	Title : CS4504 Study
	Student Name : Brian O Regan
	Student Number : 110707163
	Module : CS4504
	Exam Date: Thursday 1st May @ 09.30

2001 Autumn

Requirements Engineering

- Feasibility study
  - Find out if the current user needs be satisfied given the available technology and budget?
- Requirements analysis
  - Find out what system stakeholders require from the system
- Requirements definition
  - Define the requirements in a form understandable to the customer
- Requirements specification
  - Define the requirements in detail

Most Important & Less Important Examples

MI – in an area where life may be put at risk such as hospital systems (life support) or air traffic control systems. Financial systems such as the Stock Market where millions or billions could be lost if the system fails.

LI – Games, you might get a few complaints from users but nobody will die or lose money.

Software Risks

Scope – Requirements – Team - Software Architecture - Risk Management – Estimation - Quality Assurance - Executing the project - Communication - Environment – Epilogue

(QC STEERERS)

Scope

You have to have a vision statement for the project and commitment for the top management for your project. If you do not have that do not even think on starting your project.
An example of scope is “Go to the Moon” as set by President John F. Kennedy‘s on May 25, 1961
Make sure that you have a list of deliverables that are in scope and also that you have an out of scope list.

Requirements

You have to have business requirements for your project. This is a tricky one because you need to define how you will accept the business requirements. Things to be watching for when you deal with business requirements:

Are numbered. Helps traceability
There is no ambiguity in the requirements.
Are specific
There are no conflicting requirements
All stakeholders where involved during the requirement gathering sessions.

Team

Select the best that you can afford or are available. Do team building activities. Remember the Tuckman's stages of group development: Forming – Storming – Norming – Performing – Adjourning.

You goal is to create a cohesive team that can perform effectively and efficiently. There will be some frictions but is up to you to coach the team successfully out of a crisis.

The team have to know the technology they are going to use to build the software. The better your team, the better the possibilities of success for your project.

Software Architecture

Do the software architecture phase. If you use Scrum you can do this before you start your sprints or you can have your first sprint dedicated to software architecture.

Allowing time for architecture will save your team (and yourself) a lot of wasted time. Software architecture is a huge subject to be analyzed here. Involve a software architect. Software architects can depict the system is such a way that the system's behavior will be visible to the business and the implementation teams. The architecture is the primary carrier of system qualities such as performance, modifiability, and security under a unifying architectural vision. Developers see the details of the solution but many times fail to see the system as a whole.

Risk Management

Plan to have frequent (weekly) risk management meetings. You need to identify, assess and prioritize the risks of you project. Leaving your risks unmanaged is a way to ensure your project's failure.

Estimation

There is always the risk that you did not estimate correctly the effort. Always use more than one estimation method and triangulate. This is a difficult task but if you have performed the previous phases then you and your team have a very good understanding of the challenges of the project and you will be able to create a realistic estimation. You estimation should include all time needed to finish the project this means also time for meetings etc. Remember that various deployments take time and also that there will be bugs in the software. Again, this is a very big topic and it will be covered separately.

In addition according to Cerpa Narciso and Verner M.June. (2009), the 3 factors that contributed most for an in-house project failure are that the delivery date impacted the development process, project was underestimated, delivery date decision was made without appropriate requirements information.

Quality Assurance

Involve your quality team as early as possible. Make sure that they can create test cases out of the business requirements. Use a team of testers.

Quality Assurance will help you with validation and verification of your project. Validation ensures that "you built the right thing". Verification ensures that "you built it right".

Audit your development team that they are using the defined quality processes.

If you do not have a quality system in place you are in trouble. Standardizing processes of software projects enforces a unified way of project delivery in the organization.

The most commonly used quality frameworks are the ISO 12207, the Capability Maturity Model Integration (CMMI) and the project management body of knowledge (PMBoK). (Fairley, 2009, p.10)

Executing the project

The way you execute your project is key for the project success. There are different ways to execute your project; this depends on the methodology you are using. Here also experience helps. There are several keys to help you succeed.

Know the status of your team and resolve issues as soon as they occur. If you rely in weekly meeting then you are losing valuable time. Always state the duration, the purpose of a meeting and the expected outcome.
Be sure that you have a change request process in place. Changes will occur and you have to be ready to handle them.
At the end of each phase have a show and tell with the client.
Do team building activities.

There are many things to say for project execution. The above list is by no means complete. Project execution is a big subject that I will treat separately with articles in this blog.

