

ESCOM 2001

Using measures to understand requirements

Software Measurement Services Ltd

+44 (0)1732-863-760

PG_Rule@compuserve.com

www.software-measurement.com

Software Measurement Services



In a dynamic environment
business organisations need predictability

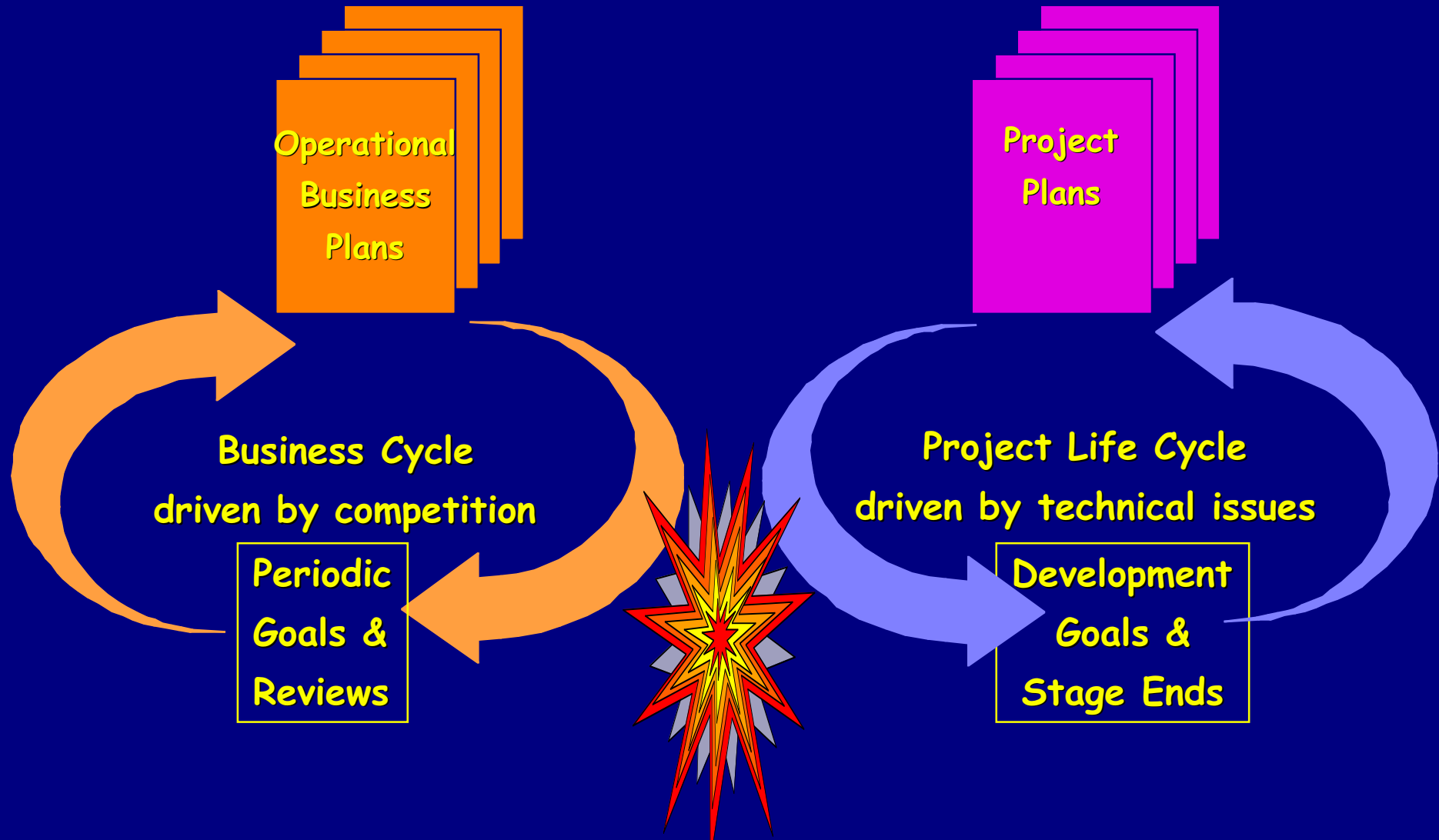
What do we want ?

Predictability !

When do we want it ?

NOW !!!

There is contention between business & project plans as they operate to different schedules



Hardware, software & technical methods have evolved but still many projects fail

- Machine code
- Assemblers
- 3GLs & 4GLs
- Structured Methods
- Object Oriented Methods & UML
- Project Management Methods
 - Waterfall, V, Spiral, RAD, eXtreme Programming

It's the REQUIREMENTS stupid !

The 'use case' considered harmful

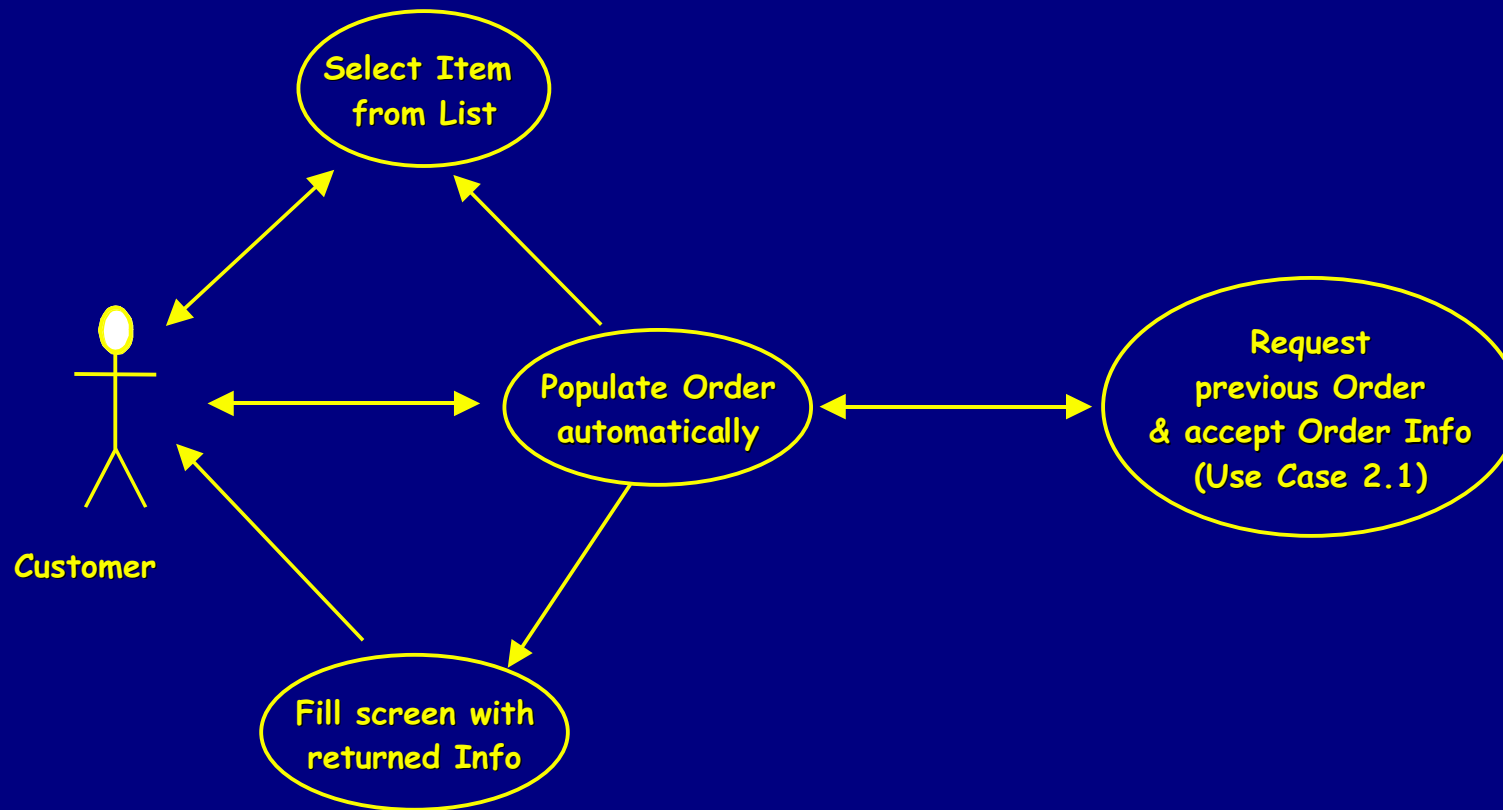
Few organisations use 'use cases' as they were intended

