**MAJOR TOPIC: INTRODUCTION TO SOFTWARE ENGINEERING**

The term software engineering was coined in 1968 as a response to the desolate state of the art of developing quality software on time and within budget. The emphasis in software engineering is on both words, software and engineering. The problem of building and delivering complex software systems on time has been actively investigated and researched. Everything has been blamed, from the customer to the “soft” in software. What is the problem? One large problem to be careful about is Complexity and Change. Useful software systems are complex. To remain useful they need to evolve with the end users’ need and the target environment. In this course, we describe object-oriented techniques for conquering complex and changing software systems.

Here are some examples of software engineering failures:

1. Year 1900 bug

In 1992, Mary from Winona, Minnesota, received an invitation to attend a kindergarten. Mary was 104 at the time.

2. Leap-year bug

A supermarket was fined $1000 for having meat around 1 day too long, on February 29, 1988. The computer program printing the expiration date on the meat labels did not take into account that 1988 was a leap year.

3. Interface misuse

On April 10, 1990, in London, an underground train left the station without its driver. The driver had taped the button that started the train, relying on the system that prevented the train from moving when doors were open. The train operator had left his train to close a door which was stuck. When the door was finally shut, the train simply left.

4. Security

CERT (Computer Emergency Response Team) at the Software Engineering Institute is a government-funded organization for assisting the community in dealing with security incidents, vulnerabilities, and security know-how. The number of security incidents reported to CERT from the United States increased from 252 incidents in 1990 to 21,756 in 2000, and more than 40,000 incidents were reported in 2001.

5. Late and over budget

In 1995, bugs in the automated luggage system of the new Denver International Airport caused suitcases to be chewed up. The airport opened 16 months late, $3.2 billion over budget, with a mostly manual luggage system.

6. Late and over budget (2)

In 2002, the Swanick Air Traffic Control system covers all the enroute air traffic over England and Wales. The system was delivered substantially over budget (cost £623 million, originally planned at £350 million) and 6 years late. Two major upgrades of the system were delivered after training of the traffic controllers had started.

7. On-time delivery

After 18 months of development, a $200-million system was delivered to a health insurance company in Wisconsin in 1984. However, the system did not work correctly: $60 million in overpayments were issued. The system took 3 years to fix.

8. Unnecessary complexity

The C-17 cargo plane by McDonnell Douglas ran $500 million over budget because of problems with its avionics software. The C-17 included 19 onboard computers, 80 microprocessors, and 6 different programming languages.

9. It was involved in at least six accidents between 1985 and 1987, in which patients were given massive overdoses of radiation.[1]:425 Because of concurrent programming errors, it sometimes gave its patients radiation doses that were hundreds of times greater than normal, resulting in death or serious injury. These accidents highlighted the dangers of software control of safety-critical systems, and they have become a standard case study in health informatics and software engineering.

10. On 26 September 1983, the nuclear early-warning system of the Soviet Union reported the launch of multiple USAF Minuteman intercontinental ballistic missiles from bases in the United States. These missile attack warnings were correctly identified as a false alarm by Stanislav Yevgrafovich Petrov, an officer of the Soviet Air Defence Forces. This decision is seen as having prevented a retaliatory nuclear attack based on erroneous data on the United States and its NATO allies, which would have probably resulted in immediate escalation of the cold-war stalemate to a full-scale nuclear war. Investigation of the satellite warning system later confirmed that the system had malfunctioned.

11. On January 13, 2018, a false ballistic missile alert was issued via the Emergency Alert System and Commercial Mobile Alert System over television, radio, and cellphones in the U.S. state of Hawaii. The alert stated that there was an incoming ballistic missile threat to Hawaii, advised residents to seek shelter, and concluded "This is not a drill". The message was sent at 8:07 a.m. local time. A second message, sent 38 minutes later, described the first as a "false alarm".

