Good Practice Guidelines for Software Engineering in New Zealand

March 2007

IPENZ ENGINEERS NEW ZEALAND
The Knowledge Network
The New Zealand Computer Society Inc.
The Institute of MEASUREMENT AND CONTROL NZ INC.

CONTENTS

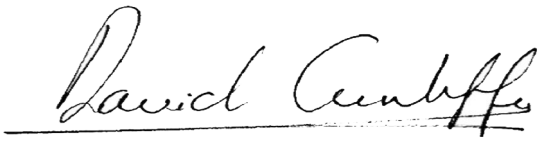
MINISTER'S FOREWORD

New Zealand's future economic development and international competitiveness is reliant upon improving our digital capability, infrastructures and skills – across government, the business sector and communities. These aims are brought together in the Labour-led Government's Digital Strategy.

Underpinning many of these areas is reliance upon international good practice and conformance with internationally-recognised standards. In the area of software engineering standards IPENZ has taken a leadership role and developed *Good Practice Guidelines for Software Engineering in New Zealand*. Increasingly, New Zealand software development companies will need to conform to accepted international benchmarks in order to remain internationally competitive. These guidelines mean that New Zealand companies will have a point of reference for the standards required by the global software market when developing new products.

The Government recognises that these guidelines are important. Industry-led initiatives, such as the development of these guidelines, assist in New Zealand's transformation into an innovative, knowledge-based, internationally-competitive economy.

Therefore, I commend the Institution and its collaborators for undertaking this valuable work.



Hon David Cunliffe

FOREWORD

New Zealand's economic development relies on good practices for developing and operating software-intensive systems. Moreover, to strengthen New Zealand's export competitiveness – both in terms of cost structures and delivered product quality – software development organisations must adopt the good practices already being used elsewhere around the world.

The financial transaction component of our economy has been largely "digital" for many years now. We are also rapidly implementing "digital government" through initiatives flowing from the Government's Digital Strategy. But these critical infrastructures for our 21st century economy fundamentally rely on trustworthy software-intensive systems.

Already our society is highly dependent on software intensive systems, and in turn their definition, development, implementation and operations. Obvious everyday examples include:

- electronic cash registers used for virtually all purchasing – whatever the final payment mechanism might be
- stock control systems
- monitoring and control systems for public utilities, transportation systems and health care
- rapid and easy web-based access to government-held information
- industrial control systems

Building and operating these kinds of critical systems are dependent on good practices across the whole software-intensive system lifecycle, such as those codified by international software engineering standards and maturity frameworks. While slavishly following good practices won't guarantee reliable or cost-effective development and operations, poor or mediocre practices make reliability and effectiveness a matter of chance rather than good management.

There are many other reasons for referring to documented good practices. They:

- foster traceability of features delivered to meet customers' requirements
- describe ways of avoiding risks caused by the failure of software-intensive systems
- provide checklists of procedures and practices that improve the overall efficiency of software development processes
- encompass considerations for building in and maintaining security, privacy and resilience to malicious attacks

In short, good practices as embodied in international software standards and frameworks are the roadmaps towards a disciplined approach to software systems development and operations.

INTRODUCTION

Engineers like solving problems – especially difficult problems. Although it has become relatively easy to commission hardware for information technology, many non-trivial problems continue to plague software development and operational processes. Software engineering problems have remained "hard".

Many of the good practices in other engineering disciplines have been codified and are enforced in nationally and internationally accepted standards. However, this doesn't generally apply to software development and operations in New Zealand. There is plenty of room for improvement in referencing to – and conforming (J Moore, personal communication)[1] with – software engineering process standards in New Zealand (Sung and Paynter, 2006).

While not a "silver bullet" solution (Brooks, 2003) to all software problems, software engineering standards can support improvement in software development and operations. Software engineering standards can also help reduce costs and complexity when specifying and evaluating components incorporating software in larger systems. International standards in software and system engineering are also excellent references for good practice in software development and operations.

Along with international economic, legal and cultural requirements, internationally accepted software engineering process standards are becoming increasingly important. Software intensive systems development and operations can be undertaken anywhere in the world – for clients anywhere in the world. So, at whatever scale they operate, New Zealand software developers will increasingly be at a competitive disadvantage if they cannot demonstrate knowledge of and adherence to software engineering standards. Successful export of value-added products that rely on software-controlled processes for manufacture, quality assurance and inventory control will also increasingly depend on demonstrable conformance and possibly obligatory compliance to software engineering process standards.

The following are some international and New Zealand examples of guidelines or requirements to comply with software engineering standards or quasi-standards:

- The Payment Card Industry (PCI) Security Standards Council's Data Security Standards (DSS) (2006) which include requirements to "develop software applications based on industry best practices and incorporate information security throughout the software development lifecycle" and to "develop all web applications based on secure coding guidelines such as the Open Web Application Security Project guidelines".
- The Association for Computing Machinery's studies of implementing voter registration databases (2006) (equivalent to New Zealand's electoral roll system) concludes that e-voting systems "technology, if engineered and tested carefully, and if deployed with safeguards against failure, can strengthen [the USA's] voting system, but…we still have work to do in meeting these goals" (Spafford, 2006).
- In New Zealand, issues with Student Management Systems (SMS) for use in primary and secondary schools led to the Ministry of Education initiating an "SMS accreditation process so schools would have reliable and consistent data about the various SMS packages available in the New Zealand market" (2005). In addition to reviewing the functionality of the SMS applications, The Ministry also reviewed suppliers' business organisation, supply of application support services, software development management and quality assurance, and technology roadmaps.
- The New Zealand Government's Communications Security Bureau's Security of Information Technology publication *NZSIT 400* (2005) – which provides guidance to government agencies for managing the risks of sharing information and includes reference to various international IT security standards.

Software engineering process standards can be used in essentially two ways (Christensen and Thayer, 2001):

- to define the "right things to do" – as normative reference model codifications of good practices to guide software engineering processes – the implicit assumption being that the quality of a software-intensive system is directly influenced by the quality of the development and operational processes
- to measure whether the "right things are being done" – that is, the degree of conformance to codifications[2] of good practices – often expressed as a score derived from a "process maturity level" framework

This document identifies some sources of software engineering-specific process standards to which practising professional engineers in New Zealand might want to refer, and it identifies situations when standards might be useful (J Dietrich, personal communication).[3]

1 Moore notes that the preferred nomenclature is "to conform", which connotes voluntary agreement to standards, as opposed to "to comply", which carries a meaning of enforced obligation.