- 57 Varieties
 - (well, OK 'only' 32)
- Individuals apply different rules... so what does it mean ?
- Lack of prioritisation
- Focus on the solution before analysing the problem
- Too early attempts at optimisation

Case 1: a bespoke client-server development cancelled at a cost of £5m GBP

- Phase 1 of multi-phase project
- 12 months into a 10 month phase... and still no end in sight ?
- 5 times larger than 'expected'
- Cancelled after 18 months
 - Supplier made a loss
 - Customer received nothing useful
- Project staff added DHTML to their CVs

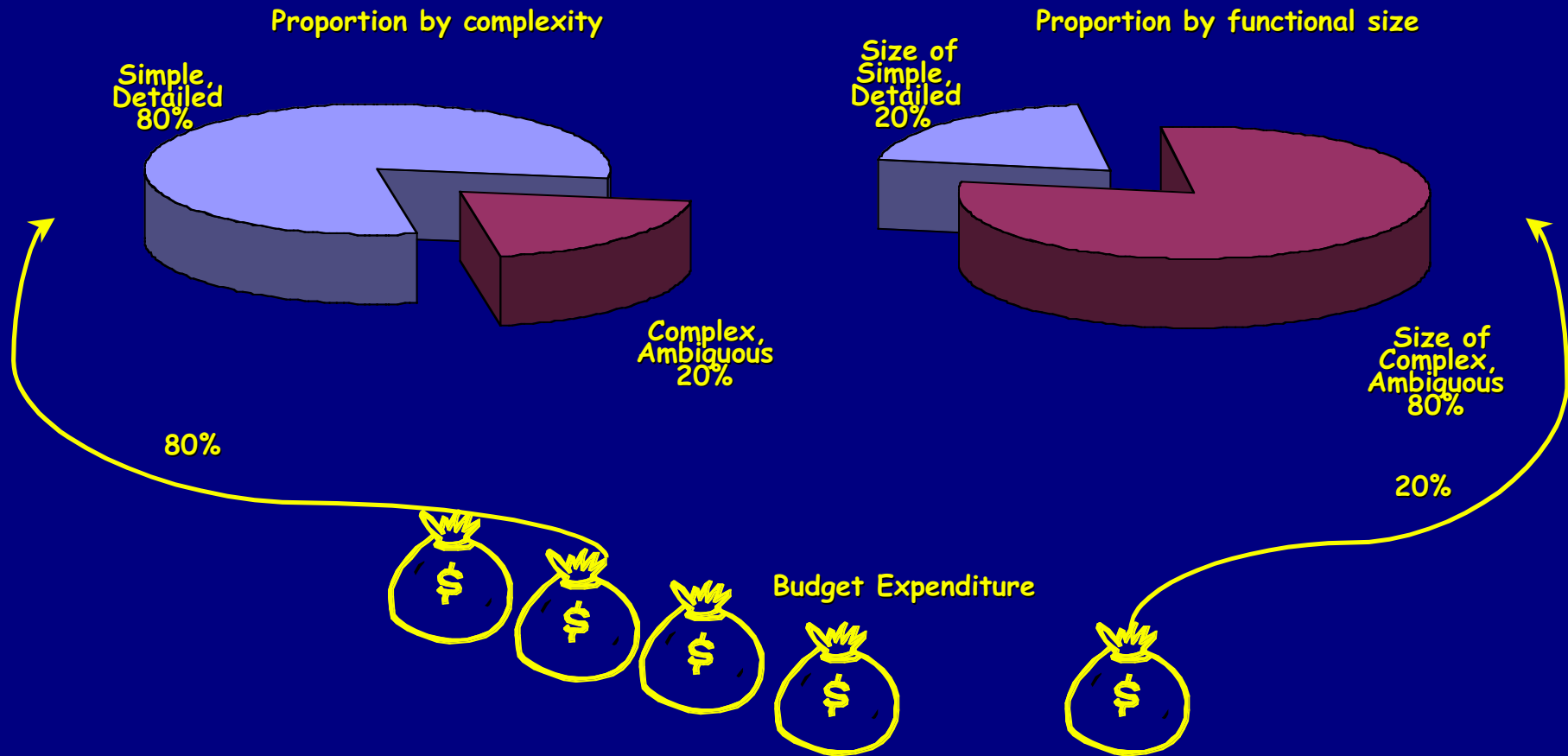
A 'typical' use case diagram ?



Don't thou do likewise !

