Using Software Engineering Technology to Improve the Quality of Medical Processes*

Lori A. Clarke
Department of Computer Science
University of Massachusetts,
Amherst, MA 01003
clarke@cs.umss.ed

George S. Avrunin
Department of Computer Science
University of Massachusetts,
Amherst, MA 01003
avrunin@cs.umass.edu

Leon J. Osterweil
Department of Computer Science
University of Massachusetts,
Amherst, MA 01003
ljo@cs.umass.edu

ABSTRACT

In this paper, we describe some of the key observations resulting from our work on using software engineering technologies to help detect errors in medical processes. In many ways, medical processes are similar to distributed systems in their complexity and proneness to contain errors. We have been investigating the application of a continuous process improvement approach to medical processes in which detailed and semantically rich models of the medical processes are created and then subjected to rigorous analyses. The technologies we applied helped improve understanding about the processes and led to the detection of errors and subsequent improvements to those processes. This work is still preliminary, but is suggesting new research directions for medical process improvement, software engineering technologies, and the applicability of these technologies to other domains involving human-intensive processes.

Categories and Subject Descriptors

D.2.4 [Software Engineering]: Software/Program Verification; J.3 [Life and Medical Sciences]: Health

General Terms: Verification

Keywords: Continuous process improvement, process verification, medical processes

1. INTRODUCTION

This paper summarizes key lessons learned from our initial efforts to apply the technology and approaches of software validation and continuous software process improvement to the reduction of errors in medical care. Medical errors occur frequently and, as reported in the 1999 Institute of Medicine report, To Err is Human [15], preventable errors in hospitals alone cause at least 97,000 deaths per year in the US.

Our initial work offers considerable promise that software engineering approaches, originally developed to support software process improvement, can reduce the incidence of such errors. This work has also demonstrated that exploration of this domain is useful in pointing to areas in which software engineering technologies should be improved.

There is surprising similarity between healthcare systems and software systems, particularly human-intensive, distributed system. Healthcare systems typically involve many different types of human agents (e.g., doctors and nurses with different specializations and roles, pharmacists, lab technicians, and support staff), hardware devices (e.g., infusion pumps, radiation therapy machines, and patient monitoring devices), and software applications (e.g., computerized physician order entry systems, decision support systems, and electronic medical records). Coordination is particularly key in these systems, as humans are often participating simultaneously in several different processes at any given time, and their participation in each process may entail the parallel performance of many different subtasks and interactions with several different devices and software applications. In performing these tasks, it is common for exceptional conditions to arise, requiring specialized actions that may vary considerably depending upon the circumstances. Continual change is also a key issue in medical processes. Changes may result from the introduction of new devices, new software applications, new personnel, or even personal preferences. New medical studies may lead to new guidelines or standards of care. Still other changes come as reactions to errors that have recently occurred locally. Such changes are usually made based only on informal analysis of poorly understood processes. These many parallels between medical processes and software engineering issues suggest that the software engineering community has much to offer in the search for ways to improve healthcare.

The University of Massachusetts Medical Safety project has been investigating how software engineering technology, originally developed to improve the quality of software systems, could be effectively applied to improving the quality of medical processes. In particular, we have undertaken several case studies intended to shed light on the applicability to healthcare processes of the classical Deming Cycle [9] of continuous process improvement employing software validation tools for error detection. These case studies have involved developing models of healthcare processes that are unusually detailed and semantically broad,

*This research was partially supported by the National Science Foundation under awards CCF-0427071, CCR-0205575, and CCF-0541035, and by the U.S. Department of Defense/Army Research Office under awards DAAD19-01-1-0564 and DAAD19-03-1-0133.

ICSE'08, May 10–18, 2008, Leipzig, Germany.
Copyright 2008 ACM 978-1-60558-079-1/08/05...$5.00.

analyzing these process models using finite-state verification and other analysis techniques, and then working with medical professionals to systematically improve the processes when errors have been found.

This project is succeeding in providing benefits to both healthcare and software engineering. The medical professionals involved have reported that this project has changed the way they view, describe, teach, evaluate, and improve their processes. Moreover, several serious problems have been uncovered and the medical processes have subsequently been improved. There have also been benefits to software engineering in that it has been necessary to enhance the technologies we have used in ways that should also improve their effectiveness when applied to software systems. Moreover, we now have a new perspective on software development, particularly for human-intensive systems.

2. APPROACH

The very broad outlines of our approach had been suggested in some earlier work [18, 19] that proposed that processes share many of the characteristics of application software and that technologies for the analysis and continuous improvement of application software might be usefully applied to processes. Applying these technologies to processes in domains other than software engineering was also suggested in [19]. The work we describe here provides substance and preliminary confirmation to these earlier suggestions.

The approach to process improvement that we have been developing is based on creating detailed, semantically rich models of the processes and then applying a number of different analysis techniques to try to detect errors in those processes. The analysis techniques not only detect errors, but also provide feedback about the source of the errors that can then be used to help determine how the processes should be improved. With help from the medical professionals, who in this case are the domain experts, computer scientists create the models and apply the analysis techniques to those models. When a defect is found in a process model, it needs to be examined carefully to determine if the problem is associated with our representation of the process, with any analysis artifacts, or if it is indeed an error in the process. If it is the latter, then domain experts propose modifications to the processes, often with the help of computer scientists who can explain the source of the problem and point out alternative solutions. The model of the modified process is then carefully reevaluated to assure that it has successfully dealt with the uncovered error and has not introduced new errors that violate the current stated requirements. With this approach, several alternative models can be considered and evaluated before a decision is made about the way the actual process should be modified. We have already had some experiences in which process model improvements were translated almost immediately into changes to actual medical processes. Thus, this approach provides a technological basis for process improvement applied not just to healthcare systems but also more broadly to other classes of human-intensive systems.

The medical community has tried to model and evaluate its processes for decades, and there is an industry devoted to helping hospitals to do so. Some of the medical professionals involved in our project had experiences with such efforts, which made them skeptical about the value of process modeling. But the greater specification breath and detail of our process modeling approach and the feedback obtained from rigorous analysis have led to valuable insights and process improvements so that they now feel the effort needed to create these models is warranted. The insights gained and the benefits of this approach have been described from both the medical perspective [12, 13], and from the computer science perspective [5-7, 20]. Here we present some observations that derive from our experiences using software engineering modeling and analysis techniques to create, validate, and improve medical processes.