2 "A normative document prescribes what an engineer should do in a specified situation rather than providing information that might be helpful. The normative literature is validated by consensus formed among practitioners and is concentrated in standards and related documents." *(ISO/IEC 19759)*

3 "In addition to the standards mentioned in [this document] there are several de-facto standards. These standards are not (yet) maintained by an international standard body but are either promoted by vendors and consultancy firms (example: IBM's Rational Unified Process RUP) or by communities like the Agile Alliance, http://www.agilealliance.org/ Examples include: Extreme Programming XP, Scrum, FDD. The existence of these de-facto standards reflects the very nature of the software industry where technology and methodologies used change frequently and [international] standard bodies are not agile enough to reflect this."

WHAT IS DIFFERENT ABOUT SOFTWARE?

In essence, software is a set of instructions to carry out a sequence of actions resulting in a set of observable states, each state being dependent on some or all previous inputs and intermediate states. A useful metaphor for both software and its development is solving a scrambled Rubik's cube. The initial state and goal state can be well defined – but a variety of action sequences ("algorithms" or "procedures") can be taken to achieve the desired result.

In practice, not all new individuals entering the domain of software development and operations have trained in computer science or software engineering. "Even if they have computing degrees, they may never have learnt about what has been done by people before them. Many [of these] people purposely ignore the lessons of the past. It may take a major project failure before these people realise that the problems they encountered were neither new nor unique." (Endres and Rombach, 2003)

Other engineering disciplines are generally well supported by science-based knowledge. However, the industrial practices of software engineering are not generally well supported by the theories of computer science. For example, there is little support from robust, well-accepted theories and abstractions about quantifying the complexities of real-world problem spaces.

In general, we can't accurately estimate how much effort should be required to solve a given software development problem unless we have solved a very similar problem before (Brooks, 2003). Empirical information about the effort and cost involved in successful software development is generally only anecdotal or proprietary in nature. The tie back to universal physical laws and constraints and the experiences of having solved numerous tangible engineering problems have very limited applicability in software development.

Furthermore, unlike a physical structure, it can be very difficult to discern the overall "software architecture" and quality attributes of a software system by examining the code implementing the system's algorithms (Baragry and Reed, 2001). This accounts for the significant effort required to reconstruct the software architecture of an operating system once the original creators have moved on and knowledge of the decision-making considerations that resulted in the design is lost (O'Brien et al, 2002; P Kruchten, personal communication; Kruchten, 2004). It's "about as easy as reconstructing a pig from a sausage" (Eastwood, 1993).

WHY WORRY ABOUT SOFTWARE ENGINEERING PROCESS STANDARDS?

Professional engineers involved in specifying, developing, evaluating or managing software – or specifying and evaluating systems incorporating software-intensive components – should ask the following questions:

- To what extent should, and does, the development process comply with recognised standards?
- Are the standards complied with appropriate for the development being undertaken?
- What are the potential problems that could arise if the software fails to perform as required? As articulated in *IEC 61508* and the accompanying paper (Durdle, 2006), this should be assessed and expressed in similar terms to the outcomes of a hardware engineering risk analysis.
- If there is some degree of non-compliance, what is the impact on the process, the software or system being delivered and the resulting risks?
- If the client is not aware of or concerned about compliance to appropriate standards, how should the impacts of the above be communicated?

HOW CAN ENGINEERING STANDARDS HELP SOFTWARE DEVELOPMENT?

Software engineering is the part of systems engineering (Doran, 2006)[4] that deals with the systematic development, evaluation and maintenance of software. Software engineering standards can provide the equivalent of instructions to "cheat" by disassembling the Rubik's cube and piecing it back to achieve a desired result. While standards can provide guidance as to how to decompose the total problem space into coordinated component problems, they won't tell the developer how to create new algorithmic procedures. So in that sense, standards are only part of an engineering approach to software system development. But they can form the basis of a robust, auditable and repeatable quality management approach to tackling the hard problems of software-intensive large-system development.

Some of the advantages of conforming with software engineering standards include (E Tempero, personal communication):

- risk management – process standards have often been formulated purposefully to identify, address, reduce and avoid risk, as discussed in *Best Practice Guidelines for Risk Management of Software-based Systems* (IPENZ, 2006)
- predictability – by following consistent processes the resource requirements for similar kinds of activity will become easier to identify and estimate
- audit – to satisfy customers' expectations that appropriate processes have been followed to produce the end product, much as "Producer Statements" do to comply with the Building Act 1991
- health and safety requirements – particularly in order to demonstrate that appropriate design consideration has been taken for life-critical and safety-critical software systems
- service differentiation – as a marketing feature, demonstrable adherence to standards might well be perceived as added value by customers who can choose between alternative suppliers of software engineering services

However, some factors to consider before mandating "heavyweight" software engineering process standards are (Glass, 2003; Garcia, 2005):

- size matters – small development projects that involve a handful of developers are vastly easier than larger projects
- application domain matters – for example the mathematical formalism often needed for scientific applications is typically unnecessary for business oriented software intensive systems
- criticality matters – if lives or vast sums of money are impacted by the results of a project it should be treated carefully, especially in respect to reliability requirements
- innovativeness matters – if the problem being addressed is quite unlike one previously solved, an exploratory and less process-driven approach to its solution may be appropriate
- what the competition is doing matters – complying with software engineering process standards might well become a requirement for entry into some markets, or remaining in existing markets
- poor performance matters – adopting software engineering process standards might help solve difficulties in getting the right quality level of software delivered to customers on time and within budget
- required investment matters – adopting software engineering process standards incurs costs in terms of process development, deployment, maintenance and appraisal
- organisational culture matters – key considerations include business strategy and goals, embedded work practices, potential for transformation and the appetite for change

Nevertheless, software engineering process standards are important to software quality assurance as they codify good practices and they go a long way to address the distinctive problems of non-trivial software development noted above. The assumption that software engineering process standards are too expensive and difficult for implementation by small-sized software development organisations is challenged by reported successes from using tailored approaches – for example in Australia (Cater-Steel, 2004), Brazil (von Wangenheim et al, 2006) and various countries in Europe (Lawthers, 1999).