- Examples in the written paper
- Confusion of 'primary' and 'alternate courses'
- Description of the software rather than the problem
- Uninformative diagrams... 'going through the motions'
- 'Reuse' in action: the 'do stuff' use case

80% of the use cases may be simple, but the other 20% may account for most of the size and complexity



In practice, 'use cases' seem to suffer some systematic problems

- Lack of rigour in the application
- Lack of a consistent level of granularity
- Belief that the *number* of use cases is equivalent to the *size* of the requirements
- A desperate, unrequited search for reuse: 'used by/uses' & 'extends' use cases

These faults can be counter-balanced by incorporating FSM techniques

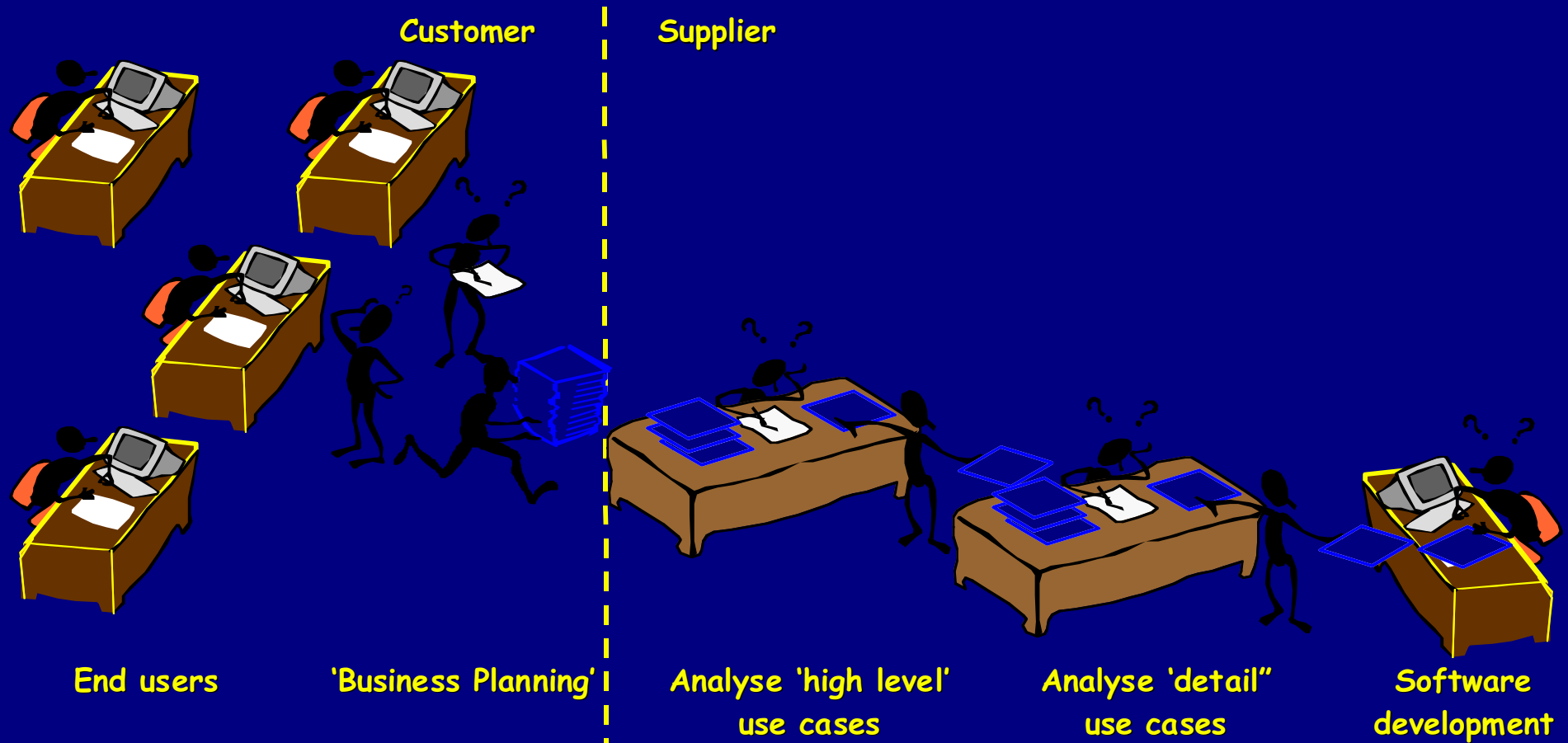
- Analysis of the conversational dialogue
- Identification of the external Business Events at a consistent, low level of granularity
- Simple cataloguing of Stimulus/Response message pairs
- Recognising that 'alternate courses' are part of the Response
- Distinguishing between 'Actors' and 'Agents'

Understand the tradeoffs between project constraints

Case 2: A multi-release development where the customer-supplier relationship broke down

- Global, business-critical application
- 200 user sites
- 3 tier client-server architecture
- Phase 1 (of five) delivered late, over-budget, poor quality
- Supplier performance no better than in-house IT group
- Inflexible contractual conditions

Analyse the problem rather than someone's interpretation of the problem



A wasteful chain of requirements engineering activity

Investigation showed the supplier doing better than appeared

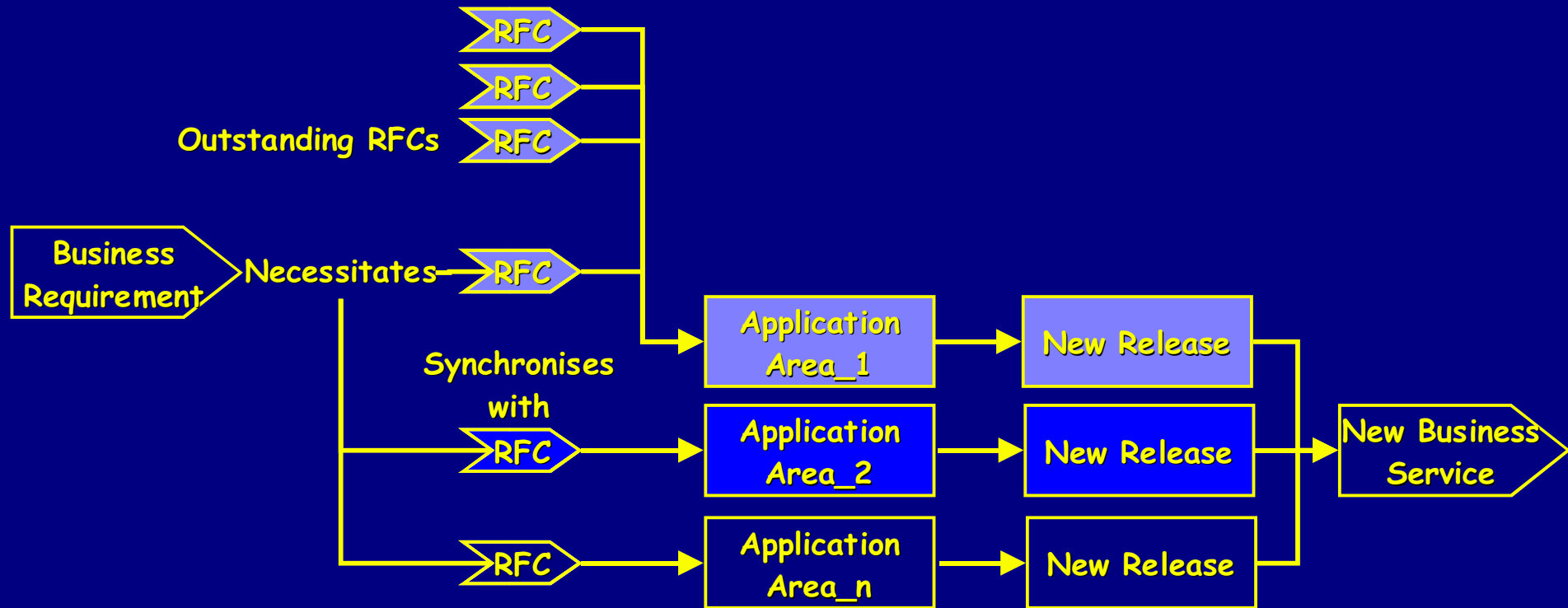
- Phase 1 delivered *3 times faster* than comparable in-house projects
- 7 months per phase rather than 18-24 months
- 'Leading edge' technology - always a bummer
- Team recruited 'in-flight' - not an experienced team used to working together