Our project has been using specific software technologies to gain a deep understanding of medical processes and the nature of medical process improvement problems. Specifically, we are using the Little-JIL [2] process definition language to model the processes, the PROPEL [8, 21] property requirements engineering system to help elucidate and represent properties, and the FLAVERS [11] and SPIN [14] finite-state verification systems to detect defects in the Little-JIL process definitions and in the actual medical processes. Our recent work has also suggested that other analysis techniques, such as simulation and fault-tree analysis, might yield useful results. In this paper, we refer to specific features of these technologies, but use them as the basis for observations intended to be more broadly applicable to other technologies. While we have found the technologies that we have used to be effective in some ways, our experiences with them are more intended to form the basis for suggestions about the fundamental needs in this domain and about requirements for better technologies.

2.1 Modeling Processes

Process Modeling Language: Our use of the Little-JIL process definition language in this project has suggested the importance of certain features such as ***support for abstraction, hierarchical decomposition, concurrency, exception handling, and operations designed to support the flexibility that human agents expect for doing their tasks*** (e.g., non-deterministic choice). In addition, ***it is important that the language have well-defined semantics so that process models described in the language can be rigorously analyzed***.

To illustrate the kind of detailed modeling we are employing in this project, Figure 1 depicts a Little-JIL definition (where the term "definition" is used by Little-JIL to denote the more detailed, semantically rich models that can result) addressing a portion of a chemotherapy process. This figure and the ensuing description are only intended to give the reader a sense of the language; full details about the language can be found elsewhere [23]. A Little-JIL definition is centered on a coordination diagram of the process, such as the one shown in Figure 1, which defines the hierarchical task decomposition of the task being modeled. This hierarchical decomposition view of the process is a relatively natural structural representation that, nevertheless at times, can obscure some of the complicated control flow that is also being represented.

The coordination diagram in Figure 1 is a view of only the top-level process tasks, represented as steps (denoted in the diagram as nodes, called step bars). In this figure, the root step is

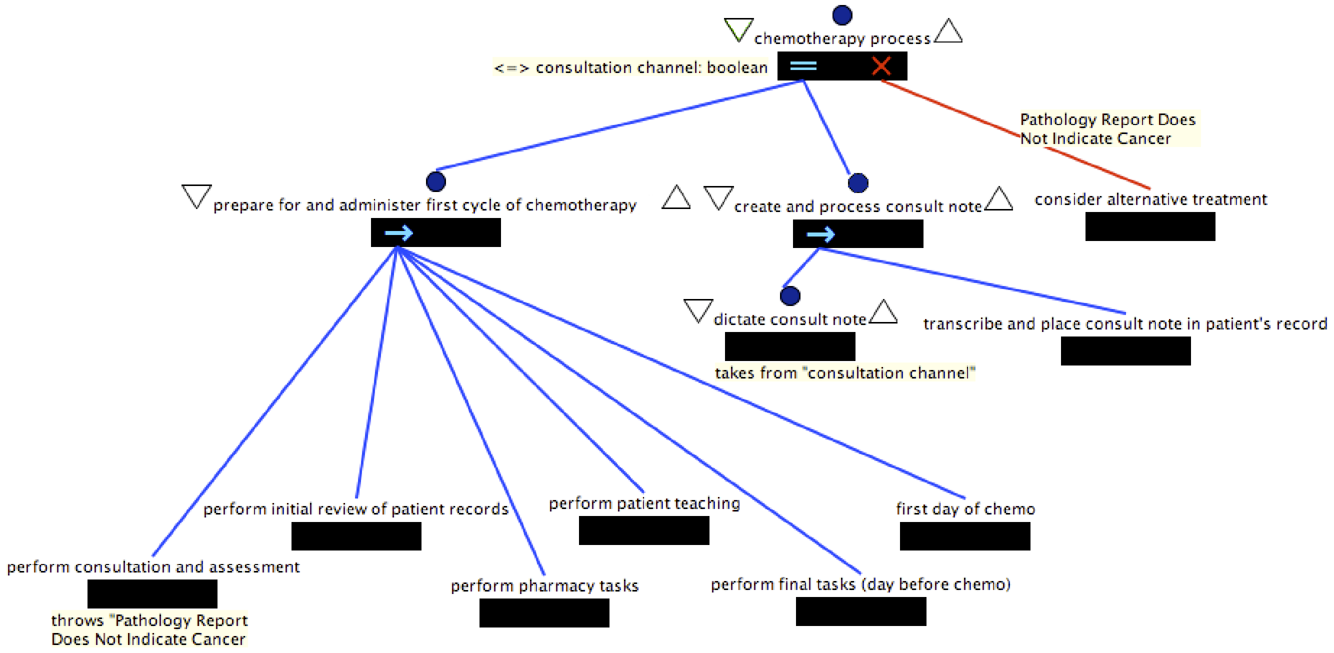

chemotherapy process
<=> consultation channel: boolean
Pathology Report Does Not Indicate Cancer
prepare for and administer first cycle of chemotherapy
create and process consult note
consider alternative treatment
dictate consult note
transcribe and place consult note in patient's record
takes from "consultation channel"
perform initial review of patient records
perform patient teaching
first day of chemo
perform consultation and assessment
perform pharmacy tasks
perform final tasks (day before chemo)
throws "Pathology Report Does Not Indicate Cancer

Figure 1. Top-level chemotherapy process definition in Little-JIL.

decomposed into two substeps executing in parallel (indicated by the equal sign in the step bar). Each substep can be further decomposed, or elaborated, down to the leaf steps, for which the process definer is unable to, or uninterested in, providing further detail. The first substep, `prepare for and administer first cycle of chemotherapy`, of the root step `chemotherapy process`, is decomposed into six substeps to be executed in sequence (indicated by the arrow pointing to the right in the step bar). Each step includes a specification of the type of agent, that is the type of participant who is to perform the tasks associated with each step, such as the type of doctor or nurse, hardware device, or software application. Specification of the agents, resources, artifacts, and some other features of the language as well as further decomposition of the steps in the diagram are not shown here.

Figure 1 also shows that the root step `chemotherapy process` has a substep `consider alternative treatment` that acts as an exception handler (indicated by the ``X" on the `chemotherapy process` step bar to which the step `consider alternative treatment` is connected). In the step `perform consultation and assessment`, if the doctor determines that the patient's pathology report does not indicate cancer, the `Pathology Report Does Not Indicate Cancer` exception is thrown (the elaboration of the `perform consultation and assessment` step is not shown). A thrown exception propagates control up the step decomposition tree until it reaches a matching handler. Thus, control is transferred to the exception handler step `consider alternative treatment` where appropriate action is specified.