An historical issue was the proliferation of software engineering standards and standards setting bodies, resulting in the quip "the nice thing about standards is that there are so many to choose from".[5] At one time, 55 producers of software engineering standards were active publishing more than 300 standards (Magee and Thiele, 2004). However, this "quagmire" (Sheard, 2000) has now solidified to a smaller set of key producers and standards (Moore, 1998; Brotbeck et al, 1999).

From a New Zealand perspective, the key internationally accepted software engineering development process standards (and quasi standards) producers are:

- IEC – the International Electrotechnical Commission
- ISO – the International Organization for Standardization
- JTC1 – the ISO/IEC Joint Technical Committee and its subcommittees working on IT standards
- IEEE – the Institute of Electrical and Electronics Engineers, Software and Systems Engineering Standards Committee
- SEI – the Software Engineering Institute
- OMG – the Object Management Group

4 Standards for the wider area of systems engineering are being harmonised and include *ISO/IEC 15288 Systems Engineering – System Lifecycle Processes; IEEE 1220 IEEE Standard for Application and Management of the Systems Engineering Process.*

5 Attributed to A Tannenbaum.

While the standards produced by these bodies aren't always consistent and at the same level of detail, they do define a common set of processes that are generally adequate for current use. In practice, the processes can be tailored to meet development requirements and capabilities – but caution should be exercised should contractual or regulatory requirements dictate compliance.

As of 2006 some New Zealand-specific IT standards have been mandated or are under development for government agencies:

- e-Government Interoperability Framework Standards relating to Network; Data Integration; Business Services; Access and Presentation; Web Services; Security; "Best Practices" in Digital Rights Management, Trusted Computing, Business Process Execution Language, Business Data Transformation, Data Modelling, Processing Structured Data, Document File Formats; Meta Data relating to government agencies; Authentication; Email Security and Intranet usage
- guidelines for the management and design of public sector websites
- the New Zealand Government Locator Service (NZGLS) Metadata Element Set, providing a set of metadata elements designed to improve the discovery, visibility, accessibility and interoperability of online information and services
- process guidelines for managing and monitoring major IT projects published in August 2001 by the State Services Commission

In general, these government-adopted standards and guidelines are representative of industry-acknowledged international good practices.

Internationally, the following standards and framework are accepted as normative prescriptions and measures of "good practice" for software engineering development processes:

- *ISO/IEC 12207 Information Technology – Software Lifecycle Processes*
- *ISO/IEC TR 19759 Software Engineering – Guide to the Software Engineering Body of Knowledge*
- *ISO/IEC 15504 Information Technology – Software Process Assessment*
- SEI's Capability Maturity Model Integration

ISO/IEC 12207

ISO/IEC 12207 "establishes a common framework for software lifecycle processes, with well-defined terminology, that can be referenced by the software industry". It "provides a process that can be employed for defining, controlling, and improving software lifecycle processes".[6] The IEEE and Electronics Industries Alliance (EIA) added to *ISO/IEC 12207* by publishing three additional standards forming part of the IEEE Software and Systems Engineering Standards collection:

> *IEEE/EIA 12207.0 Standard for Information Technology – Software lifecycle processes.* This contains *ISO/IEC 12207* in its entirety and includes six additional annexes which address:
  - basic concepts – such as categorising lifecycle processes into primary, supporting, organisational and tailoring
  - compliance situations, levels and criteria
  - lifecycle process objectives – acquisition, audit, configuration management, development, documentation, improvement, infrastructure, joint review, maintenance, management, operation, problem resolution, quality assurance, supply, training, validation, verification
  - lifecycle data objectives
  - relationships between standards
  - errata

> *12207.1 Standard for Information Technology – Software lifecycle processes.* Lifecycle data, which provides additional guidance on recording lifecycle data.

> *12207.2. Standard for Information Technology – Software lifecycle processes.* Implementation considerations, which provides additions, alternatives and clarifications to *ISO/IEC 12207* based on US industrial experience.

"*ISO/IEC 12207* is intended to be independent of development technologies and methodologies and useful for any form of lifecycle model, for example, waterfall, incremental, spiral, etc.. In fact, one of the specified responsibilities of the supplier's role is to select a lifecycle model and map the requirements of the standard to that model." (Moore, 2006)[7]

ISO/IEC TR 19759

ISO/IEC TR 19759 (International Organization for Standardization, 2005) is a republication of a document sponsored by the IEEE Computer Society commonly referred to as the guide to the "SWEBOK" (Software Engineering Body of Knowledge) (IEEE Computer Society Software Engineering Coordinating Committee, 2004). Officially it is a Technical Report of type 3: "when the joint technical committee has collected data of a different kind from that normally published as an International Standard ("state of the art", for example)". *ISO/IEC TR 19759* represents a broad consensus within the professional software engineering community of the processes of – and good practices in – software engineering for both development and operations through identifying and describing the body

6 From *ISO/IEC 12207*, Section 1.1: Purpose.

7 This is an excellent guide to the use of the IEEE Software and Systems Engineering Standards (and those IEEE standards shared with ISO/IEC) as a codified set of knowledge and good practices in accordance with *ISO/IEC TR 19759*.

of knowledge that is generally accepted as being "software engineering".

"From the beginning, the SWEBOK project was conceived as having a strong relationship to the normative literature of software engineering. The two major standards bodies for software engineering (IEEE Computer Society Software and Systems Engineering Standards Committee and ISO/IEC JTC1/SC7) are represented in the project. Ultimately, it is hoped that software engineering practice standards will contain principles directly traceable to the Guide."

ISO/IEC TR 19759 is subdivided into ten software engineering knowledge areas plus an additional section providing an overview of the knowledge areas strongly related to software engineering process disciplines:

- software requirements
- software design
- software construction
- software testing
- software maintenance
- software configuration management
- software engineering management
- software engineering process
- software engineering tools and methods
- software quality
- disciplines related to software engineering: computer engineering, computer science, management, mathematics, project management, quality management, software ergonomics, systems engineering

Appendix C of *ISO/IEC TR 19759* maps various IEEE Computer Society Software and Systems Engineering standards and ISO/IEC software engineering standards to the ten software knowledge areas.