Communication

Have everyone informed. Create a communication plan. Be proactive and not reactive. Create status reports and manage the expectations of the involved parties. Failure managing the expectations introduces unnecessary risks to your project. Communication can take up to 70% of your time.

Environment

Your project is not in isolation. It interacts with the surrounding environment. Changes in the environment affects your project. For example inability off subcontractors to deliver their work might affect your project's success.

Epilogue

Is the above list complete? No, it is not complete. It is a starting point to help you see some major areas that can hurt your project. With this article we only are scratching the tip of the iceberg. Managing software projects is really exciting and requires full commitment of your time and effort. Good luck!

Extreme Programming

- Extreme Programming emphasizes teamwork. Managers, customers, and developers are all equal partners in a collaborative team. Extreme Programming implements a simple, yet effective environment enabling teams to become highly productive. The team self-organizes around the problem to solve it as efficiently as possible.
- Extreme Programming improves a software project in five essential ways; communication, simplicity, feedback, respect, and courage. Extreme Programmers constantly communicate with their customers and fellow programmers. They keep their design simple and clean. They get feedback by testing their software starting on day one. They deliver the system to the customers as early as possible and implement changes as suggested.

Advantages

- Customer focus increase the chance that the software produced will actually meet the needs of the users
- The focus on small, incremental release decreases the risk on your project:
  - by showing that your approach works and
  - by putting functionality in the hands of your users, enabling them to provide timely feedback regarding your work.
- Continuous testing and integration helps to increase the quality of your work
- XP is attractive to programmers who normally are unwilling to adopt a software process, enabling your organization to manage its software efforts better.

Disadvantages

- XP is geared toward a single project, developed and maintained by a single team.
- XP is particularly vulnerable to "bad apple" developers who:
  - don't work well with others
  - who think they know it all, and/or
  - who are not willing to share their "superior" code
- XP will not work in an environment where a customer or manager insists on a complete specification or design before they begin programming.
- XP will not work in an environment where programmers are separated geographically.
- XP has not been proven to work with systems that have scalability issues (new applications must integrate into existing systems).

EP Suitable Examples

- Anything not working to a deadline (e.g. Microsoft Office or OS)

	Title : CS4504 Study
	Student Name : Brian O Regan
	Student Number : 110707163
	Module : CS4504
	Exam Date: Thursday 1st May @ 09.30

EX Less Suitable Examples

- Anything working on a deadline (e.g. Game industry with a release date for a game)

Syntax, Semantics and Rules of Inference of a Logic?

- Logic is a formal system in which the formulas or sentences have true or false values
- A logic includes:
  - **Syntax**: Specifies the symbols in the language and how they can be combined to form sentences. Hence facts about the world are represented as sentences in logic
  - **Semantics**: Specifies what facts in the world a sentence refers to. Hence, also specifies how you assign a truth value to a sentence based on its meaning in the world. A **fact** is a claim about the world, and may be true or false.
  - **Inference Procedure**: Mechanical method for computing (deriving) new (true) sentences from existing sentences

- Any 'formal system' can be considered a logic if it has:
  - a well-defined syntax;
  - a well-defined semantics; and
  - a well-defined proof-theory
- The syntax of a logic defines the syntactically acceptable objects of the language, which are properly called well-formed formulae (wff). (We shall just call them formulae.)
- The semantics of a logic associate each formula with a meaning.
- The proof theory is concerned with manipulating formulae according to certain rules.

Propositional Logic

The simplest, and most abstract logic we can study is called propositional logic.

Definition: A proposition is a statement that can be either true or false; it must be one or the other, and it cannot be both.

EXAMPLES. The following is a proposition - 1 + 1 = 2, whereas the following is not - 2 + 3 = 4

Let 1 = p, 2 = q and 3 = r

Syntax Example:

1. $p \wedge q \Rightarrow r$
2. $p \wedge q \vee r = r$

Semantics Example:

1. p and q is equal to r, has a True Value
2. q or r is not equal to r, has a False Value

Rules Example:
Consider the following argument (sequence of propositions):

- If the prof offers chocolate for an answer, you answer the prof's question
- The prof offers chocolate for an answer.