While step-sequencing specifications provide control over the order of step execution, Little-JIL also enables specification of synchronization through such constructs as a channel. In this figure, a channel is used to specify that a doctor cannot dictate a consult note before evaluating the patient's condition. But, because a consult note is primarily used for billing and legal purposes, the doctor may choose to dictate the consult note at any time after evaluating the patient, for example while the tasks in `prepare for and administer first cycle of chemotherapy` are already underway. This step sequencing flexibility is captured by the coordination diagram in Figure 1, in which "consultation channel"" (represented iconically as an annotation attached to the circle above the root step, `chemotherapy process`), is counted on to deliver a consult note to the step `dictate consult note`, which cannot start until after the arrival of the consult note artifact. The `perform patient consultation step`, which is not shown, is a substep of the `perform consultation and assessment step` and is the source of this artifact. Thus, this example shows that the `dictate consult note` step can potentially execute in parallel with tasks in `prepare for and administer first cycle of chemotherapy`, but only after receiving the appropriate paperwork.

Process Modeling and Elicitation: ***Carefully modeling processes leads to better understanding about those processes.*** It was not unusual to find that the medical professionals did not fully understand the processes in which they were participants. Usually they knew their tasks, but often had misunderstandings about what others involved in the process actually did or how artifacts were used. By understanding the process better, errors in the process sometimes become apparent. Thus, the activity of modeling a process often leads to the discovery of process errors and always leads to better understanding.

A multifaceted language that separates out the different issues that need to be addressed facilitates process elicitation. The Little-JIL language is multifaceted in that the step definition has many different aspects, each of which can be considered separately. Thus, for example, in defining a step in the process, one has to consider the preconditions, the postconditions, the exceptions that could be thrown or handled by the step, the artifacts that are input to or output from the step, resources that might be requested or released, which agents should execute the step, the substeps that comprise the step, and the order in which these substeps should be executed. It is not necessary to consider all of these facets at once. In our project, a first pass was made at understanding and representing the process only involving step decomposition and control flow. Later passes through the process would then address other facets. Adding these additional facets, however, often resulted in changes to the overall step decomposition and control flow. Such changes are unavoidable, since as noted above, the domain experts were often not sure of the details of the process and had to consult with others involved in the process work or reevaluate their own process activities.

From the above list of facets, it is clear that Little-JIL supports the specification of a relatively broad range of semantic features of a process. This proved to be of considerable importance, as we found that the medical processes we wanted to define required strong semantic support for specifying such aspects as concurrency, exception handling, scoping, late binding, and flexible control flow that supported the freedom of choice often desired by human agents. Thus, the language supported detailed specification of how exceptional conditions are identified and handled, how parallel tasks must be synchronized, and how the assignment of personnel to tasks is indeed late-bound. On the other hand, the language did not insist that an ordering be imposed on selecting which substep to do next, if such an ordering was not actually required in the real process. This flexibility allowed us to define processes that more closely and completely modeled how they are actually performed. Having more accurate, complete, and detailed process definitions allowed for more meaningful and accurate analysis, as described further in the next subsection.

Abstraction and hierarchical decomposition facilitates developing the process models incrementally. The process modelers had to continually make choices about how many levels to which to decompose a task and the level of abstraction or granularity of that decomposition. Abstraction allows the activities associated with a task to be conceptualized. When it turned out that more detail was required, some tasks were then further decomposed so that these details could be elaborated. This allowed for incremental development of the model, and as discussed in the next subsection, allowed for incremental analysis, which provided incremental feedback.

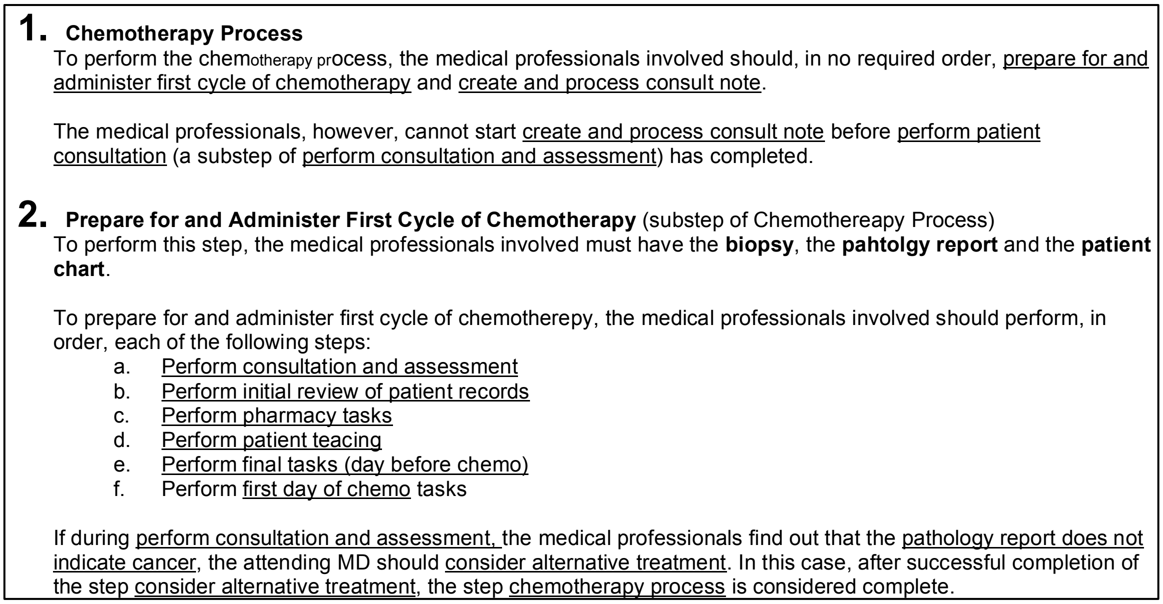
Process Comprehensibility: With the inclusion of the different facets described above, the detailed process definitions quickly became large and complex. Even seemingly simple processes, such as "verify a patient's ID", became unexpectedly large when the many variations of the process were defined. Some of our processes involved hundreds of steps, with all the facets of each step completely defined. To assist with this modeling and with comprehension, abstraction and hierarchical decomposition are extremely important. Any step in Little-JIL can be referenced many times, very much as though a step is a method call. A step, therefore, becomes an abstract representation of its definition, which for a non-leaf step is based on its hierarchical decomposition. Computer scientists are familiar and comfortable with these concepts, but we found that the domain experts had to be taught these concepts and their benefits.