The material covered by ISO/IEC TR 19759 will be extended as its users' needs evolve. For example, consideration of security aspects in software development activities and software itself has become a topic of increasing concern for high integrity systems in both commercial and governmental domains. To address these new needs, as of late 2006 the United States Departments of Homeland Security and Defense are developing two related documents:

- *A Guide to the Common Body of Knowledge to Produce, Acquire, and Sustain Secure Software* (Redwine, 2006)
- *Security in the Software Lifecycle* (Goertzel, 2006) – a source of information and guidance for good software development practices focused on security

Content from these documents might well be incorporated in future versions of *ISO/IEC TR 19759*.

ISO/IEC 15504

ISO/IEC 15504 provides a framework for the assessment of software processes to help:

- understand the state of an organisation's software-intensive systems development processes and process improvement opportunities
- determine the suitability of an organisation's software-intensive systems development processes for a particular requirement or class of requirements

ISO/IEC 15504 provides a common approach to describing the results of process assessment while allowing for some degree of comparison of assessments based upon different but compatible models and methods. The sophistication and complexity required of a process is dependent upon its context. For instance, the planning required for a five-person project team is much less than for a fifty-person team. The *ISO/IEC 15504* and CMMI frameworks have been mapped to each other by the Software Quality Institute at Griffith University in Brisbane (Rout et al, 2000).

The major benefits of the *ISO/IEC 15504* approach to process assessment are that it:

- reduces uncertainties in selecting suppliers of software-intensive systems by enabling the risks associated with the supplier's capabilities to be identified
- enables appropriate controls to be put in place to manage the risks of software-intensive systems acquisition or development
- provides a quantified basis for choice in balancing business needs, requirements and estimated project cost against the capabilities of competing sources of software-intensive systems and components of such systems
- provides a public, shared approach for process assessment
- provides a common understanding of the use of process assessment for process improvement and capability evaluation
- facilitates capability evaluation in procurement
- can be controlled and regularly reviewed in the light of experience of use
- can be changed only by international consensus

The *ISO/IEC 15504* reference model architecture has two dimensions:

- a process dimension for software-intensive systems development, largely aligned to *ISO/IEC 12207* – for which example "base practices" for each process are listed below
- a process capability dimension

ISO/IEC 15504 defines two "primary" lifecycle process categories:

- customer-supplier processes that directly impact the customer, support development and transition of software to the customer and provide for the correct operation and use of the software
- engineering processes that directly specify, implement, or maintain the software product, its relation to the system and its documentation

The customer-supplier category is broken down into the following processes:

- acquisition: preparation, supplier selection, supplier monitoring, customer acceptance
- supply
- requirements elicitation
- operation: operational use, customer support

The engineering category is broken down into the following

TABLE 1: ISO/IEC 15504 REFERENCE MODEL CAPABILITY LEVELS

Capability Level	Description
Level 0: Incomplete	There is general failure to attain the purpose of the process. There are few or no easily identifiable work products or outputs of the process.
Level 1: Performed	The purpose of the process is generally achieved. The achievement may not be rigorously planned and tracked. Individuals within the organisation recognize that an action should be performed and there is general agreement that this action is performed as and when required. There are identifiable work products for the process and these testify to the achievement of the purpose.
Level 2: Managed	The process delivers work products according to specified procedures and is planned and tracked. Work products conform to specified standards and requirements. The primary distinction from the Performed Level is that the performance of the process now delivers work products that fulfil expressed quality requirements within defined timescales and resource needs.
Level 3: Established	The process is performed and managed using a defined process based upon good software engineering principles. Individual implementations of the process use approved, tailored versions of standard and documented processes to achieve the process outcomes. The resources necessary to establish the process definition are also in place. The primary distinction from the Managed Level is that the process of the Established Level is using a defined process that is capable of achieving its process outcomes.
Level 4: Predictable	The defined process is performed consistently in practice within defined control limits, to achieve its defined process goals. Detailed measures of performance are collected and analysed, leading to a quantitative understanding of process capability and an improved ability to predict and manage performance. Performance is quantitatively managed. The quality of work products is quantitatively known. The primary distinction from the Established Level is that the defined process is now performed consistently within defined limits to achieve its process outcomes.
Level 5: Optimising	Performance of the process is optimised to meet current and future business needs and the process achieves repeatability in meeting its defined business goals. Quantitative process effectiveness and efficiency goals (targets) for performance are established based on the business goals of the organisation. Continuous process monitoring against these goals is enabled by obtaining quantitative feedback and improvement is achieved by analysis of the results. Optimising a process involves piloting innovative ideas and technologies and changing non-effective processes to meet defined goals or objectives. The primary distinction from the Predictable Level is that the defined and standard processes now dynamically change and adapt to effectively meet current and future business goals.

development processes:

- system requirements analysis and design
- software requirements analysis
- software design
- software construction
- software integration
- software testing
- system integration and testing

An additional engineering process is also defined at a basic level – system and software maintenance.

ISO/IEC 15504 "supporting" lifecycle processes are those that may be employed by any of the other processes at various points in the software lifecycle. They include:

- documentation
- configuration management
- quality assurance
- verification
- validation
- joint review
- audit
- problem resolution

ISO/IEC 15504 "organisational" lifecycle processes establish the business goals of the organisation and develop processes, products, and resource assets which, when help the organisation achieve its business goals. They include:

- management processes: project management, quality management, risk management
- organisational processes: organisational alignment, improvement processes – process establishment, process assessment, process improvement
- human resource management
- infrastructure
- measurement
- reuse

The six capability levels in the *ISO/IEC 15504* reference model are described in Table 1 (see page 9).

CMMI

CMMI was developed by the SEI and is more widely used in the USA than *ISO/IEC 15504* for which it is essentially an alternative assessment framework. CMMI provides a measurement framework to assess process maturity for systems engineering, software engineering, integrated product and process development and supplier sourcing. CMMI is also a process measurement and improvement approach that defines the elements of effective processes for software development and operations. It can be used to guide process improvement across a project, a division, or an entire organisation.

CMMI helps integrate organisational functions, set process improvement goals and priorities, provide guidance for quality processes, and provide a point of reference for appraising current processes (Carnegie Mellon University Software Engineering Institute, 2007). A CMMI "staged representation" metric commonly quoted is the "maturity level" which describes the organisation's overall maturity in software engineering processes from 1 (initial, low maturity) to 5 (optimised, high maturity).

The CMMI is a descriptive framework – it doesn't specify how organisations should satisfy the specific practices identified at each maturity level. Rather it focuses on what should be demonstrated. The IEEE Software and Systems Engineering Standards (for example) define how to implement specific practices to fulfil the CMMI requirements (for example, Land, 2005). They define processes along with the work products (also known as "artefacts") that provide support to and result from the development processes.