- Therefore, you answer the prof's question

Train Barrier – Stopped / Moving / isup / isdown

stopped: boolean
movingup: boolean
movingdown: boolean
isup: boolean
isdown: boolean
isdown: boolean
velocity:integer

Invariant:
¬ (stopped /\ (movingup \/ movingdown)) /
¬ ((movingup \/ movingdown) /\ isup) /
¬ ((movingup \/ movingdown) /\ isdown) /
¬ (isup /\ isdown) /
¬ ((velocity>0) /\ (stopped \/ isup \/ isdown))

Operation to start moving up, start_up():
pre-condition: stopped /\ isdown

post-condition: movingup /\ ¬ stopped /\ ¬ isdown

post-condition: movingup /\¬ stopped /\¬ isdown

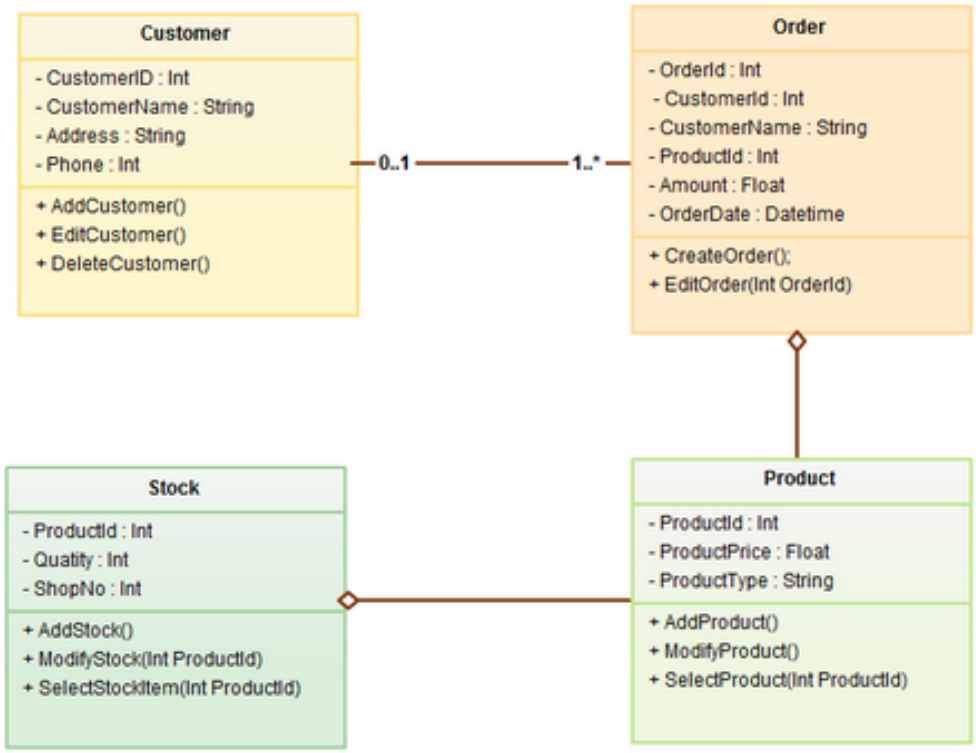
Operation to stop at top, stop_top():
pre-condition: movingup
post-condition: stopped /\ isup /\ ¬ movingup

Larman UML – Kinds of UML – THIS WAS 35 MARKS IN THE EXAM !!!

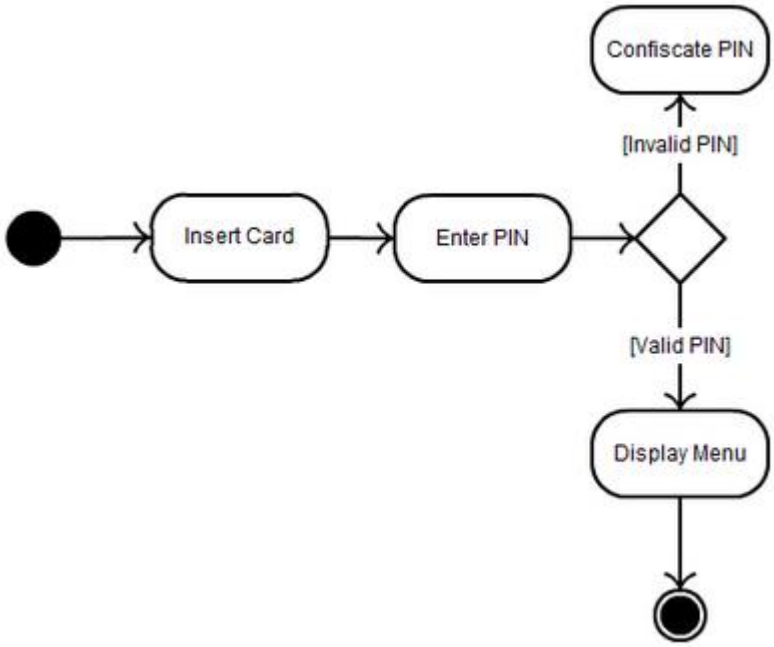
1. Class Diagram
2. Component Diagram
3. Deployment Diagram
4. Object Diagram
5. Package Diagram
6. Profile Diagram
7. Composite Structure Diagram
8. Use Case Diagram
9. Activity Diagram
10. State Machine Diagram
11. Sequence Diagram
12. Communication Diagram
13. Interaction Overview Diagram
14. Timing Diagram