Step decomposition in Little-JIL is a significant aid in understanding the basic breakdown of a task, but can also be somewhat misleading since the flow of control in Little-JIL is superimposed on the step decomposition view. For example, a parallel step may have several children, each of which can be further elaborated. This may appear to define only a simple task decomposition, but it also actually specifies that the execution of the descendant steps of one subtree can be interleaved with the execution of the descendants of the others. Thus on the positive side, Little-JIL succinctly and, from some perspectives, clearly represents these complicated potential traces. On the negative side, few domain experts (and probably many computer scientists) could be expected to completely grasp the complexity of what is being described.

The process definitions needed to be reviewed carefully and repeatedly by the medical professionals to assure that the definitions represented the actually processes appropriately. Thus, it is important that domain experts be able to understand, although it is probably not necessary that they be able to develop, the process definitions. Little-JIL coordination diagrams provide a visual spatial representation, as shown in Figure 1, that shows the step decomposition. The editing tool used to develop these diagrams can assist in efforts to understand process definitions by allowing the viewer to select the facets that are to be explicitly shown and by providing wizards that furnish more detailed information about steps and check for simple well-formedness. On the one hand, we were pleasantly surprised at how well some of the medical professionals learned the process definition language and were able to comprehend the Little-JIL process definitions. On the other hand, this comprehension was sometimes superficial. Thus, additional support is needed for domain experts to understand the process definitions. There are many alternative representations to help with model understanding that should be explored. For example, a role based view that shows the activities for a type of agent, such as the triage nurse, might be effective. We have experimented with creating a natural language, textual representation of the process definition. An example of a hyperlinked, textual representation for part of the process shown in Figure 1 is shown in Figure 2. As with the spatial representation, the viewer should be able to determine which facets of the process definition should be described and have some control over customizing the phrases that are used. Although a textual description did not address all the concerns about comprehensibility, the medical professionals greatly appreciated having a textual representation available for review.

2.2. Analyzing Processes

Analysis is a cornerstone of our approach. In fact, ***if the system is interesting enough to warrant being modeled then the model is probably complex enough to warrant careful scrutiny by rigorous and automated analysis techniques***. Without such scrutiny one should have serious concerns about the validity of the model and any decisions made based on that model. Thus as mentioned above, to support rigorous analysis, the semantics of the modeling language must be formally and precisely defined.



1. Chemotherapy Process

To perform the chemotherapy process, the medical professionals involved should, in no required order, prepare for and administer first cycle of chemotherapy and create and process consult note.

The medical professionals, however, cannot start create and process consult note before perform patient consultation (a substep of perform consultation and assessment) has completed.

2. Prepare for and Administer First Cycle of Chemotherapy (substep of Chemothereapy Process)

To perform this step, the medical professionals involved must have the **biopsy**, the **pahtolgy report** and the **patient chart**.

To prepare for and administer first cycle of chemoterepy, the medical professionals involved should perform, in order, each of the following steps:

a. Perform consultation and assessment
b. Perform initial review of patient records
c. Perform pharmacy tasks
d. Perform patient teacing
e. Perform final tasks (day before chemo)
f. Perform first day of chemo tasks

If during perform consultation and assessment, the medical professionals find out that the pathology report does not indicate cancer, the attending MD should consider alternative treatment. In this case, after successful completion of the step consider alternative treatment, the step chemotherapy process is considered complete.

Figure 2. Hyperlinked, textual view of part of the process shown in Figure 1.

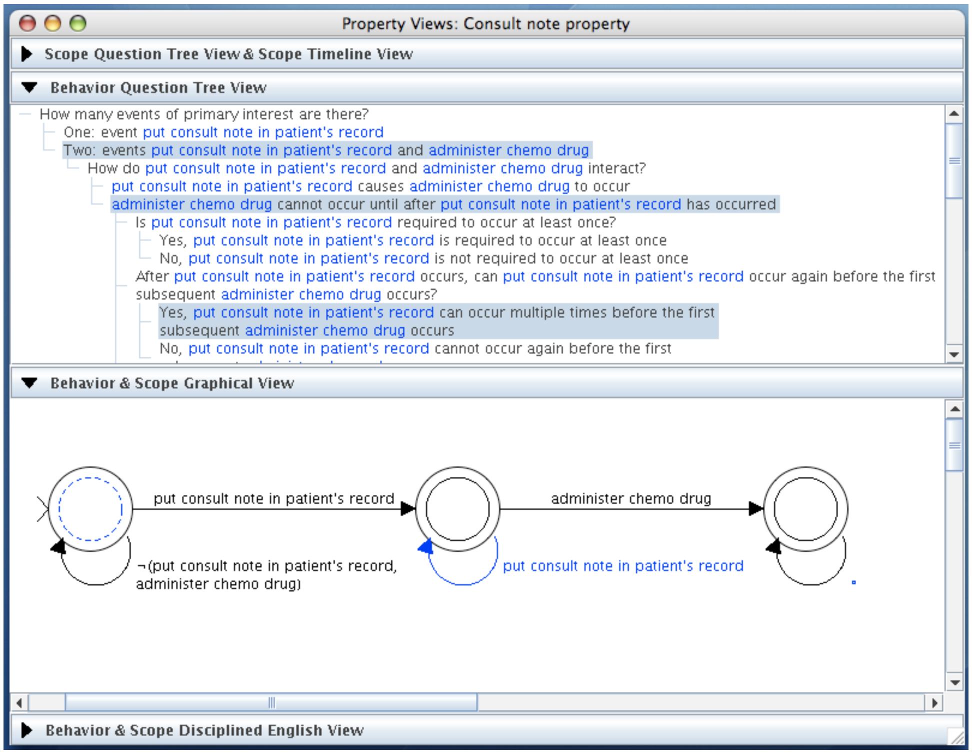
Even with careful analysis, complex processes represented by detailed models are bound to contain defects, just as most interesting programs are bound to contain defects. Since we are creating models on which to base decisions and further reasoning, enough to justify this trust. As the models are repeatedly validated using a range of analysis techniques, we increase our confidence in their accuracy. Moreover, if decisions made using the models then fail to provide the expected results, that too is a form of validation that should result in carefully scrutiny to determine the cause and subsequent improvement to the model. Thus, users of the model must recognize this limitation and evaluate recommendations derived from the model carefully, especially when dealing with life-critical processes, such as many medical procedures.

