
Doline Patchong  
Department of Computer Science  
University of North Dakota  
Grand Forks, North Dakota  
patchon@cs.und.edu

Emanuel S. Grant  
Department of Computer Science  
University of North Dakota  
grante@cs.und.edu

Douglas Olson  
Center for People and the Environment  
University of North Dakota  
Grand Forks, North Dakota  
olsen@umac.org

Abstract

Mission critical software systems must address concerns such as high reliability and availability, in order to maintain continuous functionality and minimize costly unscheduled downtime. The development of these systems occurs in an environment in which software maintenance is not the obviously main concern. Consequently, applying software engineering principles in developing such systems is often considered expensive, resource consuming and non-essential. Such practices make it necessary to compare software development that emphasizes software engineering principles against those that employ less standard approaches. One such comparison will focus on the differences in the amounts of effort needed in the creation and maintenance for both development approaches. This comparison will help in determining the benefits of one approach versus the other in a corporation’s decision-making process. Some aspects of comparison include modularity, readability, maintainability, extendibility, and reusability of the various components in the competing approaches.

1. Introduction

During the software development cycle [1, 2], the desired of the developer is the delivery of a working product. This focus may lead to neglecting some essential by-products of the development process, such as documentation, system maintainability, reusability,...the ities [3]. Mission critical systems [4] are particularly interesting illustrations of this phenomenon, as they constitute a milieu in which high reliability and availability requirements are crucial to their usage. Mission critical systems are required to maintain continuous functionality with minimal downtimes. The application of software engineering (SE) principles [5, 6] in the development processes in such environments is not only considered to be a low essential, but also time intensive and resources consuming. It is therefore fascinating to assess the real, rather than non-apparent, benefits of applying SE principles versus the non-application of these principles in such environments. In this report a plan and preliminary results of the evaluation of the application of SE principles on a mission critical system is presented. This evaluation will address the differences in the amounts of effort needed in the production and maintenance for strong software engineering approach versus a less strict approach to software development. Aspects of the evaluation include modularity, readability, maintainability, extensibility, evolvability and reusability of the various components.

The University of North Dakota (UND) Agricultural Camera (AgCam) project[1] provides an appropriate mission critical environment for this research. AgCam is a multi-spectral imaging system designed for use onboard the International Space Station. Data produced by AgCam will be used by researchers, farmers, ranchers, foresters, natural resource managers, K-12 teachers, tribal government officials, and others, who work together in learning communities with the Upper Midwest Aerospace

---

1 http://www.umac.org/content/sensors/sensors.shtml
Consortium (UMAC)\(^2\), a network of researchers at 8 universities in 5 states. Development of AgCam is providing students and faculty with extensive experiential learning opportunities in multidisciplinary team settings. AgCam science investigations and project management are led by the Northern Great Plains Center for People and the Environment (CP&E)\(^3\), located within the Odegard School of Aerospace Sciences\(^4\) at UND.

The AgCam project facilitates the evaluation of two software systems; the Flight Software System (FSS) and the Science Operation Center (SOC). The FSS was developed to operate two cameras on the International Space Station, with the main development concern being its mission critical requirements. The AgCam Mission Planning System (AMPS) modules, a sub-system of the SOC, had also been developed earlier – And like the Flight Software, the development goal was the fulfillment of the specific functional requirements, but without the software engineering principles in mind. SOC is intended to receive and process user’s requests from the Digital Northern Great Plains\(^5\) (DNGP). In order to satisfy these user requests AgCam sends commands and receives telemetry from the Flight Software through the National Aeronautics and Space Administration (NASA) Payload Operations Integration Center\(^6\) (POIC) Huntsville, Texas. Apart from being developed in a mission critical environment, another interesting argument to gauge the benefits of applying SE principles for the AgCam project is the particular conditions in which the project takes place. It is a long-term project that started in 2000 with a plan deployment on a shuttle launch in 2008. The software is scheduled to be operational for an indefinite lifetime. Completing the development of the AMPS and developing the SOC using the UML modeling and specification language in an object-oriented development methodology, allows for an investigation on the benefits of the application of SE principles in such environment.

1.1 Introducing UML

To apply SE principles to our case study, the AgCam project, we are using the Unified Modeling Language\(^7\) (UML), a non-proprietary standardized specification language for object-oriented software system modeling. UML offers an industry wide standard for development; suited particularly to domain-specific modeling and iterative development processes. The benefits of using UML include a professional design and documentation before the coding phase of the software development process. This allows all the stakeholders to know exactly what they get in advance. The models can be tested, verified and validated, thus preventing consequences of specifications errors in the final product. The UML also allows the easily identification of reusable code, which provides more efficiency to the development process and contributes to reducing the overall development cost. The models created help to a more effective maintenance. Another asset of the UML is the efficient training of new development team members, hence reducing their induction period. More generally, the UML provide an effective communication vehicle between internal (development teams) and external stakeholders.