Software systems are complex creations. They perform many functions; they are built to achieve many different, and often conflicting, objectives. Many systems are so hard to understand, even during their development phase, that they are never finished: these are called vaporware. Software development projects are subject to constant change. Thus the issues such as requirement changes, over budgeting, employee turnover etc. is normal and has to be dealt with. Software engineers are required to have expertise in all of these to do a better job.

What is Software Engineering?

Modeling: Software engineering is a modeling activity. Software engineers deal with complexity through modeling, by focusing at any one time on only the relevant details and ignoring everything else. In the course of development, software engineers build many different models of the system and of the application domain.

Problem-solving: Software engineering is a problem-solving activity. Models are used to search for an acceptable solution. This search is driven by experimentation. Software engineers do not have infinite resources and are constrained by budget and deadlines. Given the lack of a fundamental theory, they often have to rely on empirical methods to evaluate the benefits of different alternatives.

Knowledge Acquisition: Software engineering is a knowledge acquisition activity. In modeling the application and solution domain, software engineers collect data, organize it into information, and formalize it into knowledge. Knowledge acquisition is not sequential, as a single piece of additional data can invalidate complete models.

Rationale-driven Activity: Software engineering is a rationale-driven activity. When acquiring knowledge and making decisions about the system or its application domain, software engineers also need to capture the context in which decisions were made and the rationale behind these decisions. Rationale information, represented as a set of issue models, enables software engineers to understand the implication of a proposed change when revisiting a decision.

Some SE Concepts

In this section, we describe the main terms and concepts we use throughout the lectures. A Project, whose purpose is to develop a software system, is composed of a number of Activities. Each Activity is in turn composed of a number of Tasks. A Task consumes Resources and produces a Work Product. A Work Product can be either a System, a Model, or a Document. Resources are either Participants, Time, or Equipment. A graphical representation of these concepts is shown in Figure each rectangle represents a concept. The lines among the rectangles represent different relationships between the concepts. For example, the diamond shape indicates aggregation: a Project includes a number of Activities, which includes a number of Tasks. The triangle shape indicates a generalization relationship; Participants, Time, and Equipment are specific kinds of Resources. Figure 1-1 is represented in the Unified Modeling Language (UML) notation. We use UML throughout the book to represent models of software and other systems. Intuitively, you should be able to understand this diagram without full knowledge of the UML semantics. Similarly, you can also use UML diagrams when interacting with a client or a user, even though they may not have any knowledge of UML.



Participants and roles:

Developing a software system requires the collaboration of many people with different backgrounds and interests. The client orders and pays for the system. The developers construct the system. The project manager plans and budgets the project and coordinates the developers and the client. The end users are supported by the system. We refer to all the persons involved in the project as participants. We refer to a set of responsibilities in the project or the system as a role. A role is associated with a set of tasks and is assigned to a participant.

Systems and Models

We use the term system as a collection of interconnected parts. Modeling is a way to deal with complexity by ignoring irrelevant details. We use the term model to refer to any abstraction of the system.

Work Products

A work product is an artifact that is produced during the development, such as a document or a piece of software for other developers or for the client. We refer to a work product for the project’s internal consumption as an internal work product. We refer to a work product that must be delivered to a client as a deliverable.

Activities, Tasks, and Resources

An activity is a set of tasks that is performed toward a specific purpose. For example, requirements elicitation is an activity whose purpose is to define with the client what the system will do. Delivery is an activity whose purpose is to install the system at an operational location. Management is an activity whose purpose is to monitor and control the project such that it meets its goals (e.g., deadline, quality, budget). Activities can be composed of other activities. The delivery activity includes a software installation activity and an operator training activity. Activities are also sometimes called phases. A task represents an atomic unit of work that can be managed: A manager assigns it to a developer, the developer carries it out, and the manager monitors the progress and completion of the task. Tasks consume resources, result in work products, and depend on work products produced by other tasks. Resources are assets that are used to accomplish work. Resources include time, equipment, and labor. When planning a project, a manager breaks down the work into tasks and assigns them to resources.