The analyses that we have been considering include finite-state verification to determine if all traces through a model adhere to properties that indicate the legal sequences of events (e.g., [11, 14]), fault-tree analysis to reveal vulnerabilities if steps in the process are not executed appropriately [22], and discrete-event simulation [17] to determine the aggregate behavior after a large number of traces have been executed. These are by no means all the kinds of analyses that should be considered, but each is substantially different and provides distinctive kinds of feedback. Since we have the most experience with finite-state verification, here we emphasize our observations from those experiences. A more thorough description of our experiences applying finite-state verification to medical processes is given in [5].

Property Specification: Before applying finite-state verification, we first needed to determine the properties of the system that should be evaluated. We usually started with medical guidelines that we first restated as high-level requirement statements, being careful to use consistent terminology. For example, "administer chemo" might have a different meaning depending on the part of the process that is being described by the guidelines. These requirements are usually described at such a high-level of abstraction that they are process-independent, meaning that a wide range of approaches could be used to satisfy them. Although this makes them generally applicable in many different settings, it is usually difficult to determine what is actually required for them to be satisfied. For example, a high-level generic property for most medical processes is: "Do the right procedure, on the right patient, at the right time."

Working with the medical professionals, we translated each high-level property into a set of more specific and measurable requirement statements. Determining if it is the "right patient", for example, might require checking that the name on the ordered medical procedure matches the name on the wrist-band ID, matches the name on the medical chart, and matches the name and date of birth provided by the patient (assuming the patient is conscious and speaks and understands the same language as the medical professional checking the ID). For this second level of refinement, we were also careful to use the terms defined in a glossary, adding terms when necessary.

These refined requirements were still not precise enough to form the basis for verifying the processes, however. The patient ID verification example, raises questions such as "when does the patient ID need to be checked?", "do all three checks have to happen in any particular order?", and "can other events happen in between checking the patient chart and checking the patient ID?" The Propel system was designed to help elicit these types of details from domain experts and then to represent them as a finite-state automaton that can serve as the basis for finite-state verification. These low-level, detailed, and narrowly focused requirement specifications are frequently called properties. Before verification could be done using these properties, the events mentioned in the properties had to be mapped to events in the process.


Property Views: Consult note property
Scope Question Tree View & Scope Timeline View
Behavior Question Tree View
Behavior & Scope Graphical View
put consult note in patient's record
administer chemo drug
¬(put consult note in patient's record,
administer chemo drug)
put consult note in patient's record
Behavior & Scope Disciplined English View

```
How many events of primary interest are there?
  One: event put consult note in patient's record
  Two: events put consult note in patient's record and administer chemo drug
    How do put consult note in patient's record and administer chemo drug interact?
      put consult note in patient's record causes administer chemo drug to occur
      administer chemo drug cannot occur until after put consult note in patient's record has occurred
        Is put consult note in patient's record required to occur at least once?
          Yes, put consult note in patient's record is required to occur at least once
          No, put consult note in patient's record is not required to occur at least once
        After put consult note in patient's record occurs, can put consult note in patient's record occur again before the first
        subsequent administer chemo drug occurs?
          Yes, put consult note in patient's record can occur multiple times before the first
          subsequent administer chemo drug occurs
          No, put consult note in patient's record cannot occur again before the first
```

Figure 3. Screen shot of the PROPEL question tree and finite-state automaton templates used during the development of a property.

This mapping was usually straightforward, but needed to be done with care, since an event in the property might map to different events in the process definition. Taken together, these properties and the bindings of event names to step names provided a detailed and rather process-specific description of the overall requirements for the process.

To represent the properties, Propel provides templates for commonly occurring verification property patterns [10]. These templates explicitly indicate the options that need to be considered for each pattern. In some ways, this is similar to the facets of the Little-JIL process modeling language since, like the facets, they remind the user of the many different concerns that must be taken into consideration. Propel provides the specifier with three alternative representations of the templates: disciplined, natural language text where the options are represented as phrase choices; finite-state automata graphs where options are represented by optional transitions, labels, and accepting states; and by question trees that first select the appropriate pattern based on answers to a few initial questions, but then continue to pose questions about all the options associated with the selected pattern template. Figure 3 provides an example of the finite-state automaton view and the question tree view for a simple property that was developed to assure that there is a signed consult note in the patient's record before chemotherapy is administered.

The medical professionals initially had difficulty understanding the difference between a process model, an operational view, and a requirement or property specification, a goal-oriented view. Moreover, the medical professionals tended to think in terms of "war stories" about what went wrong. We could sometimes map such a story to an appropriate set of properties. More work is definitely needed to determine how to better exploit these war stories, which are examples of scenarios that led to process errors.

Developing the properties provided valuable feedback about the current process model. We usually started developing the process models before the properties, based on the medical guidelines and the information provided by the medical experts who were working with us. We made note, however, of any requirements that were mentioned during this process elicitation. Those requirements, plus existing guidelines or protocols, were the initial set of high-level requirements. We also asked the medical professionals to suggest other, perhaps unstated but important, properties. In these discussions, it was not uncommon for medical professionals to propose requirements about details that were not even represented in the current process models, even though the medical professionals making the suggestion might have been working with us for months on developing these models. Clearly, concentrating on the requirements provided a different perspective, one focused on the intent of the process. We decided that if a requirement was important enough to be stated, that the models should be developed to the point where the tasks relevant to that requirement are actually represented and can be reasoned about during analysis. Thus, for example, we usually decided not to model the low-level details about how a form is to be filled out. On the other hand, since patient ID errors are common and often serious, details about how to check patient ID were added to the process models after requirements about this aspect of the process arose. Thus, what was deemed important by the domain experts determined the scope and granularity of the process models and the requirements for those models.