Other Sources of Software Engineering Development Process and Related Standards

Another potential source of good practice guidance for New Zealand software-intensive systems engineering is the British Standards Institute-initiated TickIT website (2005). TickIT is essentially a UK-focused accreditation scheme for certification organisations that conduct ISO 9000 audit/registration for software development companies. A TickIT accredited ISO 9001 certificate is roughly equivalent to between CMMI Maturity Levels 2 and 3. The TickIT Web site offers:

- "Getting The Measure Of TickIT – Guidance and Information about the emerging ISO measurement standards for improving software processes and how they relate to ISO 9001", published in February 2002
- "TickIT Reference List – An extension of the standards list in appendix 5 of Issue 5 TickIT Guide including details of standards databases, an extensive reading list and procurement information", published with revisions in July 2001

Software Architecture Standards

One of the major challenges to large software-intensive systems development is establishing the "software architecture" at the early stages of development. An April 2004 report jointly released by the Royal Academy of Engineering and the British Computer Society noted:

"In general, the significance of architecture for complex IT projects also tends to be poorly appreciated."

The SEI has developed elements of a "Software Architecture Lifecycle Integration" process approach to defining and analysing software architectures for large software-intensive systems development (2007):

- The Quality Attribute Workshop (QAW) offers a method for eliciting quality attribute requirements.
- The Attribute-Driven Design (ADD) method provides an approach to define software architectures by basing the design process on the system's quality attribute requirements.
- The Active Reviews for Intermediate Designs (ARID) concentrate on whether the software architecture design being proposed is suitable from the point of view of other parts of the architecture that must use it.
- The Architecture Trade-off Analysis Method (ATAM) helps a system's stakeholders understand the consequences of software architectural decisions with respect to the

TABLE 2: CMMI

CMMI Maturity Level	Process Area	Number of Specific Practices that Must Be Demonstrated Above Lower Maturity Level
1: Initial Ad hoc – occasionally chaotic – processes that rely on individual "heroes and heroics" to deliver "death march" (Yourdon, 2004) projects.		
2: Managed Basic project management disciplines are in place to allow repeatable success for similar projects	Requirements Management Project Planning Project Monitoring and Control Process and Product Quality Assurance Configuration Management Supplier Agreement Management Measurement and Analysis	15 24 20 14 17 17 18 125
3: Defined A wider range of software engineering processes for development and operations are documented and complied – possibly with some tailoring.	Requirements Development Technical Solution Product Integration Verification Validation Organisational Process Focus Organisational Process Definition Organisational Training Integrated Project Management Risk Management	20 21 21 20 17 19 17 19 20 19 193
4: Quantitatively Managed Detailed metrics are collected and used to manage the quality of both the software engineering processes and end deliverables.	Organisational Process Performance Quantitative Project Management	17 20 37
5: Optimised Continuous software engineering process improvement based on quantitative analyses.	Organisational Innovation and Deployment Causal Analysis and Resolution	19 17 36

system's quality attribute requirements and business goals.

- The Cost Benefit Analysis Method (CBAM) and Software Architecture Comparison Analysis Method (SACAM) are methods for architecture-based economic and business-goal analyses of software-intensive systems to help the system's stakeholders choose between software architectural alternatives.

In addition, various volumes in the SEI Series in Software Engineering define good practices and for software architecture development Bass et al, 2003). The book *Documenting Software Architectures: Views and Beyond* (Clements et al, 2002) represents a de facto application guide to the rather abstract *IEEE Standard 1471 Recommended Practice for Architectural Description of Software-Intensive Systems*. As of 2006, *IEEE 1471* is in the process of being adopted as an ISO/IEC standard (J Moore, personal communication).

Other open standards for "enterprise information systems architecture" – from the definition of "business architecture" requirements and processes, through to information architecture, technology and software implementation, documentation and change management – include:

- the Open Group Architecture Framework ("TOGAF") Version 8.1 "Enterprise Edition" (2006)
- the National Association of Chief Information Officers' Enterprise Architecture Toolkit version 3.0 (2006)
- the US Federal Enterprise Architecture Reference Models (2006)

Software-intensive Systems Modelling Notation Standards

Graphical and mathematical representations of real objects are important for engineering analysis and design in the physical world. The equivalent representations for software-intensive systems make use of various modelling notations, one of which has been published as *ISO/IEC 19501 Information technology – Open Distributed Processing – Unified Modelling Language (UML) Version 1.4.2*[8] (UML release 2.0 is also available from the Object Management Group, 2007). Another modelling notation – the Business Process Modelling Notation (BPMN) – is emerging as a de facto standard for describing the business processes and software-intensive system interactions.

UML is a graphical notation for visualising, specifying, constructing and documenting the artefacts of a software-intensive system. UML offers a standard way to write a system's planning documents, including conceptual things such as business processes and system functions, as well as concrete things such as programming language statements, database schemas, and reusable software components. It provides a commonly understood notation set for use by developers of "object-oriented" software-intensive systems – from architecture design to intermediate-level descriptions of implementation.

From *ISO/IEC 19501*:
"Developing a model for an industrial-strength software system prior to its construction or renovation is as essential as having a blueprint for large building. Good models are essential for communication among project teams and to assure architectural soundness. We build models of complex systems because we cannot comprehend any such system in its entirety. As the complexity of systems increase, so does the importance of good modelling techniques. There are many additional factors of a project's success, but having a rigorous modelling language standard is one essential factor."

UML defines the following graphical diagrams for software-intensive systems:

- use case diagrams
- class diagrams
- behaviour diagrams – statecharts, activity diagrams, sequence and collaboration interaction diagrams, and component and deployment implementation diagrams

UML has some shortcomings in relation to systems modelling, which have led to the development of extensions to its language by the SysML language being developed by the SysML Partners forum (Open Source Specification Project, 2006).

BPMN (Object Management Group, 2007)[9] is a graphical notation for business process modelling aimed at business users. It is gaining widespread industry acceptance as a process-centric (as opposed to UML's object-centric) approach that is more intuitive for business analysts. BPMN focuses on modelling control and message flows. BPMN also offers the option of explicitly modelling business objects that may be exposed through business services.