Functional and Nonfunctional Requirements

Requirements specify a set of features that the system must have. A functional requirement is a specification of a function that the system must support, whereas a nonfunctional requirement is a constraint on the operation of the system that is not related directly to a function of the system.

Software Engineering Development Activities

Following are the technical activities associated with object-oriented software engineering. Development activities deal with the complexity by constructing and validating models of the application domain or the system. Development activities include

* Requirements Elicitation
* Analysis
* System Design
* Object Design
* Implementation
* Testing

Requirements Elicitation

During requirements elicitation, the client and developers define the purpose of the system. The result of this activity is a description of the system in terms of actors and use cases. Actors represent the external entities that interact with the system. Actors include roles such as end users, other computers the system needs to deal with (e.g., a central bank computer, a network), and the environment (e.g., a chemical process). Use cases are general sequences of events that describe all the possible actions between an actor and the system for a given piece of functionality.

Analysis

During analysis, developers aim to produce a model of the system that is correct, complete, consistent, and unambiguous. Developers transform the use cases produced during requirements elicitation into an object model that completely describes the system. During this activity, developers discover ambiguities and inconsistencies in the use case model that they resolve with the client. The result of analysis is a system model annotated with attributes, operations, and associations.

System Design

During system design, developers define the design goals of the project and decompose the system into smaller subsystems that can be realized by individual teams. Developers also select strategies for building the system, such as the hardware/software platform on which the system will run, the persistent data management strategy, the global control flow, the access control policy, and the handling of boundary conditions. The result of system design is a clear description of each of these strategies, a subsystem decomposition, and a deployment diagram representing the hardware/software mapping of the system. Whereas both analysis and system design produce models of the system under construction, only analysis deals with entities that the client can understand. System design deals with a much more refined model that includes many entities that are beyond the comprehension (and interest) of the client.

Object Design

During object design, developers define solution domain objects to bridge the gap between the analysis model and the hardware/software platform defined during system design. This includes precisely describing object and subsystem interfaces, selecting off-the-shelf components, restructuring the object model to attain design goals such as extensibility or understandability, and optimizing the object model for performance. The result of the object design activity is a detailed object model annotated with constraints and precise descriptions for each element.

Implementation

During implementation, developers translate the solution domain model into source code. This includes implementing the attributes and methods of each object and integrating all the objects such that they function as a single system. The implementation activity spans the gap between the detailed object design model and a complete set of source code files that can be compiled.

Testing

During testing, developers find differences between the system and its models by executing the system (or parts of it) with sample input data sets. During unit testing, developers compare the object design model with each object and subsystem. During integration testing, combinations of subsystems are integrated together and compared with the system design model. During system testing, typical and exception cases are run through the system and compared with the requirements model. The goal of testing is to discover as many faults as possible such that they can be repaired before the delivery of the system. The planning of test phases occurs in parallel to the other development activities: System tests are planned during requirements elicitation and analysis, integration tests are planned during system design, and unit tests are planned during object design.

Managing Software Development

Following are the activities involved in managing a software engineering project. Management activities focus on planning the project, monitoring its status, tracking changes, and coordinating resources such that a high-quality product is delivered on time and within budget. Management activities not only involve managers, but also most of the other project participants as well. Management activities include

* Rationale Management
* Software Configuration Management
* Project Management
* Software Life Cycle

Requirements Elicitation

During requirements elicitation, the client and developers define the purpose of the system. The result of this activity is a description of the system in terms of actors and use cases. Actors represent the external entities that interact with the system. Actors include roles such as end users, other computers the system needs to deal with (e.g., a central bank computer, a network), and the environment (e.g., a chemical process). Use cases are general sequences of events that describe all the possible actions between an actor and the system for a given piece of functionality.

Analysis

During analysis, developers aim to produce a model of the system that is correct, complete, consistent, and unambiguous. Developers transform the use cases produced during requirements elicitation into an object model that completely describes the system. During this activity, developers discover ambiguities and inconsistencies in the use case model that they resolve with the client. The result of analysis is a system model annotated with attributes, operations, and associations.