UML consists of a set of diagrams to model specific views of a software system. The models can be classified into three sets, that is: Structure diagrams, such as class diagrams, component diagrams, composite structure diagrams, deployment diagrams, object diagrams and package diagrams; Behavior diagrams, such as activity diagrams, state machine diagrams and use case diagrams; and Interaction diagrams, which include communication diagrams, interaction overview diagrams, sequence diagrams and UML timing diagrams. The type of project dictates the models to be used. For AgCam, the Use

\(^2\) http://www.umac.org/
\(^3\) http://www.umac.org/content/ngp_env/env.shtml
\(^4\) http://www.aero.und.edu/f1_Home/index.php
\(^5\) http://www.umac.org/content/ngp_env/env.shtml
\(^6\) http://www.nasa.gov/audience/foreducators/k-4/features/F_People_Behind_the_Astronauts.html
\(^7\) http://www.omg.org
case diagrams, class diagrams and sequence diagrams where used as they apply to a business model. The use case diagrams display the relationships between actors, which are the users of the system, and use cases, which are the systems functional requirements. The class diagrams models class structure and content using design elements such as classes, packages and objects; they also display their relationships such as containment, inheritance, association and others. The sequence diagrams model the behavior of objects in a use case. Other diagrams, such as collaboration diagrams, which illustrate the relationship and interaction between software objects; state diagrams, which display the sequences of states that an object of an interaction goes through; and activity diagrams, which describe the workflow behavior of a system; might be used later to suit some specific research purpose.

The work being conducted is targeted at a qualitative evaluation of the benefits of conducting mission critical software development that enforces the main software engineering principles of: rigor and formality, separation of concern, modularity, abstraction, anticipation of change, generality, and incrementality [5]. In this work, we will focus on the principles of separation of concern, abstraction, anticipation of change, and incrementality. A set of metrics will be gathered on the work done in designing, implementing, and testing the SCO sub-system of AgCam. These metrics will be evaluated against those of the FSS. The FSS was developed without consideration of software engineering principles. From the evaluation, it is anticipated that the benefits of applying software engineering principles will be clearly demonstrated.

2. The Case Study: The Science Operation Center (SOC)
Our case study is SOC, a software system developed for the AgCam project of the University of North Dakota. The SOC will support the AgCam operations for an indefinite period of time, commencing in 2008. The research work pertain in implementing software engineering principles on SOC and collecting metrics to evaluate model-driven software engineering approach to traditional software development that does not emphasize SE principles. The SOC is a software system designed to operate two cameras, a red image camera and a near infrared (NIR) image camera. These cameras will be located on board the ISS, and will be directly operated by the FSS, another software system of the AgCam project. The FSS will be installed on a laptop that resides in the WORF rack of the ISS, and will receive commands from two sources, i.e. the crew graphical user interfaces (GUI) and the SOC. The crew GUI also resides on the WORF rack and provides an abridged version of the commands and telemetry to the ISS Crew members for operations on board the ISS. This, abridged set of commands, is to permit basic monitoring and operation of FSS for the crew embarked on the ISS.

The SOC will provide full commanding and telemetry to the FSS, for staff members at UMAC. SOC will be used to sent command to and receive telemetry data from the POIC, a division of NASA. The POIC receives all the commanding and telemetry data requests for all the payloads on board the ISS. It provides a single communication medium to the ISS, over a proprietary communication protocol. Finally, it forwards all the commanding and telemetry replies to the various payloads owners on the ground. The SOC will support two operational modes, which consist of an on-line and an off-line mode. This implies that this software should include the functionalities and constraints of a real-time software system. The SOC will be comprise of a number of modules. The following collaboration diagram illustrates these modules and their relationship.
2.1 SOC modules

The modules making up the SOC are the AgCam Mission Planning Station, the Commanding Station, the Telemetry Monitoring Station, the Telemetry Recording Station, the Telemetry Processing Station and the System Manager. A database system will facilitate the sharing of information between the various modules. The AgCam AMPS will receive image requests from the customers via the DNGP. The DNGP is an on-line graphic information system for archiving and delivering remote sensing images. AMPS will process these requests to determine their validity using the data provided by the NASA Trajectory and Operations Officer (TOPO) at the NASA Johnson Space Center via email. TOPO data provide information on the position of the ISS relative to the earth in order to determine the time and duration when the AgCam camera (ISS) will be in position to capture the requested images. Processing the users requests will result in an email to the users on whether or not their request are feasible, and if feasible, when the images of the requested surfaces will be taken. The AMPS will store successful images acquisition requests in the database in order the pass those details to the Commanding Station.

