lot of memory is used when the classes under test have extensive attributes. Unit testing of class methods may be time sensitive so the class attributes cannot be stored in slow external L3 memory. To get around this problem a mock test class needs to be created which then dynamically creates the real test class (in fast L1 memory) that inherits the class under test. Extensions to the basic CPPUnitLite macros were added to the EmbeddedUnit test harness to allow test classes to inherit the class being tested.

The third system investigated was a video teaching tool for pool-players where “best-shot” strategies were identified through image analysis, and these strategies were send over a telecommunication link to remote observers. The RF communications module required setup of specific hardware register configurations; tested using the EmbeddedUnit WatchDataClass discussed previously. Testing the wireless protocol was a greater challenge since that involved synchronizing two devices. This was solved by categorizing and setting the devices into known states: a master device and a slave device. The following test listing (running on both slave and master) demonstrates a loopback testing scenario with the slave device resending the packet on receipt; and the master checking the validity of the bounced packet.

```
TEST(LoopbackTest, DEVELOPER_TESTS) {
  int iRandomNumber = rand();
  int iReceived;

  if (Master) { // Master test
    SendPacket(iRandomNumber);
    iReceived = ReceivePacket();
    CHECK(iReceived == iRandomNumber);
  }
  else {       // Slave test
    iReceived = ReceivePacket();
    SendPacket(iReceived);
  }
}
```

The simplicity of the test code belies the problems underlying its use. These tests are relying on an external stimulus to complete; and will hang if that is not received. We are proposing to use the processor’s internal watch-dog timers to ensure recovery; although the issue of then re-synchronizing the systems for further reliability tests of the transmission channel (e.g. system passes the test if 95% of the transmissions are successful) remains unsolved.

8. Conclusions

One of the hallmarks of the 21st Century is that software is truly becoming ubiquitous. This fact drives the field into many new and uncharted areas; and this increasing diversity implies that we are unlikely to be able to utilize a single production methodology for every different type of system. In fact, these new types of systems often demand that new characteristics be incorporated into the system’s requirements – e.g. trust is an important characteristic for many e-commerce type systems. In this paper, we are concerned with the emerging field of embedded biomedical devices; and for this class of product – intolerance of defects is the main distinguishing characteristic. Hence, the paper outlines our approach to producing a specialized production methodology, which explicitly targets defect prevention, while retaining essential features of current common production processes within this domain – to allow easy industrial transfer between the processes. While, the outline of the new process sounds relatively straight-forward to make the process realistic a great deal of technology needs to be produced to support the new process, specially a number of testing tools, one for each of the proposed sub-processes. These tools can be considered to be either acceptance test tools (in the style of Fit) or unit test tools (in the style of xUnit). The paper discusses progress to date, and while we have made great advances in these areas, a large number of issues still remain unresolved in completing the picture.

We have demonstrated EmbeddedFit and EmbeddedUnit using Analog Devices’ integer ADSP-BF5XX (BlackFin) micro-controller and highly parallel, multi-processor-oriented TS201 (TigerSHARC) micro-processor. Although these are from widely different ends of the spectrum, they belong to one manufacturer’s family of processors. Our next step is to examine whether it is true in other processor families, as it was with the Analog Devices products, that it is easier to develop a xUnit tool for micro-controllers, with their plethora of internal resources, than for the more sophisticated processors.

Despite having many unresolved issues and components, we believe that the approach and tool support have already demonstrated considerable success; e.g. allowing undergraduate students to discover undocumented “silicon issues” from a DSP manufacturer.
9. Acknowledgements

Financial support was provided through the University of Calgary, Analog Devices and the Natural Sciences and Engineering Council of Canada (NSERC) through a Collaborative Research and Development grant (CRD 299423-03). MRS is Analog Devices University Ambassador.

10. References