System Design

During system design, developers define the design goals of the project and decompose the system into smaller subsystems that can be realized by individual teams. Developers also select strategies for building the system, such as the hardware/software platform on which the system will run, the persistent data management strategy, the global control flow, the access control policy, and the handling of boundary conditions. The result of system design is a clear description of each of these strategies, a subsystem decomposition, and a deployment diagram representing the hardware/software mapping of the system. Whereas both analysis and system design produce models of the system under construction, only analysis deals with entities that the client can understand. System design deals with a much more refined model that includes many entities that are beyond the comprehension (and interest) of the client.

Object Design

During object design, developers define solution domain objects to bridge the gap between the analysis model and the hardware/software platform defined during system design. This includes precisely describing object and subsystem interfaces, selecting off-the-shelf components, restructuring the object model to attain design goals such as extensibility or understandability, and optimizing the object model for performance. The result of the object design activity is a detailed object model annotated with constraints and precise descriptions for each element.

Implementation

During implementation, developers translate the solution domain model into source code. This includes implementing the attributes and methods of each object and integrating all the objects such that they function as a single system. The implementation activity spans the gap between the detailed object design model and a complete set of source code files that can be compiled.

Testing

During testing, developers find differences between the system and its models by executing the system (or parts of it) with sample input data sets. During unit testing, developers compare the object design model with each object and subsystem. During integration testing, combinations of subsystems are integrated together and compared with the system design model. During system testing, typical and exception cases are run through the system and compared with the requirements model. The goal of testing is to discover as many faults as possible such that they can be repaired before the delivery of the system. The planning of test phases occurs in parallel to the other development activities: System tests are planned during requirements elicitation and analysis, integration tests are planned during system design, and unit tests are planned during object design.

Managing Software Development

Following are the activities involved in managing a software engineering project. Management activities focus on planning the project, monitoring its status, tracking changes, and coordinating resources such that a high-quality product is delivered on time and within budget. Management activities not only involve managers, but also most of the other project participants as well. Management activities include

• Rationale Management

• Software Configuration Management

• Project Management

• Software Life Cycle

Rationale Management

Rationale is the justification of decisions. Given a decision, its rationale includes the problem that it addresses, the alternatives that developers considered, the criteria that developers used to evaluate the alternatives, the debate developers went through to achieve consensus, and the decision. Rationale is the most important information developers need when changing the system. If a criterion changes, developers can reevaluate all decisions that depend on this criterion. If a new alternative becomes available, it can be compared with all the other alternatives that were already evaluated. If a decision is questioned, they can recover its rationale to justify it.

Software Configuration management

Software Configuration management enables developers to track changes. The system is represented as a number of configuration items that are independently revised. For each configuration item, its evolution is tracked as a series of versions. Selecting versions enables developers to roll back to a well-defined state of the system when a change fails. Configuration management also enables developers to control change. After a baseline has been defined, any change needs to be assessed and approved before being implemented. This enables management to ensure that the system is evolving according to project goals and that the number of problems introduced into the system is limited.

Project Management

Project management does not produce any artifact of its own. Instead, project management includes the oversight activities that ensure the delivery of a high-quality system on time and within budget. This includes planning and budgeting the project during negotiations with the client, hiring developers and organizing them into teams, monitoring the status of the project, and intervening when deviations occur.

Software Life Cycle

In this course, we describe software engineering as a modeling activity. Developers build models of the application and solution domains to deal with their complexity. By ignoring irrelevant details and focusing only on what is relevant to a specific issue, developers can more effectively resolve issues and answer questions. The process of developing software can also be viewed as a complex system with inputs, outputs, activities, and resources. It is not surprising, then, that the same modeling techniques applied to software artifacts are used for modeling software processes. A general model of the software development process is called a software life cycle.