The properties that we specified and verified were primarily concerned that specific sequences of events did (or did not) occur. Occasionally we encountered a few properties concerned with state information, which, as is usually done, we dealt with by creating events that set or check a value. We encountered few properties that were concerned with timing events. One such example is the need to use or return a unit of blood within a set time period. There were a number of properties, however, that were concerned with efficiency, since inefficiency can impact patient safety. For example, a property might be that there is a check on the patient's ID before a critical procedure is performed. On the other hand, we might also want to check that no more that "N" such checks are performed before the procedure, since processes that have excessive checks are inefficient (and ineffective since the medical professionals may become lax about performing checks if they are deemed to be excessive.) In [5] an example with a richer sequence of events is used to determine if the process is inefficient. This property was based on a nurse's intimate knowledge of the process and where inefficiencies often arise.

Issues surrounding the ***specification and handling of exceptions have been particularly interesting, and often problematic***. Most of the errors that we have found in the process models and in the processes themselves involve exceptions. This is not surprising since studies have shown that errors are most likely to occur during the handling of exceptional cases. Medical guidelines, however, often do not even indicate what is to be done in such situations. This leads to variation in how medical professionals respond to these situations and, consequently, is more likely to lead to errors. From a technological point of view, the analysis tools that we used should be significantly improved to handle exceptional cases better. For example, often the properties were wrong because exceptional cases were not taken into account. That is, the property was stating what was expected only if no exceptions arose. Better support for indicating when exceptional and non-exceptional situations are to be considered in the property specifications should be explored.

Finite-state Verification: Finite-state verification problems are known to often explode in size, making it impractical to analyze large systems. Thus, we employed a number of optimization techniques when creating the internal representation of the Little-JIL process definitions. These optimizations are conservative but often reduce the size of the model by introducing some imprecision. The consequences of this imprecision are that the counter example traces through the internal model that violate the property may not correspond to actual executable traces through the process model. Thus, they are a false positive or spurious indication of an error. It is interesting to note that in our analysis of Little-JIL processes, spurious error reports hardly ever occurred. We believe that this is because this process language mostly describes control and event flow and does not represent compound objects or references to such objects using aliasing, which are known to degrade the results of static analysis. Moreover, unlike programs, most of the traces through our process models tended to correspond to actual executable behavior. Although this is a known benefit associated with analyzing high-level designs, it is interesting to see that it also applies to the detailed models provided by a process definition language, such as Little-JIL.

Using finite-state verification, we mostly found errors in the process models, as opposed to errors in the actual processes themselves. Although this might seem somewhat disappointing, as noted above, validating the models is an important part of process improvement. Before process models can be used for decision-making, they should be carefully validated. Using analysis techniques, we found numerous defects in our process models, but we also found some interesting and subtle errors in the actual processes that could have serious consequences. Analysis of the process models were also useful in helping to determine the causes of these errors and in evaluating alternative potential solutions for correcting them.

The set of property specifications associated with a process model continued to grow as the process was modified, as defects were found in the process models, and as errors were found in the processes themselves or in the clinical setting. This set of properties was invaluable. It provided a baseline set of requirements against which to verify the process models after any change. As this collection grew, we gained more confidence in the requirements and in the processes that adhered to these requirements. We suspect that domain experts will be more willing to consider process improvements knowing there is a detailed model of the process and its properties that can be used to evaluate those improvements,

Other analyses: As mentioned above, we intend to use the process models as the basis for a range of analysis techniques. Two techniques that we have initially explored are fault-tree analysis and discrete-event simulation. Fault-tree analysis creates a tree representation of how a hazard could occur in terms of what would need to fail or be faulty. The trees are then translated into Boolean flow equations that can be evaluated to determine minimum cut sets. Each minimum cut set indicates the collection of failures that would have to occur together for the hazard to arise. Historically fault trees have been used in traditional engineering disciplines such as mechanical engineering (although Leveson [3, 16] and others has also explored their use in systems engineering). In this prior work, fault trees tended to be large and were usually created by a team of safety experts. One of the drawbacks of this approach is that it is difficult to be sure that fault trees created in this way are sufficiently complete and accurate. We have demonstrated that fault-trees can be derived automatically from Little-JIL process definitions, using a template for each step kind and facet of the language [4]. The resulting fault trees are surprisingly large and complicated. Although one could argue that the fault trees derived from a process model may also be inaccurate if the process model is inaccurate, the process model is significantly simpler than the derived fault tree. Moreover, the process model should have first undergone rigorous validation before the corresponding fault-tree was generated and analyzed. In addition, a single, carefully validated process model can be used to derive a very large number of fault trees, namely a different one for each hazard to be studied. To give a sense of the complexity and size of these trees, a fault tree for a single hazard derived from a blood transfusion process is given in Figure 4.

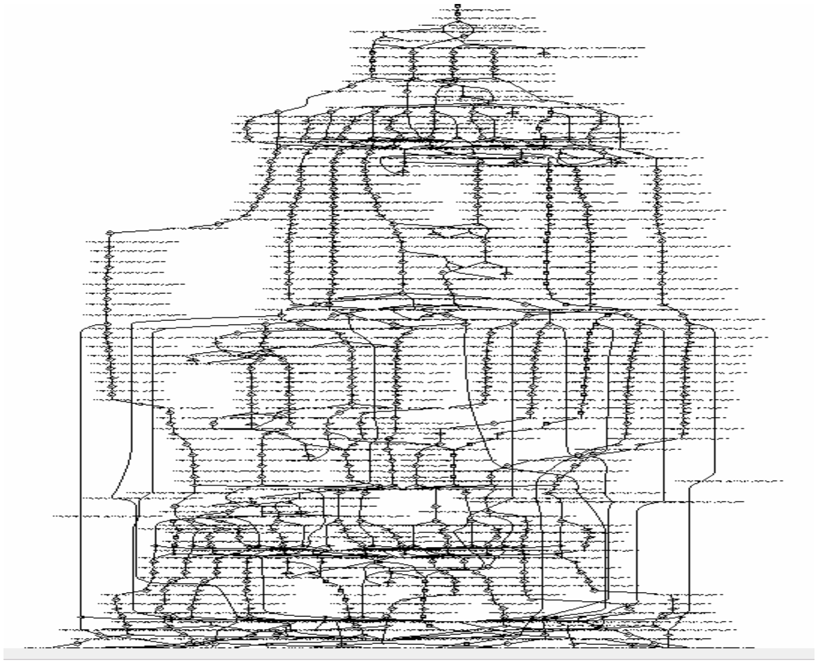


Figure 4. Example of a fault tree for a single hazard derived from a medical process.