BPMN and UML complement each other. Business analysts will likely use BPMN as a front-end modelling language and software developers will use UML for subsequent technically-oriented systems development.

8 Unified Modelling Language (UML) is undergoing enhancement with the latest specification (version 2.0) being available for free download from http://www.omg.org/technology/documents/formal/uml.htm The UML specification version 1.4.2 adopted by ISO/IEC is available for free download from http://www.omg.org/cgi-bin/doc?formal/05-04-01

9 Business Process Modelling Notation (BPMN) Version 1.0 Specification was adopted by the Object Management Group on 6 February 2006.

HOW CAN STANDARDS HELP SOFTWARE-INTENSIVE SYSTEM OPERATIONS?

Software-intensive systems operations is an area of increasing focus as typically some 80 per cent of total cost of ownership relates to ongoing "service management" costs. Standards for software systems operations are important because:

- they embody good practices for service management of software-intensive systems
- management of software-intensive systems is often critical to the success of business and government organisations
- they can help define and manage the organisation's policies, internal controls and business practices in respect to software intensive systems in cost-effective ways because they represent codified good practices available for reuse
- they can provide efficiency gains; reduce reliance on experts; reduce errors; and they can provide documentation for auditable procedures

Internationally, two sets of standards and one framework have become increasingly accepted as normative prescriptions and measures of good practices for software engineering operations processes:

- the *ISO/IEC 20000* standards
- ISO/IEC 17799-1 Code of Practice for Information Security Management and ISO/IEC 27001 (an update to ISO/IEC 17799-2)
- Control Objectives for Information and related Technology (COBIT) framework

ISO/IEC 20000

In 2005, the ISO/IEC released:

- *ISO/IEC 20000-1: Information technology – Service management – Specification*
- *ISO/IEC 20000-2: Information technology – Service management – Code of practice*

The ISO/IEC 20000 standards cover good practices for service delivery and service support. They address the following aspects:

> service delivery processes
  - service level management
  - service reporting
  - service continuity and availability management
  - sudgeting and accounting for IT services
  - capacity management
  - information security management

> relationship processes
  - business relationship management
  - supplier management

> resolution processes
  - incident management
  - problem management

> control processes
  - configuration management
  - change management

> release process
  - release process management

The UK Office of Government Commerce's IT Infrastructure Library (ITIL) publications[10] are becoming generally accepted as a standard source of good practices in process management for software assets, service support, service delivery, IT infrastructure management and security management in support of the ISO/IEC 20000 standards.

ISO/IEC 17799-1 and ISO/IEC 27001

ISO/IEC 17799-1 provides a framework for information security management and management practices to improve the reliability of information security in inter-organisational relationships. *ISO/IEC 17799-1* documents good practice in information security management systems as a "Code of Practice". *ISO/IEC 27001* is a "specification for information security management systems". Areas addressed include:

- organisational security
- asset classification and control
- personnel security
- physical and environmental security
- communications and operations management
- access control
- systems development and maintenance
- business continuity management
- compliance

A related security standard *ISO/IEC 15408 Security Techniques – Evaluation Criteria for IT Security* (also known as the Common Criteria for IT Security Evaluation[11]) defines criteria for a common and comparable evaluation of IT security, focusing on the security of systems and products.

Another source of information about good information security practices is the *Standard of Good Practice* (SoGP) (2005) published biannually by the Information Security Forum (ISF). It is available free of charge from the ISF. The SoGP is developed based on the practices of and incidents experienced by large organisations around the world. The SoGP can be used as a governing document for information security by itself or in conjunction with other standards such as *ISO/IEC 17799-1*, *ISO/IEC 27001* and COBIT.

10 Refer to Office of Government Commerce Web site at http://www.itil.co.uk/

11 Unofficial versions of the Common Criteria and Common Evaluation Methodology released for public consultation can be downloaded from http://www.commoncriteriaportal.org/public/expert/index.php?menu=3

COBIT

COBIT (Information Systems Audit and Control Association, 2005) is a comprehensive framework for software-intensive systems governance, providing management tools such as metrics and maturity models to complement a governance control framework. COBIT defines 34 software-intensive systems operations processes grouped along the lines of the classic four-phase plan-do-check-act quality management cycle. Each process is described in terms of:

- process description, metrics, practices and mapping of the process to process domains, information criteria, required resources and governance areas
- detailed control objectives for the process
- management guidelines including process inputs and outputs, a RACI (Responsible, Accountable, Consulted Informed) chart
- a maturity model for the process

The processes defined in COBIT 4.0 are outlined in Table 3.

The COBIT framework addresses governance of software service management processes within organisations by linking together:

- business requirements
- a generally accepted service delivery process model
- the major resources involved in service delivery
- management control objectives

To establish the control objectives for software-intensive systems operations and monitor performance levels COBIT defines specific:

- benchmarking of process capabilities expressed as maturity models, derived from the SEI's Capability Maturity Models
- "balanced scorecard" goals and metrics for process performance and outcome measurement
- goals for getting software-intensive systems processes under control

TABLE 3: COBIT 4.0 PROCESSES

Plan and Organize:	Acquire and Implement:
Define a strategic IT plan	Identify automated solutions
Define the information architecture	Acquire and maintain application software
Determine technological direction	Acquire and maintain technology infrastructure
Define the IT processes, organisation and relationships	Enable operation and use
Manage the IT investment	Procure IT resources
Communicate management aims and direction	Manage changes
Manage IT human resources	Install and accredit solutions and changes
Manage quality	
Assess and manage IT risks	
Manage projects	
Monitor and Evaluate:	**Deliver and Support:**
Monitor and evaluate IT performance	Define and manage service levels
Monitor and evaluate internal control	Manage third-party services
Ensure regulatory compliance	Manage performance and capacity
Provide IT governance	Ensure continuous service
	Ensure systems security
	Identify and allocate costs
	Educate and train users
	Manage service desk and incidents
	Manage the configuration
	Manage problems
	Manage data
	Manage the physical environment
	Manage operations

In early 2007 it is planned that a fine-tuned COBIT 4.1 will be released along with several other revised publications to create an aligned COBIT 4 series of publications (Information Systems Audit and Control Association, 2006).

An alternative top-down approach to defining governance for software-intensive systems operations is Australian *Standard AS 8015 – Corporate governance of information and communications technology.* This standard provides guiding principles for owners, board members, directors, partners, senior executives, or similar on the effective, efficient, and acceptable use of information and communication technology. *AS 8015* is quite brief – the substantive content is less than five pages. It defines six high-level principles with application guidance expected to be released in subsequent handbooks to accompany the standard.