Competitive markets need commitment & discipline

Case 3: a mobile telephony organisation driven by marketing & competition

- Enhancement & support of existing application portfolio
- Highly volatile market... time critical
- Large, high volume applications
- Quality of service impacts end-user (the public)
- Highly-constrained software... call rating & billing systems, SMS, WAP, data-warehouse, voicemail
- Monthly releases of enhancements

Change must be co-ordinated across multiple application areas



Improvement opportunities included...

- Make measurements available to project teams
- More rigorous & comprehensive configuration management
- Measure the requirements specified as well as the functionality delivered
- Track 'scope creep'
- Construct test cases earlier, based on the Transaction Catalogue
- Use COSMIC FSM to size multi-layered software
- Improve management commitment to completeness & accuracy eg. of time records
- Record & investigate defects found during construction, testing & operation

Measured requirements as the basis for managing supplier agreements

Case 4: An outsourced contract where the flow of requirements to the supplier is controlled

- All software development, enhancement & support outsourced for a 7 year commitment
- Annual accounting of functionality delivered
- Functional size measures used as basis for contractual payments
- Several hundred software staff
- 30 trained MkII FPA analysts
- Requirements sized by seconded 'measurement analyst'
- Supplier QA of project measurements

Providing assurance to both customer & supplier

- Annual audit of measurement practices and results
 - Consumes approx. 25 work days effort
- 3rd party, independent auditor
- 3 year's worth of results...
 - Mean size error per project between +/- 2% and +/- 2.5%
 - Mean size error over annual workload +/- 1%
 - Measured software traceable to requirements
 - Consistency of practices & procedures across projects & over time
- Confidence for both customer & supplier maintains good relationship

In each of these four distinct cases size measures have helped with requirements

A client-server project using 'use-cases'

- Earlier use of size measures could have improved estimates and prevented cancellation

A global multi-release development...

- ...where the customer-supplier relationship was improved by measuring & explaining performance

An organisation driven by marketing...

- ...where measures have helped control & understand requirements and costs

An outsourced scenario...

- ...where the flow of requirements to the supplier is monitored for contract pricing

Conclusion

- Measurement can be used to improve...
 - Correctness, Completeness, Consistency
 - Testability, Traceability
- Measured requirements can...
 - Improve customer-supplier relationships
 - Serve as a basis for contract pricing, contract negotiation & control of scope creep
- Understanding **product size** is crucial for
 - planning & implementing the software process
 - managing project constraints (time, cost, resources, customer satisfaction)



fin

Using measures to understand requirements

P. Grant Rule

Abstract

Many approaches fashionable with technically-oriented practitioners clearly fail to satisfy the need for clarity of requirements. Some even trade short-term acceleration for long-term maintenance & support costs. What is missing? What can be done to ensure that new technologies help rather than hinder? This paper suggests some simple process improvements that might have made all the difference in a number of cases including:

- *A bespoke client/server development using Use Cases to drive OOA/D, cancelled at a cost of £5m GBP due to poorly understood requirements.*
- *A global, multi-release development where the customer/supplier relationship broke down due to misunderstood tradeoffs between speed and productivity.*
- *The case of an organisation driven by marketing & competition that has struggled to understand and control requirements and costs.*
- *A case where the 'flow of requirements' to an outsourced supplier is controlled using measures agreed at a contractual level.*

1. Introduction

Business organisations need, above all else, huge predictability.

In order to manage their budgets and the commitments to their stakeholders, business organisations rely on software projects completing on-schedule, for predictable costs. If they fail in this, the organisation may be absorbed or destroyed altogether.

However, the business and technical environment in which commercial organisations operate is forever changing. Organisations are subject to pressures from customers, competitors, legislators, predators, shareholders and staff... and the pace of change seems to be increasing. Hence, there is contention between the need for predictability and the dynamic nature of the environment.

Some fashionable approaches to software development have been derived to resolve this conflict.

Arguably, structured and object-oriented methods, including the Unified Modeling Language (UML), have been produced to reduce the variability of the technical processes of software development and maintenance. Similarly, project management methods, from the 'waterfall' and 'spiral' models, through Rapid and Joint Application Development and now eXtreme Programming, have 'evolved' in reaction to changing requirements.

Through all this, the ability to understand and to be able to communicate about, the requirements allocated to software remains a necessity. Without a clear understanding of the requirements, and the changes to the requirements, it must remain difficult to deliver a predictable software process.

2. Improving the use of use cases

- *Case: A bespoke client/server development using Use Cases to drive OOA/D, cancelled at a cost of £5m GBP due to poorly understood requirements.*

2.1. It is important to understand the granularity of the requirements

The Use Case technique is one of the most popular modern methods of documenting requirements. However, in practice there seems to be much disparity in how different people understand use cases. Various practitioners, including IBM, report finding up to 32 different

interpretations ^[1]. As different individuals apply different rules, even within one project, the results can be ambiguous.

Over the past few years, SMS consultants have observed a number of projects using the use case technique, in a variety of application domains, and have identified a number of common issues.

The main concern is that there is a tendency for developers to define the ‘easy’ use cases first and in great detail, while documenting more ‘difficult’ use cases very briefly, putting them aside ‘until later’. At the extreme, we have found use cases that say the equivalent of ‘do stuff’ or ‘browse content’, wholly failing to describe the required interaction between the actor and the application.