The Commanding Station’s mission is to send commands to the FSS to operate the cameras and maintain the whole system on the ISS. These operations include turning the designated camera or the pointing system on or off; define a pointing angle for the cameras and take images with the selected camera. The maintenance options include editing the command queue, where the command queue is a set of text files, where the text files are the commands to be executed by the flight software at a given
time. There are two types of commands, namely: the **Level1** commands and the **Level2** commands. The commands files are named after a time tag. This time correspond to the desired execution time for the Level1 commands. This execution time is set from the building of the command. For instance a `take-image` command, which is a Level1 command, will be executed at the time specified in the command file name. The Level2 commands will include in their file name their arrival time, that is, the time at which they are received on the Flight Software; hence they will be executed in a first-in-first-out order and might be paused if a Level1 commands needs to be executed. On the Commanding Station, the commands are stored in the database and uploaded to the POIC, when switching to the online operational mode. These on-line operational modes are assigned and managed by NASA. The POIC will forward the command queue received from the Commanding Station to the AgCam payload onboard the ISS.

The Telemetry Monitoring Station is designed to receive telemetry-monitoring data. These data provides information about the status of the AgCam payload on the ISS. They include information on electric current and voltages, various temperature censors, cameras power status, etc. This information is produced once per second by the FSS and sent to the Telemetry Monitoring Station via the POIC in a **Health & Status** packet.

The Telemetry Recording Station records image telemetry. The later are the successful image captures from the FSS. They are destined to the final user who initially requested the image. They are sent over the User Datagram Protocol (UDP) because they constitute a huge amount of data and will be faster received over this protocol, which eliminates the latency caused by the hand-check and all the acknowledgements in Transmission Control Protocol (TCP). The Telemetry Recording Station will receive these data from connection to the POIC on a dedicated port, this to avoid packet lost, in the absence of a reception control mechanism provided by the TCP protocol. All the packets will be recorded and processed later to reconstitute the actual images.

The Telemetry Processing Station will process the recorded image telemetry, to check for missing packets or segments, in order to proceed to the re-ordering of those missing data. The Telemetry Station will also deliver the final product to the initial user.

The System Manager is another module of the SOC with unique purpose, i.e. to monitor and maintain all the other various stations. Monitoring operations include checking the on and off status of the various station, as well as activating logged user sessions on the stations. There are two types of users for the SOC system –Lead Operators, with unrestricted privileges and Operators with restricted privileges on SOC. The maintenance operations include turning the various stations on and off as well as assigning their configurations parameters such as their IP addresses and the protocols on which they transfer and receive data.

### 3. Research Methodology

The research being carried with the SOC involves the building of the system from requirements analysis, design specification, coding, testing, and delivery and maintenance. In order to make a statement on the input of SE principles in the software development process of such an environment, Domain-Specific Models have been produced to capture system requirements and transform them to design solution. The chosen modeling tool is Rational Rose® by IBM®, and the development tools are Microsoft Visual C++® and Microsoft SQL 2005®. The modeling of the SOC permits three parts of the software to be developed concurrently; that is, the Graphical User Interfaces (GUI), the Application Programming Interface (API), and the Database (DB). NASA standards were adopted for

---

the GUI design. The following two sections, will discuss the modeling and development details for the API and the DB of SOC.

### 3.1 SOC API modeling

Two UML modeling tools were proposed for the AgCam SOC modeling phase: Microsoft Visio 2003 and IBM- Rational Rose. IBM Rational Rose Developer for Visual Studio is a full visual modeling environment based on the UML. It includes code generation support for the Visual C++ and Visual Basic 6 languages and support round-trip engineering activities. On the other hand, Visio 2003 is diagramming programs that can help create business and technical diagrams that document and organize complex ideas, processes, and systems. The Microsoft Office Visio 2003 SDK provides developers with tools and documentation to support custom solution development on the Visio 2003 platform using Microsoft Visual Studio 6.0 and Microsoft Visual Studio .NET 2003 development tools. As it clearly appears, Microsoft Visual Studio 6.0 does offer code generation features but no round trip engineering facility. This contributed to the choice of Rational Rose as the UML modeling tool for this project; this I prevision of the development phase where the basic framework for coding will be done automatically as well as the round trip engineering process. The importance of the round trip engineering lies in the updates made to the models when major corrections are done to the basic framework in the coding phase. With this, comes a version management concern, which is also provided automatically by Rational Rose.

In the research three diagrams were adopted for modeling; that is use case diagrams, class diagram and sequence diagrams. Twenty-six UML diagrams were used to model the SOC. The following diagram represents one of the use case diagrams of SOC: the Commanding Station use case diagram.