The *AS 8015* governance principles are:

- establish clearly understood responsibilities for information and communications technology
- plan information and communications technology to best support the organisation
- acquire information and communications technology validly
- ensure information and communications technology performs well whenever required
- ensure information and communications technology conforms with formal rules
- ensure information and communications technology use respects human factors

CONCLUSION

Software engineering is still a rapidly developing discipline. There remain many challenges in addressing the persistent "software crisis". Yet there are also many proven software engineering-specific techniques and approaches derived from traditional professional engineering disciplines that have been codified as good practices in internationally generally accepted software engineering standards. These standards are being harmonised into a corpus of consistent standards that the international professional software engineering community can refer to as generally accepted knowledge, good practices and measures to guide responsible professional conduct in software engineering.

Ethically, it behoves practicing professional software engineers in New Zealand to be aware of the good professional practices described in the standards outlined in this paper. Consistent with the practices of other professional engineering disciplines, New Zealand professional software engineers should endeavour to apply these good practices to solving their client's problems using processes that demonstrably address:

- client needs
- timeliness
- cost-effectiveness
- ethical requirements including sustainability
- risk awareness and management

ACKNOWLEDGEMENTS

These guidelines were developed by Duncan Hall. Many individuals provided support and review during the preparation of this document. They include:

A Holt
B Durdle
C Skinner
D Montgomery
E Tempero
J Moore
J Dietrich
M Milner
N Procter
New Zealand Computer Society anonymous reviewer(s)
P Croll

REFERENCES

Association for Computer Machinery Committee on Guidelines for Implementation of Voter Registration Databases 2006, *Study of Accuracy, Privacy, Usability, Security, and Reliability Issues.* Retrieved 8 December 2006 from http://www.acm.org/usacm/VRD_report.pdf

Baragry, J & Reed, K 2001, Why We Need A Different View of Software Architecture, *Proceedings of the Working IEEE/IFIP Conference on Software Architecture*, pp125–134.

Bass, L, Clements, P & Kazman, R 2003, *Software Architecture in Practice*, (Second Edition), Addison-Wesley & Pearson Education.

Brooks, F 2003, *A Handbook of Software and Systems Engineering – Empirical Observations, Laws and Theories*, Fraunhofer Institut Experimentelles Software Engineering, Pearson.

Brooks, F 2003, Three Great Challenges for Half-Century-Old Computer Science, *Journal of the ACM,* 50(1), pp25–26.

Brotbeck, G, Miller, T & Statz, J 1999, *A Survey Of Current Best Practices And Utilization Of Standards In The Public And Private Sectors,* State of Texas Department of Information Resources.

Carnegie Mellon University Software Engineering Institute 2007, *Capability Maturity Model Integration*. Retrieved 11 February 2006 from http://www.sei.cmu.edu/cmmi

Carnegie Mellon University Software Engineering Institute 2007, *Software Architecture Life-Cycle Integration*. Retrieved 26 August 2006 from http://www.sei.cmu.edu/activities/architecture/arch_lci.html

Cater-Steel, A 2004, Low-rigour, Rapid Software Process Assessments for Small Software Development Firms, *Australian Software Engineering Conference Proceedings*, pp368–377.

Clements, P et al 2002, *Documenting Software Architectures: Views and Beyond,* Addison-Wesley & Pearson Education.

Common Criteria 2007, *The Common Criteria Project.* Retrieved 4 March 2006 from http://www.commoncriteriaportal.org/

Doran, T 2006, IEEE 1220: For Practical Systems Engineering. *IEEE Computer*, May 2006, pp92–94.

Durdle, B 2006, *Best Practice Guidelines for Risk Management of Software-based Systems*, IPENZ.

Eastwood, A 1993, It's a Hard Sell – and Hard Work Too, *Computing Canada*, 18(22), p35.

Endres, A & Rombach, D 2003, *A Handbook of Software and Systems Engineering – Empirical Observations, Laws and Theories*, Fraunhofer Institut Experimentelles Software Engineering. Pearson.

Federal Enterprise Architecture 2006, *Reference Models*. Retrieved 1 September 2006 from http://www.whitehouse.gov/OMB/egov/a-2-EAModelsNEW2.html

Garcia, S 2005, How Standards Enable Adoption of Project Management Practice, *IEEE Software*, September–October 2005, pp22–29.

Glass, R 2003, *Facts and Fallacies of Software Engineering*, Addison-Wesley & Pearson Education.

Goertzel, K M et al 2006, *Security in the Software Lifecycle: Making Software Development Processes - and the Software Produced by Them - More Secure*, Draft Version 1.2 (August 2006), US Department of Homeland Security. Retrieved 20 February 2007 from https://buildsecurityin.us-cert.gov/daisy/bsi/resources/dhs/87.html

Hunter, R 2001, Software Process Improvement, in *The Project Manager's Guide to Software Engineering's Best Practices,* Christensen, M & Thayer, R eds, IEEE Computer Society Press.

IEEE Computer Society Software Engineering Coordinating Committee 2004, *A Guide to the Software Engineering Body of Knowledge*. Retrieved 11 February 2006 from http://www.swebok.org

Information Security Forum 2005, *Standard of Good Practice for Information Security*, January 2005. Retrieved 20 March 2006 from http://www.isfsecuritystandard.com/index_ie.htm

Information Systems Audit and Control Association 2005, *COBIT 4.0 – Control Objectives, Management Guidelines, Maturity Models*. Retrieved 11 February 2006 from http://www.isaca.org/

Information Systems Audit and Control Association 2006, *COBIT Focus Newsletter*, vol.2. Retrieved 15 December 2006 from http://www.isaca.org/cobitnewsletter

International Organization for Standardization 2005, *ISO/IEC 19759:2005*. Retrieved 5 March 2006 from http://webstore.ansi.org/ansidocstore/subscriptions/dept.asp?dept_id=3107&pagenum=74

IPENZ 2004, *Practice Note 04* "Safety and Engineers". Retrieved from http://www.ipenz.org.nz/ipenz/forms/pdfs/PN04_Safety.pdf

Kruchten, P 2004, An Ontology of Architectural Design Decisions in Software-Intensive Systems, in *Second Groningen Workshop on Software Variability*, pp54–61.