The result in a number of projects is that around 80% of the project budget and schedule is expended on ‘easy’ use cases (which may represent only 20% of the functionality). Then, as the project deadline approaches, the project team realises that the 20% of uses cases that remain, the ones that are ‘difficult’ or ‘complex’, represent maybe 80% of the functional size (see Figure 1). By the time this realisation dawns, there is insufficient budget and time remaining to complete the required work. Hence the project is late, over budget, the customer is dissatisfied and someone has to find the unbudgeted funds or make-do with inadequate functionality.

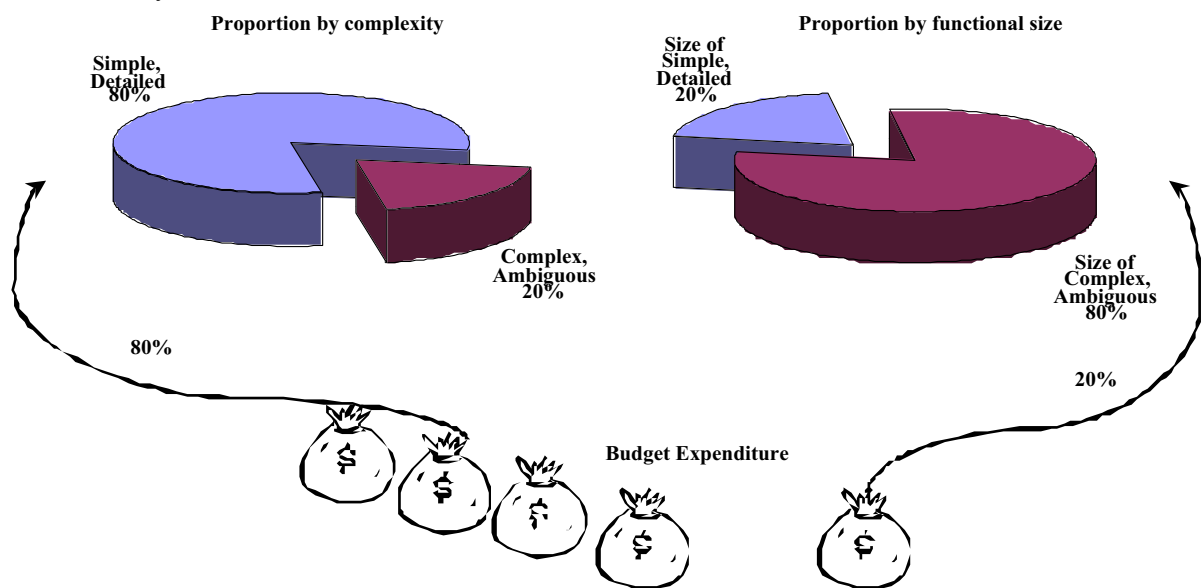


Figure 1: 80% of use cases may be ‘simple’, but a small number of ‘complex’ use cases may account for 80% of the functional size

2.2. It is important to analyse the problem, rather than someone’s interpretation

Another factor affecting the productivity of some projects is a result of the way the elicitation and description of requirements is organised. Often the customer has a specialist group, let’s call it ‘Business Planning’, that is deemed to be ‘responsible for defining the customer’s business requirements’. This group interacts with the end users and managers to determine the customer organisation’s needs. They also have the responsibility to convey the customer requirements to the supplier. On the supply side, there may be a ‘systems analysis’ group, responsible for ‘determining the customer requirements’. However, in practice they communicate to and through the customer’s Business Planning group. They are not permitted to talk directly to the end users (that would ‘interfere with the normal business processes and

production’). All communication between the supplier’s systems analysts and the end users is filtered through Business Planning. This is a recipe for much misunderstanding.

We have seen several projects that involve a chain of activity (see Figure 2) where...

...Business Planning documents ‘business requirements’ in textual form – they then pass the resulting documents to the supplier’s Systems Analysts. The Systems Analysts take the textual requirements and document a set of ‘high level use cases’, often cutting & pasting the text directly from one document into another. The added value, if any, is limited to a use case diagram of doubtful worth. Subsequently, there is a further activity by a ‘system designer’ to analyse the ‘high level use case’ and to produce one or more ‘detailed use cases’. Again, often the text is directly copied and little value is added, although the format may change.

This process wastes much time, effort and budget, adds little value and fails to describe requirements such that they can be understood unambiguously and implemented efficiently. The last two steps are self-referential and cannot possibly improve on the initial statements, because they only refer to an *interpretation* of the problem, rather than to the problem itself.

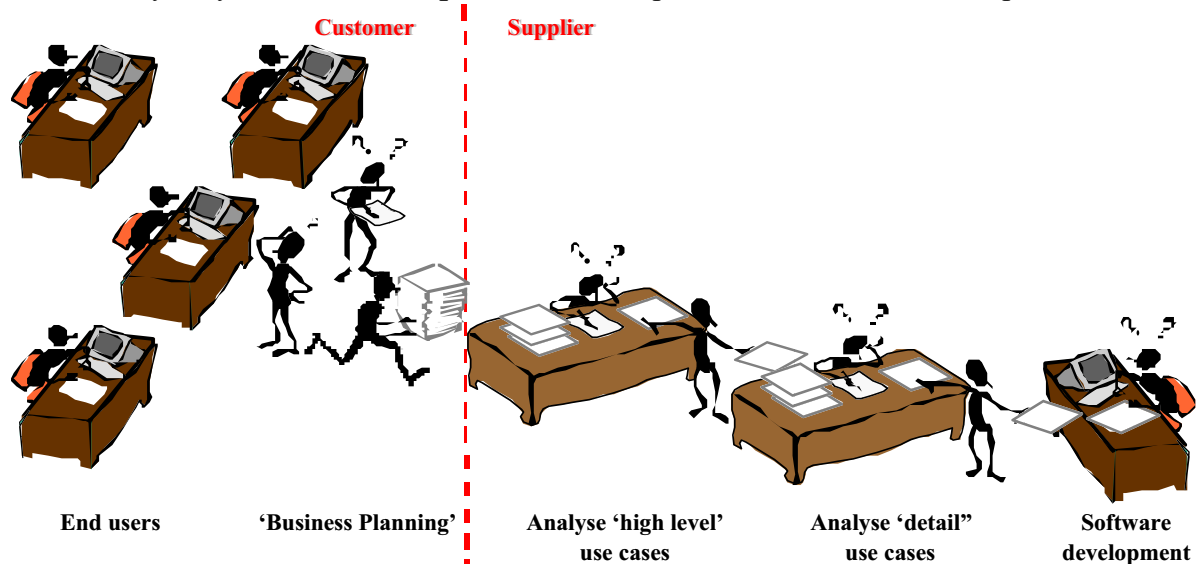


Figure 2: A wasteful chain of requirements ‘engineering’ activity

In order to derive additional detail and remove ambiguity, the supplier’s systems analysts *must* be given access to the end users and managers and allowed to analyse the requirements themselves. The customer’s ‘business planning’ group may help, establish the scope, determine the feasibility and benefits from the customer’s viewpoint, but they must not be allowed to filter (ie. ‘act as a barrier to’) communication between the supplier and the owners of the requirements.