	Title : CS4504 Study
	Student Name : Brian O Regan
	Student Number : 110707163
	Module : CS4504
	Exam Date: Thursday 1st May @ 09.30

CLASS DIAGRAM

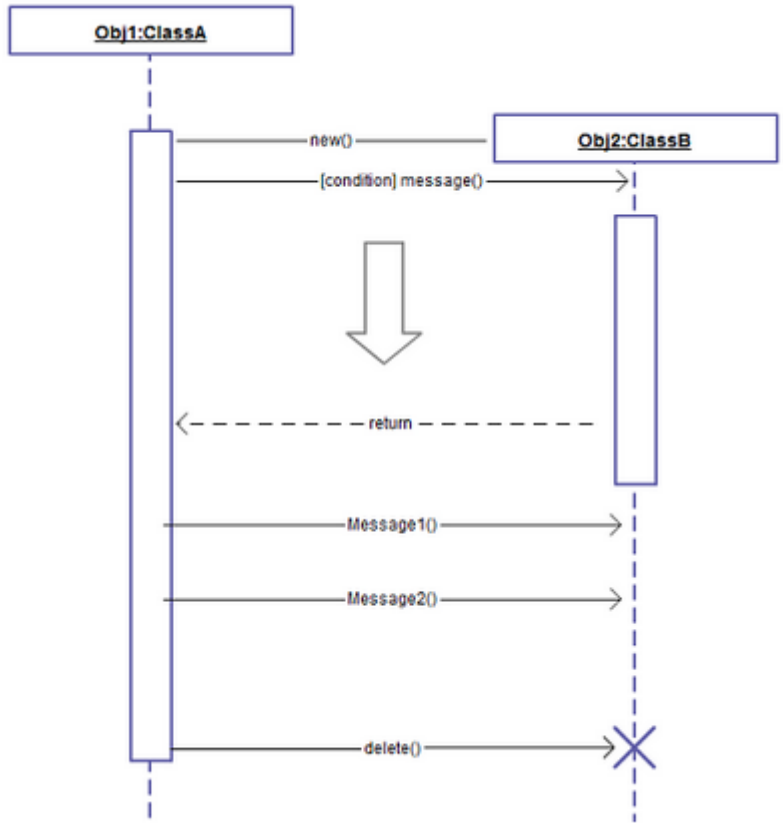


ACTIVITY DIAGRAM

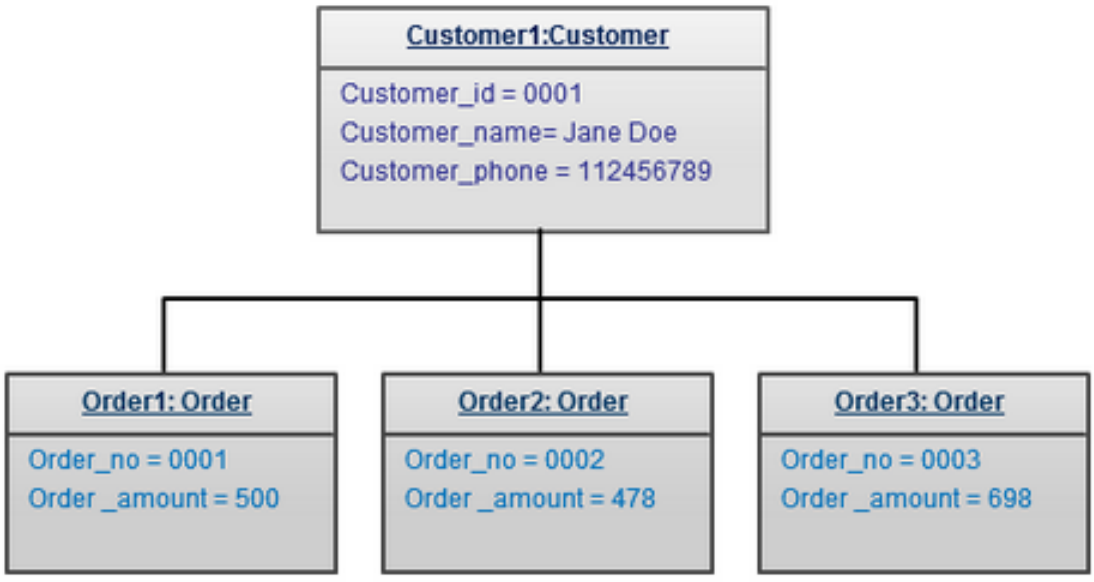


	Title : CS4504 Study
	Student Name : Brian O Regan
	Student Number : 110707163
	Module : CS4504
	Exam Date: Thursday 1st May @ 09.30

SEQUENCE DIAGRAM

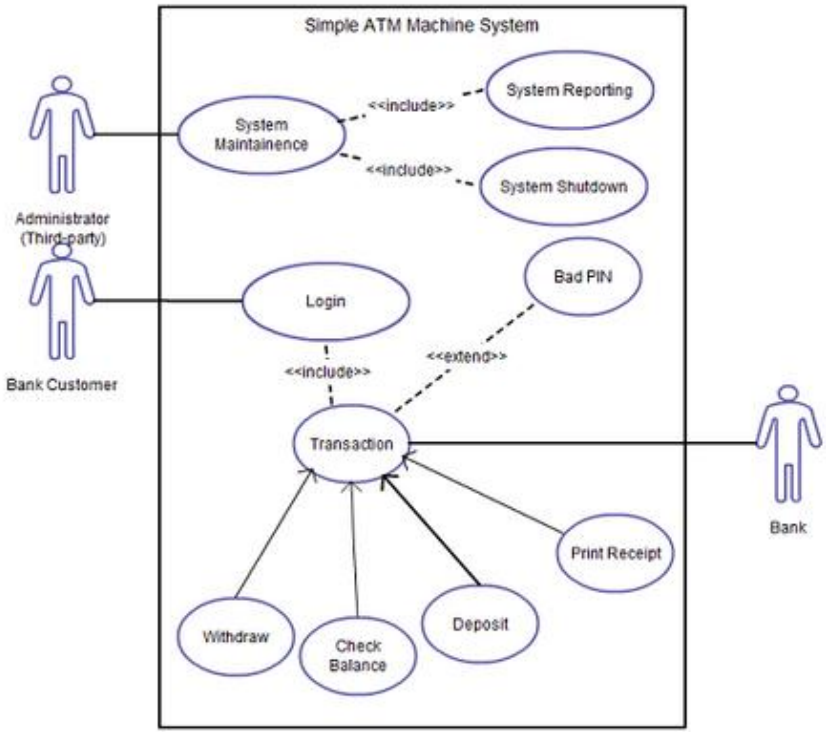


OBJECT DIAGRAM

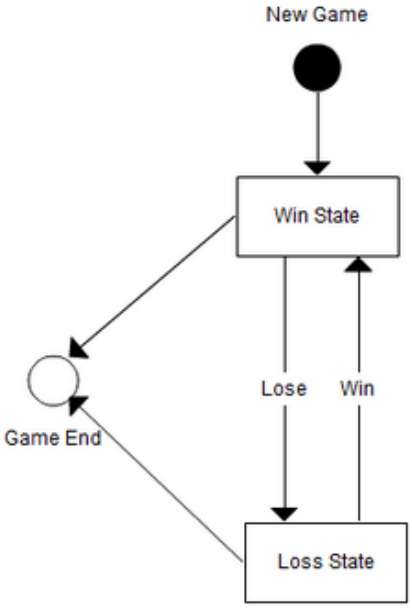


	Title : CS4504 Study
	Student Name : Brian O Regan
	Student Number : 110707163
	Module : CS4504
	Exam Date: Thursday 1st May @ 09.30

USE CASE DIAGRAM



STATE MACHINE DIAGRAM



ALL DIAGRAMS CAN BE FOUND AT : http://creately.com/blog/diagrams/uml-diagram-types-examples/

ucc Coláiste na hOllscoile Corcaigh, Éire University College Cork, Ireland	*Title : CS4504 Study*
	Student Name : Brian O Regan
	Student Number : 110707163
	Module : CS4504
	Exam Date: Thursday 1st May @ 09.30

Why is Low-Coupling and High-Cohesion desirable in an object-oriented design?