We have also been experimenting with using the process models to drive discrete-event simulations. Simulations can be used to evaluate performance and efficiency. For our emergency room case study, one of the issues is how to determine the best resource mix that delivers optimal flow and minimal waiting times. For example, would it be better to hire more nurses, more doctors, or add more beds to reduce waiting time for patients? Here again, we suspect that our more detailed and validated model will be a better foundation for considering these issues than the high-level models that are currently used for simulation.

3. Conclusion

We have described how software engineering technologies and approaches can be incorporated into a Deming Cycle of continuous improvement to medical processes. This approach entails creating a process model that is detailed and addresses a broad range of semantic issues, extensively analyzing that model to validate that it is a reasonable basis for making decisions about the real process, and then using the validated model to detect errors and other problems in the actual process, with the eventual goal of improving that process. This approach to process improvement is being evaluated by applying it to critical processes from the medical domain. Currently we are involved in three case studies, emergency room flow, blood transfusion, and chemotherapy administration, each of which has different characteristics and places different demands on the supporting technology that we are developing. This work has led to a number of observations that seem particularly important.

The choice of a modeling language should depend on what one wants to do with the model and the process. In our case, we want to reason about how medical errors can occur, and how to guard against them. To be the basis for definitive reasoning the process modeling language itself must have well-defined semantics. But if the reasoning is to be relevant to a real-world process, then the language must also be capable of capturing in detail all aspects of the way in which the process might be performed. This dictates the selection of a process modeling language that allows fine-grain details to be represented. But, the language also needs to support the specification of such semantics as complex control flow, concurrency, exception handling, and scoping. While being quite precise about such issues, the language must also provide for the flexibility desired by humans.

Needing to deal effectively with each of these semantic issues creates challenges for a process language. Exception handling, for example, is an important element in human-intensive systems, but specifying it adds complexity both to the process and to the process model. Often process descriptions are not clear about what exceptional situations might arise and how to deal with them if they do. In the medical domain, this can be a source of wide variation in behavior, which is undesirable and often leads to errors. Once the exceptional conditions and how to handle them are determined, this information needs to be represented in the model, which itself can be difficult to do with most process modeling languages. Even with a supportive language, such as Little-JIL, representing exceptions has proven to be tricky and error prone. Exceptions also complicate the specification of requirements and any analysis being applied. We found that they were a major source of errors in the models and in the actual processes.

Additional specification complexity is added when the model incorporates execution semantics. Our work has the eventual goal of using executing process definitions to provide coordination support and proactive guidance to humans. This should not be undertaken unless these process models have been extensively analyzed. Our experience has indicated, however, that there are considerable challenges in developing an executable process language that has sufficient clarity, semantic breadth, and capacity for detail.

In our opinion, if a process is complicated enough to warrant precise and detailed modeling then the accuracy of the model requires careful scrutiny. This is particularly true for detailed models, such as the models we are developing using Little-JIL. Our experience suggests that these sorts of detailed and complex process models should be developed incrementally so that high-level, more-abstract views of the process can be validated before more-detailed models are developed. The scope and granularity of the model should be determined by the questions the model is intended to address. There is no doubt that detailed models require more effort to develop and maintain, but provide more definitive, in-depth feedback (in other words, there is no free lunch). But, our work has been quite satisfying in that the detailed process models and the analysis that we have applied have indeed discovered errors in actual medical processes. Indeed every step in this approach, from process modeling, to property specification, to process model verification has led to the discovery of errors of one kind or another in the actual processes.

Future work: This work has already suggested many directions for future research. A number of these directions are suggested by attempts to use process models to introduce automation. As already noted, we would like to eventually use validated process models to guide medical professionals while they are actually executing their processes in a clinical setting. For example, a doctor's hand held device could indicate the current process status for each patient and highlight the most urgent items according to the most recent recommended protocols. There are human interface issues in how to represent this information on the handheld device so that it can be immediately understood and used. There are other interesting issues in how to determine and maintain coherence between the actual process and the process model.

One of the most frequently asked questions is what is the cost in terms of time and effort to elicit and evaluate a process. To date, all the process models were developed and analyzed concurrently with technology development and involved students who were learning about using that technology. It would be interesting to assess the cost of having trained computer scientists work with medical professionals on complex processes to experimentally evaluate the costs and error detection effectiveness. Related questions revolve around the generalizability of medical processes and the degree to which each hospital might have to create its own hospital-specific process models. We believe, that for well-designed process models, the customization of a general process to a particular hospital setting should mostly involve changes to the low-level process steps. This hypothesis needs to be evaluated. Moreover, there is the issue of how many processes would need to undergo such careful modeling and validation. In the medical domain, it has been suggested that the number of such processes may be surprisingly small, in which case it would not be infeasible to develop models of each of these if this approach were to be found to indeed reduce the number of medical errors and help improve the efficiency of medical care. Medical protocols change frequently, however, so at least the generic versions of these models would need to be updated regularly and then recustomized. Clearly there are interesting issues about how to do this efficiently and accurately. As noted above, process models could be used to provide guidance during real execution of the process. In addition, they seem to hold considerable promise as teaching aids. Both are important when processes are being changed frequently. It is currently difficult for medical professionals to stay up-to-date on the latest recommended protocols without such assistance. Providing updated process models that can provide on-line guidance would help address this problem.

Finally, we note that the proposed approach does not seem to be specific to any particular set of technologies or restricted to any particular domain. We have demonstrated its effectiveness by using a specific modeling language and set of analysis tools, but other languages and tools could be used as well to support process improvement. We have tried to indicate the requirements for these capabilities. The medical domain has proven to be an interesting and challenging domain, and an important one to address, but we believe the approach is applicable to many domains, especially those that rely importantly upon complex, human-intensive processes.

Going further, we now envision a new paradigm for system development and improvement that is driven by a detailed understanding and evaluation of a coordination model that provides the context in which application software and hardware devices will be used. Evaluation of this model could be used to derive context requirements for the software and hardware devices, and analysis techniques could subsequently be applied to determine the consistency among these requirements and the provided components. In the medical domain, this would allow the processes to be evaluated with respect to the medical devices and software systems that are employed. A preliminary description of such an approach is provided in [1]. To achieve this vision will require advances in process modeling, software analysis, and system safety. In today's world, processes, software, and hardware devices rarely operate in isolation from each other, and thus process improvement must be considered in this broader system context.