2.3. Measuring requirements enforces rigour and highlights ambiguity early

Software measurement techniques help engineers to produce software requirements that are unambiguous and can be sized. They improve the rigour with which requirements are documented. Thus they can be used as the basis for agreement between customer and supplier.

During 2000, SMS performed a Functional Sizing & Estimating Study for one supplier that had been working on Phase-1 of a multi-phase project for some 12 months. Phase-1 had already overrun by several months, making the management wonder about the commitment represented by subsequent phases. The study showed that the project was some *five times larger* than the supplier had originally understood. After protracted discussions with the

customer, the supplier withdrew from the project at a cost to them of £5m GBP (\$7.5m USD), leaving the customer with nothing to show for nearly 18 months of work.

An example of the use case template used by this project is presented in Table 1.

Table 1: Example of a use case template (adapted from a real project)

Field	Field Contents
Use Case Name:	Unique identifier for the use case
Primary Actor:	Name of the main actor
Supporting Actors:	Names of associated actors assisting the Primary Actor
Description:	Text description of the interaction between Actor and Application
Assumptions:	Conditions that are assumed to be TRUE
Pre-Conditions:	Entry Conditions that must be TRUE before the use case can commence
Primary Course:	A description of the most common set of steps in the use case. In practice, this usually means a list of the stimulus/response pairs that make up the use case, where the Actor provides the stimulus and the application provides the response.
Post Conditions:	Exit Conditions that must be TRUE at the end of the use case
Alternate Courses:	A description of other steps and results that may occur instead of the Primary (most likely) Course. This includes all possible error conditions.
Activity Diagram:	An illustration intended to clarify the text of the Primary and Alternate Courses.
Used By:	Identification of any other Use Cases that (re)use the functionality of this Use Case.
Uses:	Identification of any other Use Cases that this Use Case (re)uses.
Issues:	Description of any unresolved queries regarding the requirements described by this Use Case
Version Number:	The version number of this version of the Use Case document
Last Date Changed:	The date this document was last modified
Last Changed By:	The name of the author who made the last modification

Illustrations of the way this template was used is given in Figure 3 & Table 2.

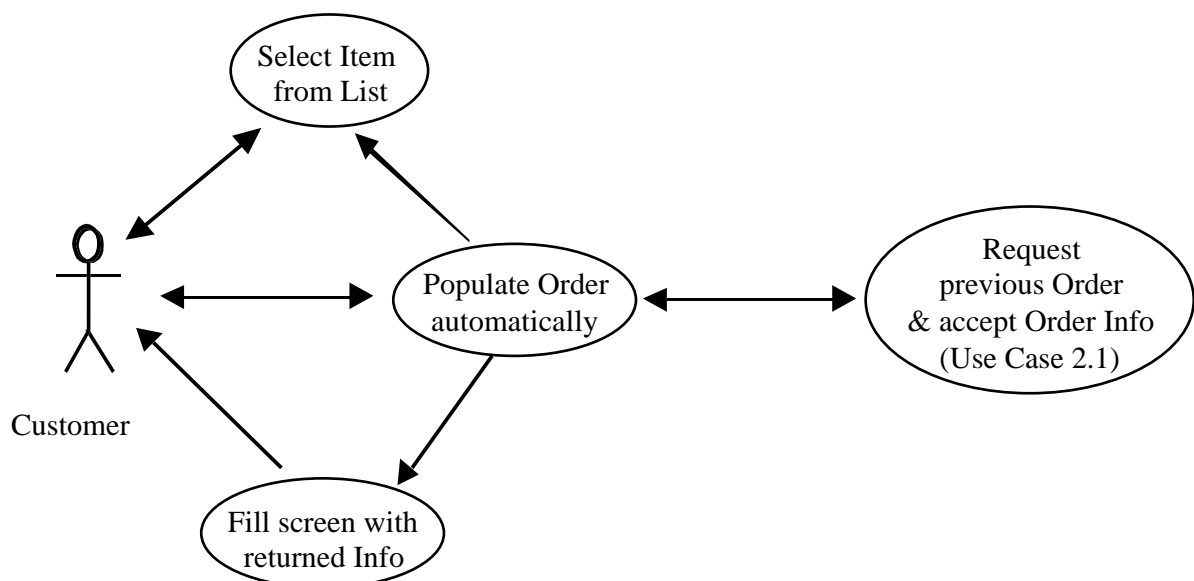


Figure 3: Example of a 'typical' use case diagram (adapted from a real project)

Early measurement of the requirements would have exposed the real size of the product and enabled better cost estimation and decision making by both customer and supplier. Simply applying measurement to the requirements identified various issues of ambiguity and highlighted the amount of uncontrolled scope-creep that was being experienced.

Table 2: Example of a 'detailed use case' (adapted from a real project)

Use Case Name	UC1.1.1 Display Previous Order List
Primary Actor	Order System
Supporting Actors	none
Description	The Order System shall be able to retrieve previous Orders as a list from which a selection may be made. The list shall be in descending sequence of Order Number for the current Customer only. The System shall mark Orders in the list as "Completed" or "Partial". The displayed list shall be ten items long. Only Orders created within the last thirty days will be displayed, it shall be possible to request the next ten Orders in descending Order Number order. It shall also be possible to select a particular Order by Order Number, previous Orders will be displayed with sufficient information to enable the Customer to make a choice.
Assumptions	none
Pre-Conditions	Customer has requested a list of previous Orders
Primary Course	Order System requests an Order collection using enter Customer criteria from Order Information Order Information System returns a collection of Orders Order System displays list of Orders with information to enable a choice to be made
Post Conditions	List of Orders satisfying selection is displayed on Web Page
Alternate Courses	none
Activity Diagram	<pre> graph LR Start((Start)) --> S1(Select Order from List) S1 --> S2(Obtain Order information) S2 --> S3(Fill screen with Order information) S3 --> End(((End))) </pre>
Used By:	none
Uses:	The Use Case uses Request Previous Order from Order database (UC1.3.1) and Send Previous Order from Order database (UC1.3.2)
Issues	none
Version Number	1.0000.
Last Date Changed	2/2/2000
Last Changed By	A. N. Other

In practice, as a means of describing the requirements allocated to software, use case descriptions seem to suffer from the following problems:

- lack of rigour in the application of the technique
- lack of a consistent level of granularity – hence, although a developer may know the **number** of use cases, this tells them little about the **size** of the application to be developed and hence the effort required

- UML practitioners freely apply the concept of abstract use cases (‘uses/used by’ and ‘extends’) that are more concerned with identifying opportunities for reuse than with analysing the problem and describing the requirements
- too early ‘optimisation’ of the solution and a tendency to jump into ‘the first solution that is thought of’ rather than do a considered evaluation of a number of solution options

These faults can be counter-balanced by incorporating the MkII Function Point Analysis (FPA) concept of the ‘logical transaction’ into the use case description.