- Cohesion refers to the degree to which the elements of a module/class belong together, suggestion is all the related code should be close to each other, so we should strive for high cohesion and bind all related code together as far as possible. It has to do with the elements **within** the module/class
- Coupling refers to the degree to which the different modules/classes depend on each other, suggestion is all modules should be independent as far as possible, that's why low coupling. It has to do with the elements **among** different modules/classes
- Low coupling means components can be swapped out without affecting the proper functioning of a system. Basically modulate your system into functioning components that can be updated individually without breaking the system.
- Cohesion refers to what the class (or module) will do. Low cohesion would mean that the class does a great variety of actions and is not focused on what it should do. High cohesion would then mean that the class is focused on what it should be doing, i.e. only methods relating to the intention of the class.
- As for coupling, it refers to how related are two classes / modules and how dependent they are on each other. Being low coupling would mean that changing something major in one class should not affect the other. High coupling would make your code difficult to make changes as well as to maintain it, as classes are coupled closely together, making a change could mean an entire system revamp.
- All good software design will go for high cohesion and low coupling.

Loose coupling makes it possible to:

- Understand one class without reading others
- Change one class without affecting others
- Thus: improves maintainability

High cohesion makes it easier to:

- Understand what a class or method does
- Use descriptive names
- Reuse classes or methods

Example of Low Cohesion:

```
--------------------
| Staff            |
--------------------
| checkEmail()     |
| sendEmail()      |
| emailValidate()  |
| PrintLetter()    |
--------------------
```

Example of High Cohesion:

```
----------------------------
| Staff                    |
----------------------------
| -salary                  |
| -emailAddr               |
----------------------------
| setSalary(newSalary)     |
| getSalary()              |
| setEmailAddr(newEmail)   |
| getEmailAddr()           |
----------------------------
```

High-Coupling Example

```
public class CartEntry
{
   public float Price;
   public int Quantity;
}

public class CartContents
{
   public CartEntry[] items;
}

public class Order
{
   private CartContents cart;
   private float salesTax;

   public Order(CartContents cart, float salesTax)
   {
      this.cart = cart;
      this.salesTax = salesTax;
   }

   public float OrderTotal()
   {
      float cartTotal = 0;
      for (int i = 0; i < cart.items.Length; i++)
      {
         cartTotal += cart.items[i].Price *
cart.items[i].Quantity;
      }
      cartTotal += cartTotal*salesTax;
      return cartTotal;
   }
}
```

Low-Coupling Example

```
public class CartEntry
{
   public float Price;
   public int Quantity;

   public float GetLineItemTotal()
   {
      return Price * Quantity;
   }
}

public class CartContents
{
   public CartEntry[] items;

   public float GetCartItemsTotal()
   {
      float cartTotal = 0;
      foreach (CartEntry item in items)
      {
         cartTotal += item.GetLineItemTotal();
      }
      return cartTotal;
   }
}

public class Order
{
   private CartContents cart;
   private float salesTax;

   public Order(CartContents cart, float salesTax)
   {
      this.cart = cart;
      this.salesTax = salesTax;
   }

   public float OrderTotal()
   {
      return cart.GetCartItemsTotal() * (1.0f + salesTax);
   }
}
```

Inspection and Testing
(Validation and Verification)

- These are complementary processes
- Inspections can check conformance to specifications, but not with customer's real needs
- Testing must be used to check compliance with non-functional system characteristics like performance, usability, etc.

- Verification
  - Are you building the product right?
  - Software must conform to its specification

- Validation
  - Are you building the right product?
  - Software should do what the user really requires
- Software inspections (static)
  - Concerned with analysis of static system representations to discover errors
  - May be supplemented by tool-based analysis of documents and program code
- Software testing (dynamic)
  - Concerned with exercising product using test data and observing behaviour

Program Testing

- Can only reveal the presence of errors, cannot prove their absence
- A successful test discovers 1 or more errors
- The only validation technique that should be used for non-functional (or performance) requirements
- Should be used in conjunction with static verification to ensure full product coverage

Software Inspections

- People examine a source code representation to discover anomalies and defects
- Does not require systems execution so they may occur before implementation
- May be applied to any system representation (document, model, test data, code, etc.)