One of the most gratifying aspects of the research described here is its suggestion that the approaches, understandings, and technologies that the software engineering community has developed over the past few decades may have profoundly important impacts upon a very broad spectrum of other disciplines. We have already seen the potential of the work of our community to effect important improvements in medical care. We can see glimpses of applicability in such other domains as law, government, manufacturing, and fundamental scientific research. No less gratifying is the sense that grappling with the problems of these domains is enriching software engineering research by confronting our approaches and technologies with novel challenges that promise to ultimately improve the work in the software engineering domain as well.

4. ACKNOWLEDGMENTS

Many people have made contributions to the work described here. We would particularly like to thank Dave Brown, Lucinda

Cassels, Bin Chen, Stefan Christov, Rachel Cobleigh, Heather Conboy, Elizabeth Henneman, Philip Henneman, Wilson Mertens, and Sandy Wise,

5. REFERENCES

1. G. S. Avrunin, L. A. Clarke, E. A. Henneman and L. J. Osterweil Complex Medical Processes as Context for Embedded Systems. *ACM SIGBED Review, special issue on Workshop on Innovative Techniques for Certification of Embedded Systems*, *3* (4). 9-14.
2. A. G. Cass, B. S. Lerner, E. K. Mccall, L. J. Osterweil, S. M. Sutton Jr. and A. Wise, Little-JIL/Juliette: A Process Definition Language and Interpreter. in *22nd International Conference on Software Engineering*, (Limerick, Ireland, 2000), 754-758.
3. S. S. Cha, N. G. Leveson and T. J. Shimeall, Safety Verification in Murphy Using Fault Tree Analysis. in *10th International Conference on Software Engineering*, (Singapore, 1988), 377--386.
4. B. Chen, G. S. Avrunin, L. A. Clarke and L. J. Osterweil, Automatic Fault Tree Derivation from Little-JIL Process Definitions. in *Software Process Workshop and Process Simulation Workshop*, (Shanghai, China, 2006), Springer-Verlag LNCS, 150-158.
5. B. Chen, G. S. Avrunin, E. A. Henneman, L. A. Clarke, L. J. Osterweil and P. L. Henneman, Analyzing Medical Processes. in 30th *International Conference on Software Engineering*, (Leipzig, Germany, 2008), to appear.
6. S. C. Christov, B. Avrunin, G.S., Chen, B., Clarke, L. A., Osterweil, L.J., Brown, D., Cassells, L., Mertens, W., Rigorously Defining and Analyzing Medical Processes: An Experience Report, in *Workshop on Model-Oriented Trustworthy Health Information Systems (MOTHIS)*, (Nashville, TN, 2007). (to appear in LNCS Volume on Models in Software Engineering Workshops and Symposia at MoDELS 2007, Reports and Revised Selected Papers, Editor: Holger Giese)
7. L. A. Clarke, Y. Chen, G. S. Avrunin, B. Chen, R. Cobleigh, K. Frederick, E. A. Henneman and L. J. Osterweil, Process Programming to Support Medical Safety: A Case Study on Blood Transfusion. in *Software Process Workshop 2005*, (Beijing, China, 2005), Springer-Verlag, 347-359.
8. R. L. Cobleigh, G. S. Avrunin and L. A. Clarke, User Guidance for Creating Precise and Accessible Property Specifications. in *14th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, (Portland, OR, 2006), 208-218.
9. W. E. Deming *Out of the Crisis*. MIT Press, Cambridge, 1982.
10. M. B. Dwyer, G. S. Avrunin and J. C. Corbett, Patterns in Property Specifications for Finite-State Verification. in *21st International Conference on Software Engineering*, (Los Angeles, CA, 1999), 411-420.
11. M. B. Dwyer, L. A. Clarke, J. M. Cobleigh and G. Naumovich Flow Analysis for Verifying Properties of Concurrent Software Systems. *ACM Transactions on Software Engineering and Methodology*, *13* (4). 359-430.
12. E. A. Henneman, G. S. Avrunin, L. A. Clarke, L. J. Osterweil, C. J. Andrzejewski, K. Merrigan, R. Cobleigh, K. Frederick, E. Katz-Basset and P. L. Henneman. Increasing Patient Safety and Efficiency in Transfusion Therapy Using Formal Process Definitions. *Transfusion Medicine Reviews*, *21* (1). 49-57.
13. E. A. Henneman, R. Cobleigh, G. S. Avrunin, L. A. Clarke, L. J. Osterweil and P. L. Henneman Property Specification to Improve the Safety of the Blood Transfusion Process. *Transfusion Medicine Reviews*. to appear.
14. G. J. Holzmann *The SPIN Model Checker*. Addison-Wesley, 2004.
15. L. T. Kohn, J. M. Corrigan and M. S. Donaldson (eds.). *To Err is Human: Building a Safer Health System*. National Academy Press, Washington DC, 1999.
16. N. G. Leveson *Safeware: System Safety and Computers*. Addison-Wesley, 1995.
17. J. Misra Distributed Discrete Event Simulation. *ACM Computing Surveys*, *18* (1). 29-55.
18. L. J. Osterweil, Software Processes are Software, Too. in *Ninth International Conference on Software Engineering*, (Monterey, CA, 1987), IEEE Computer Society Press, 2-13.
19. L. J. Osterweil, Software Processes Are Software, Too, Revisited. in *19th International Conference on Software Engineering*, (Boston, MA, 1997), 540-558.
20. L. J. Osterweil, G. S. Avrunin, B. Chen, L. A. Clarke, R. L. Cobleigh, E. A. Henneman and P. L. Henneman, Engineering Medical Processes to Improve their Safety: An Experience Report. in *Proceedings of the IFIP Working Group 8.1 Working Conference on Situational Method Engineering: Fundamentals and Experiences (Method Engineering 2007)*, (Geneva, 2007), Springer-Verlag, 267--282.
21. R. L. Smith, G. S. Avrunin, L. A. Clarke and L. J. Osterweil, PROPEL: An Approach Supporting Property Elucidation. in *24th International Conference on Software Engineering*, (Orlando, FL, 2002), 11-21.
22. W. Vesely, F. Goldberg, N. Roberts and D. Haasl. Fault Tree Handbook U.S. Nuclear Regulatory Commission, Washington, D.C., 1981.
23. A. Wise. Little-JIL 1.5 Language Report, Department of Computer Science, University of Massachusetts, Amherst, 2006, 28.