In fact, the functionality described by the use case in Table 2 consisted of that shown in Table 3.

Table 3: Actual functionality expressed as Stimulus/Response Pairs

Stimulus/Response 1	An Actor selects the Display Customer Details function from the application menu and the application responds by displaying the Select Customer screen.
Stimulus/Response 2	The Actor enters a value for the Customer_Id and the application responds by retrieving the Customer Record with a matching Customer_Id and displaying the Customer Details screen – if no match is found, the Select Customer screen is displayed with an error message ‘No match found for that Customer_Id’ – the Actor may then repeat the operation.
Stimulus/Response 3	The Actor selects the List Orders function from a drop-down menu and the application responds by listing all Orders in descending sequence whose Ordered_By field contains a value equal to the previously entered Customer_Id and whose Order_Date is not more than the Current Date less 30 days.
	If there are more than 10 Orders, the most recent 10 Orders are displayed (ie. the Orders with the 10 latest Order_Numbers). A scroll bar is provided to enable the Actor to scroll up and down the list of orders for this customer. If there are zero Orders, the screen is displayed with a message ‘No orders made within last 30 days’.
	Each Order is listed on one row, including an Order_Status field, indicating whether this order is Partial or Completed.
Stimulus/Response 4	The Actor highlights one of the Orders from the list and selects the Display Order Details function from a drop-down menu and the application responds by retrieving the details of the highlighted Order and displaying them.

This maps very well to the MkII FPA logical transaction ^[2], which is defined as...

...the lowest level business process supported by a software application. It comprises three elements: input across an application boundary, some related processing, and output across the application boundary. Each logical transaction is triggered by a unique event of interest in the external world, or a request for information and, when wholly complete, leaves the application in a self-consistent state in relation to the unique event.

Hence, the Primary Course and Alternate Course parts of the use case template can be replaced (or at least supplemented) by a table (see Table 4) that decomposes the Actor/Application interaction into logical transactions. This can be done very early in the product lifecycle, can be refined later as necessary, enforces a consistent level of granularity, highlights ambiguity where it exists and is inherently measurable.

Table 4: Expressing the example use case as measurable logical transactions

Id	Stimulus	Input fields	Object Class Referenced	Response	Response fields
1	Select function from menu	None 0	None 0	Display empty screen <Select Customer>	None 0
2	Query Customer Details	Customer_Id 1	Customer 1	<u>Primary Route</u> Display screen <Customer Details> <u>Alternate Route</u> Display screen <Select Customer>	<u>Primary Route</u> First_Name Last_Name Organisation Address Phone Fax <u>Alternate Route</u> Error Message 7
3	Select List Orders	Customer_Id 1	Customer Order 2	Display screen <Orders List>	<u>Primary Route</u> Order_Id Order_Date Product Quantity Order_Status <u>Alternate Route</u> Info Message 6
4	Select Specified Order	Order_Id 1	Order Product 2	Display screen <Order Details>	Order_Id Order_Date Product Quantity Order_Status Product_Description Product_Price Order_Value 8
Sub-Totals:		3	5		21
MkII Weights:		0.58	1.66		0.26
Contributions:		1.74	8.3		5.46
Functional Size		= 1.74 + 8.3 + 5.46 = 15.5 fp (MkII r1.3.1)			

3. Trading speed and productivity

- *Case: A global, multi-release development where the customer/supplier relationship broke down due to misunderstood tradeoffs between speed and productivity.*

Late in 1999, a supplier organisation performing a bespoke project in Denmark called upon SMS to assist with sizing and estimating requirements. The supplier had just made delivery of Phase-1 of the project and completed User Acceptance Testing of the product with their customer. However, this success was only achieved at the expense of many late nights and the provision of additional resources to the project team. Furthermore, the supplier was being castigated by the customer because of their apparent ‘inflexibility’ and ‘poor productivity’, which was ‘lower than that previously achieved by the customer’s in-house software staff’.

The project used use cases to document requirements. These requirements were initially documented by a ‘business planning’ group which handled all interactions with the suppliers’

systems analysts. The end-users were located at over 200 different sites. The architecture was three-tier client/server.

The results of SMS' initial study of Phase-1 indicated that the supplier was being asked to deliver in some 7 months, functionality that had previously taken between 18 and 24 months to deliver. That is, about 3 times faster than previous projects of similar size performed by the in-house staff.

Both the supplier and customer organisations recognised from their experience of this first phase that better understanding and control of the requirements were needed. Subsequent phases applied a systematic estimation process and the plans and commitments were based on those requirements.

4. Requirements in a telecoms organisation driven by marketing

- *Case: The case of an organisation driven by marketing & competition that has struggled to understand and control requirements and costs.*

Many organisations already have a wealth of software. Their concern is not so much the development of new applications, but the maintenance and support of their existing software portfolio. This is especially a problem when the market in which the organisation operates is highly volatile and competitive, the size of the software applications is large, the volume of data that must be processed is high and the quality of the service provided to customers must be 'best in class'. All the above criteria apply to the world of mobile telephony.

Telecommunications software bridges the exceedingly vague borders between highly constrained signal-routing software operating close to real-time and the less constrained information processing applications used for customer billing, management information, etc. However, as the tariff to be applied to a mobile telephone call, and indeed whether there is sufficient credit available to make the call, has to be calculated before the call is connected, even the 'less constrained' applications have to perform well.