Inspections Appropriate for:

- Banking / Financial Systems
- Air Traffic Control Systems

Testing Appropriate for:

- Computer Games
- Microsoft Office

Statistical Testing

Statistical testing of software is here defined as testing in which the test cases are produced by a random process meant to produce different test cases with the same probabilities with which they would arise in actual use of the software. Statistical testing of software has these main advantages:

- For the purpose of reliability assessment and product acceptance, it supports directly estimates of reliability, and thus decisions on whether the software is ready for delivery or for use in a specific system. This feature is unique to statistical testing;
- For the purpose of improving the software, it tends to discover defects which would cause failures with the higher frequencies before those that would cause less frequent failures, thus focusing correction efforts in the most cost-effective way and delivering better software for a given debugging effort. Statistical testing has been reported to achieve dramatic improvements;
- From the point of view of costs, it facilitates the automation of the test process, thus allowing more testing at acceptable cost than manual testing would allow.

	Title : CS4504 Study
	Student Name : Brian O Regan
	Student Number : 110707163
	Module : CS4504
	Exam Date: Thursday 1st May @ 09.30

Operational Profile

A profile is a set of independent possibilities called elements, and their associated probability of occurrence. If operation A occurs 60 percent of the time, B occurs 30 percent, and C occurs 10 percent, for example, the profile is [A, 0.6...B, 0.3...C, 0.1]. The operational profile is the set of independent operations that a software system performs and their associated probabilities.

Developing an operational profile for a system involves one or more of the following five steps:

1. Find the customer profile
2. Establish the user profile
3. Define the system-mode profile
4. Determine the functional profile
5. Determine the operational profile itself

The process for developing the operational profile is depicted in Figure 9-1.

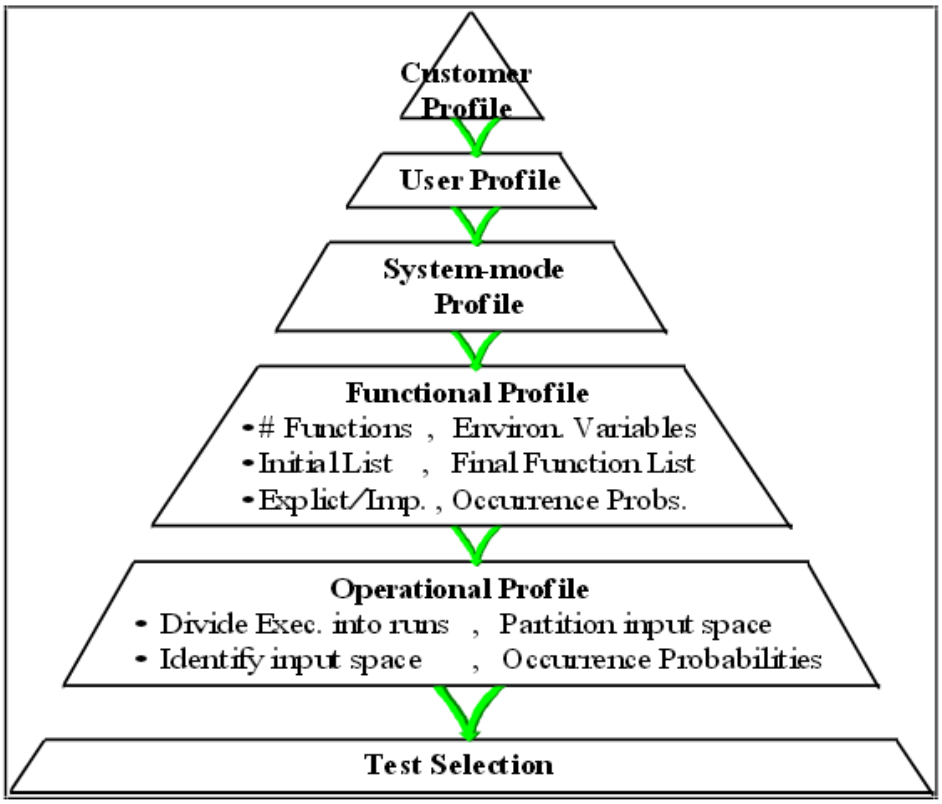


FIGURE 9-1. Operational Profile Development[2]