---

The Commanding Station Use-case diagram

---
The Commanding Station use case diagram depicts the relationship between the users and the use cases of the station. The users are the Operator and the Lead Operator. The arrows from these operators to the use cases describes their access privileges relative to the functionalities of the station. In occurrence, the Operator does not have access to ‘Edit/Build_Cmd_q’ use case, as there are no arrows mapping them. This diagram also includes a generalization arrow, which is the arrow between the Operator and the Lead Operator. This generalization arrow is to represent the relationship between both users. In this case, the arrow going from the Lead Operator to the Operator means that the Operator is a generalization of the Lead Operator, hence, the Lead Operator inherits all the privileges of the Operator; that is, the Lead Operator can realize all the use cases the Operator has access to, and as such, there is no need to have arrows directly relating the Lead Operator to those use cases.

The use cases represent the functionalities requirements for the Commanding Station. They are related to the operators by realization arrows, and to other use cases by labeled arrows; the label on these arrows indicating the relationship between them, such as <include>, which suppose that one of the use case calls the functions of the other. Later in the development phase, the use case might represent procedures, functions or programs containing both procedures and functions. Having an abstract of these functional requirements and their relationships with each other and with the users facilitates and enhances the development of the software and ease the maintenance by easing the reading of the codes. Additionally this approach emphasizes the modularity provided to the software system.

Class diagrams have been developed for each of the use cases. The class diagrams represent the objects necessary to perform a use case, as well as the relationship between these objects. For instance, in a class diagram for the ‘Edit/Build_Cmd_q’ use case, there are objects representing the Operator, the Lead Operator, the command entity and the command queue entity. Links binding the objects represents various association types between them. These include but are not limited to inheritance relationships and composition associations. Objects are schematized as classes. A class is depicted as a box composed of three elements: a class name in the first section, the class attributes in the second and the class methods in the third section of the box. The class name identifies the class. The class attributes are the information stored about an object. The class methods list all the operations or methods applied to the class. An example is the composition association between the command class and the command queue class; a black rhombus on the side of the command class materializes the fact that the command queue contains the command class elements. The notation ‘1..*’ on the side of the command class is an indication of the multiplicity between the two classes. The later informs that one command queue class may contain many commands objects.

An Illustration of two classes and their relationship
For each of the use cases, a sequence diagram was developed to model the flow of logic within the use case, between the objects involved and in a timely manner. As a sketch of the interaction between objects over the progression of time, the sequence diagrams are made up of the class objects, in named boxes, and their lifelines as vertically dashed lines under the class object. Horizontal arrows with messages names above them display the interaction between the objects. Opaque rectangles on top of the lifelines are activation boxes that lie in between the messages and their responses, to materialize the duration of a particular process in relation to the lifetime of the object. Sequence diagrams were particularly helpful to model the system behavior in the flow of realization of a use case.

3.2 SOC database modeling and development
The SOC database was modeled using Entity Relationship (ER) modeling [7]. The ER diagrams, obtained from the ER modeling, represent the structure of the data stored in a database; together with the constraints between them. Entities are discrete objects to be stored and the relationship captures how these objects are related to one another. The entities or tables are represented by rectangles tagged with the entity name. The fields or columns of the table are represented by ovals linked to the rectangle and tagged with their corresponding names. One of the fields might have an underlined name. The later will be the primary key for the table; that is, the values stored in that column will uniquely identify each record of the table. The relationships are drawn as diamonds, and might link one or may entities. The following diagram shows a part of the ER modeling of the SOC. It represents two tables, that is, the ‘UserRequest’ table, designed to store the user requests information, and the ‘OutputFile’ table, designed to store all the successful image capture requests information. The diagram shows their interaction with the AgCam AMPS process.

ER diagram of the AgCam Mission Planning: The Output File and the User Request table
Microsoft SQL 2005 has been chosen for the SOC database development. The Microsoft development environment and the fact that SQL is a standard for relational databases highly contributed to this choice. The modeled entities were implemented in tables. The SOC stations will insert, update, retrieve and delete data from these tables.

### 3.3 Evaluation metric

A Gantt chart for the SOC development was prepared as a first step in the project. This chart is used as a mechanism for recording the projected and actual times for each major task in the development cycle. The amount of time spent in each phase of the development cycle will be a key metric in evaluating the benefits of applying SE principles on this project. Other metrics being recorded are the number of components created in each sub-system, the often used lines of code per component, test cases generated and bugs discovered, etc. The evaluation process will include a comparison of these SOC matrices available metrics from the previously developed sub-system of AgCam.

### 4. Conclusion

Early results have indicated that there are qualitative benefits to applying model-driven techniques based on SE principles in developing systems such as AgCam. Some of these early benefits includes:

1. Greater understanding between the developers and managers of the project;
2. Better documentation of the system during development;
3. Greater confidence in the product being developed;
4. Smoother integration between the system components; and
5. Enhanced team work productivity.

It is anticipated that the evaluation of the metrics being recorded will add quantitative validation to the asserted qualitative benefits, at the end of the development process.

### Reference