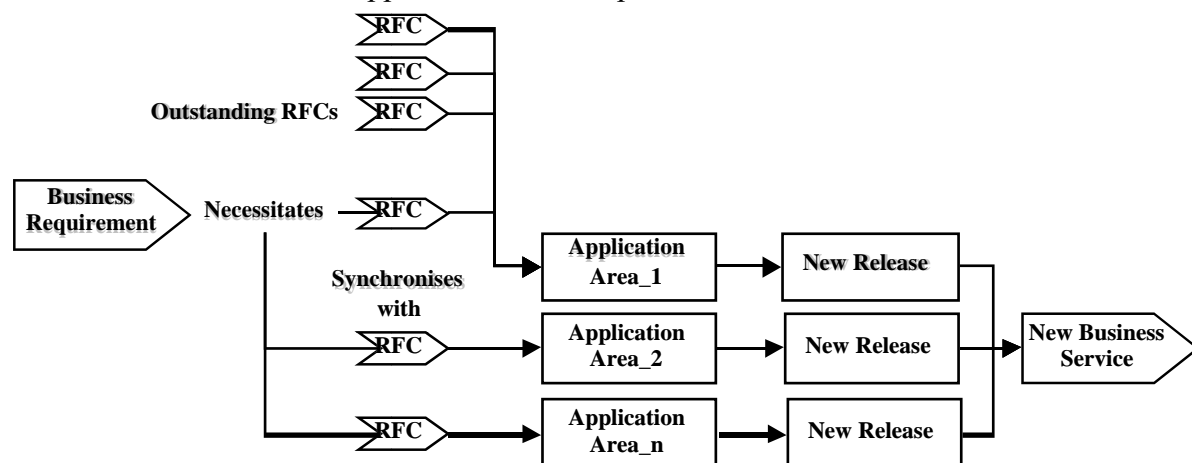


Figure 4: Co-ordinating business change across multiple application areas

Telephony also is a highly competitive business. Organisations must rapidly take advantage of each new technological advance in order to present new services to their customers, or risk losing market share to competing companies. Consumer-oriented business is cyclical in nature, driven by seasonal sales peaks, such as the Christmas period, that dominate annual sales of, for example, 'pay as you go' mobile phones. Due to the lead-time necessary to prepare marketing and sales campaigns, it is often necessary for organisations to prepare and run advertising campaigns for new services before the software capability to

support those services is available. In such an environment, the software enhancement strategy is based around many relatively small, monthly releases.

The management task is complicated even further when business critical applications are outsourced to specialist suppliers over whose staff the organisation has no direct control.

In the case in question, the Programme Office has identified the application areas that absorb around 80% of the maintenance & support budget. Each of these areas is supported by a dedicated team that handles many Requests-For-Change (RFC) during each year. Each RFC is prioritised from the business perspective and the impact of that change is assessed from the perspective of the software engineers, both in the application area immediately concerned and across other application areas that may be impacted (see Figure 4). Often there is a need to co-ordinate and synchronise multiple RFC's in more than one application area in order to deliver a new business requirement.

The RFCs scheduled for completion each period are sized using MkII FPA and this information, along with records of effort expended, duration and cost are reported to senior management quarterly. This information is used to help inform decisions about the estimates and feasibility of future work and also to evaluate the productivity and performance and in-house groups and external suppliers.

An external agency, SMS, is used to collect functional size figures for each period, while effort figures are collected via an purpose built Time Recording System, supplemented by project management records made by individual project managers. Data relating to operational failures originated from software defects are collected via the help desk systems and an independent testing group. This measurement regime currently is exercised to varying degrees by the different application areas. It is a continual struggle to improve the quality of the project management data when so much effort is focused on delivering new services on time. The improvements likely to make most impact in the short term are:

- Making the measurement information more readily available to individual project teams.
- A more rigorous approach to configuration management, including version control of requirements documents and specifications
- Moving the measurement activity to the stage when business requirements are specified as well as when each RFC is completed – this will enable measurement and subsequent prediction of the scope creep experienced and enable the functional analysis and measures to feed into test case design
- Change to using the COSMIC-FFP ^[3] functional sizing technique to better capture the functional size of the multi-layered software architectures involved in telephony applications. Recent trials with this have shown great promise
- Management commitment to improve the completeness and accuracy of work time records
- Better recording and investigation of software defects, both during testing and operation

5. Controlling requirements outsourced to a supplier

- *Case: An outsourced organisation where the 'flow of requirements' to an external supplier is controlled using measures agreed at a contractual level.*

For the past three years SMS has provided a Functional Size Audit service to a large UK retail bank. This client has outsourced all its software development and maintenance activities to a supplier via a contract valid for seven years. Each year, the supplier is obliged by the contract to measure and report the functional size of all software requirements fulfilled during

the year, whether new requirements or enhancements (additions, changes and deletions) made to existing requirements.

The supplier has several hundred software engineers involved in this contract, with one or two specialist staff dedicated to managing the collection, quality assurance, analysis and reporting of the data across the entire outsourced organisation. Around 30 staff have been trained to use the functional sizing technique (in this case, MkII Function Point Analysis) and the functional size measurements are made by trained staff from one project being seconded for short periods to the project for which measurements are needed.

The supplier is very conscientious and the measurement staff use, as well as the standard public domain references, local counting practices supported by 'case histories' for unusual situations. They operate well-defined documentation and quality control procedures to ensure that the measurements are consistent and repeatable by different individuals. The supplier's Quality Assurance group performs reviews of the practices, procedures and results, reporting issues and tracking them to resolution.

None-the-less, regardless of all these efforts by the supplier, because the measurement data is used to determine the price of the services supplied to the customer, it is of such importance that the customer needs assurance that the procedures and results are reasonable and truthful. Hence, the customer employs SMS as an independent third party 'auditor' to verify and validate the suppliers practices.

The results over three years have shown that there is a mean size error between the un-audited and the audited results of between +/-2% and +/-2.5% for each individual project. Over the entire annual workload, this reduces to an error of less than +/-1% (due to the 'swings and roundabouts' effect).

Clearly this gives very good confidence to both customer and supplier that these measurements form a stable basis for the contractual arrangements.

6. Conclusion

These cases show that organisations are using measurement techniques to improve the correctness, completeness, consistency, testability and traceability of functional requirements. Very simple techniques can be used easily to improve the quality of common requirements approaches. These techniques enable the requirements to be quantified and the measurement procedures employed and the results produced can be subject to audit and quality assurance. Hence they can form a solid foundation for pricing software, negotiating contracts and controlling scope creep.

Understanding the product size is crucial to understanding the software process and for managing project constraints such as duration, time-to-market and productivity, along with other factors that affect customer satisfaction.

Measurement techniques such as these are a necessary first step in implementing process improvements in a systematic way.

7. References

- [1] C. R. Symons & G. Jensen: "Collaborative Study of Object Oriented Software Developments", IBM Global Information Services & Software Measurement Services, limited circulation to study participants, 1998/99
- [2] UKSMA Metrics Practices Committee: "MkII Function Point Analysis Counting Practices Manual v1.3.1", UK Software Metrics Association, September 1998
- [3] Common Software Measurement International Consortium: "COSMIC-FFP Measurement Manual Version 2.0", October 1999.