Land, S 2005, IEEE Software Engineering Standards Support for the CMMI Project Planning Process Area, *IEEE Computer Society ReadyNote*.

Lawthers, I 1999, Software Process Improvement in Regions of Europe, *European Analysis Report*. Retrieved 5 March 2006 from http://www.cse.dcu.ie/spire/spire.html

Magee, S & Thiele, D 2004, Engineering Process Standards: State of the Art and Challenges, *IEEE IT Professional*, September–October 2004, pp38–44.

Ministry of Education 2005, *E-Admin Programme – Accredited Student Management System*. Retrieved 8 December 2006 from http://www.minedu.govt.nz/index.cfm?layout=document&documentid=10428&indexid=11255&indexparentid=5646&goto=00#TopOfPage

Moore, J 1998, *Software Engineering Standards – A User's Road Map,* IEEE Computer Society Press.

Moore, J 2006, *The Road Map to Software Engineering: A Standards-based Guide,* IEEE Computer Society Press, p205.

National Association of Chief Information Officers 2006, *Enterprise Architecture Toolkit*, Retrieved 1 September 2006 from http://www.nascio.org/

New Zealand Government 2007, *E-Government Standards.* Retrieved 11 February 2006 from http://www.e.govt.nz/standards

New Zealand Government Communications Security Bureau 2005, *NZ Government Information Technology Security Manual: NZSIT 400*. Retrieved December 2006 from http://www.gcsb.govt.nz/publications/nzsit/index.html

Object Management Group 2007, *Business Process Modelling Notation*. Retrieved 20 March 2006 from http://www.bpmn.org/

Object Management Group 2007, *Unified Modelling Language*. Retrieved 14 December 2006 from http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML

O'Brien, L, Stoermer, C & Verhoef, C 2002, *Software Architecture Reconstruction: Practice Needs and Current Approaches: Technical Report CMU/SEI-2002-TR-024*, Carnegie Mellon Software Engineering Institute, Pittsburgh.

Open Source Specification Project 2006, *Systems Modelling Language*. Retrieved 20 March 2006 from http://www.sysml.org/

Payment Card Industry Security Standard Council 2006, *Data Security Standard*, Version 1.1. Retrieved 8 December 2006 from https://www.pcisecuritystandards.org/tech/download_the_pci_dss.htm

Redwine, S T et al 2006, *Software Assurance, A Guide to the Common Body of Knowledge to Produce, Acquire, and Sustain Secure Software*, Draft Version 1.1 (September 2006) US Department of Homeland Security. Retrieved 20 February 2007 from https://buildsecurityin.us-cert.gov/daisy/bsi/resources/dhs/95.html

Rout, T, Tuffley, A & Cahill, B 2000, *CMMI Evaluation – Capability Maturity Model Integration Mapping to ISO/IEC 15504-2,* Defence Materiel Organisation (Australia). Retrieved 5 March 2006 from http://www.sqi.gu.edu.au/cmmi/report/top.html

Sheard, S 2000, Software Productivity Consortium, *The Frameworks Quagmire, A Brief Look*. Retrieved 11 February 2006 from http://www.software.org/Quagmire/frampapr.doc

Spafford, E 2006 (8 November), *ACM experts say more required to improve e-voting*, [Press Release], Washington: The Association for Computing Machinery. Retrieved 8 December 2006 from http://campus.acm.org/public/pressroom/press_releases/11_2006/election.cfm

State Services Commission and the Treasury 2001, *Guidelines for Managing and Monitoring Major IT Projects*. Retrieved 11 February 2006 from http://www.ssc.govt.nz/display/document.asp?DocID=2726

Sung, P & Paynter, J 2006, Software Testing Practices in New Zealand, in *19th Annual Conference of the National Advisory Committee on Computing Qualifications*.

The Open Group 2006, *The Open Group Architecture Framework – Version 8.1*. Retrieved 1 September 2006 from http://www.opengroup.org/togaf

The Royal Academy of Engineering 2004, *The Challenges of Complex IT Projects*. Retrieved 4 March 2006 from http://www.raeng.org.uk/news/publications/list/reports/Complex_IT_Projects.pdf

TickIT 2005, *TickIT Guide*. Retrieved 4 March 2006 from http://www.tickit.org/index.htm

von Wangenheim, C, Anacleto, A & Alviano, C 2006, Helping Small Companies Assess Software Processes, *IEEE Software*, January-February 2006, pp91-98.

Yourdon, E 2004, *Death March*, (Second Edition), Yourdon Press/Prentice Hall Professional Technical Reference.

APPENDIX 1: EXAMPLES OF KEY SOFTWARE DEVELOPMENT AND OPERATIONS ENGINEERING PROCESS STANDARDS

Source	Title
ISO/IEC	9126, Product Quality 12207, Software Lifecycle Processes; 15271, Guide to use of ISO/IEC 12207 14143, Functional Size Measurement 14598, Product Evaluation 14764, Software Maintenance 15026, Software Integrity Levels 15288, System Lifecycle Processes 15408, Security Techniques – Evaluation Criteria for IT Security 15504, Software Process Assessment 15939, Software Measurement Process 16085, Risk Management 17799-1, Code of Practice for Information Security Management 19501, Unified Modelling Language (UML) Version 1.4.2 19759, Guide to the Software Engineering Body of Knowledge (SWEBOK) (Technical Report) 20926, IFPUG 4.1 Unadjusted Functional Size Measurement Method 27001, Specification for Information Security Management Systems
IEC	61508, Functional Safety Of Electrical/Electronic/Programmable Electronic Safety-Related Systems
IEEE	610.12 Standard Glossary of Software Engineering Terminology 730 Standard for Software Quality Assurance Plans 828 Standard for Software Configuration Management Plans 829 Standard for Software Test Documentation 830 Recommended Practice for Software Requirements Specifications 982.1 Standard Dictionary of Measures to Produce Reliable Software 1008 Standard for Software Unit Testing

Source	Title
IEEE cont.	1012 and 1012 Standard for Software Verification and Validation 1016 Recommended Practice for Software Design Descriptions 1028 Standard for Software Reviews 1042 Standard for Software Configuration Management 1044 Standard Classification for Software Anomalies 1045 Standard for Software Productivity Metrics 1058 Standard for Software Project Management Plans[12] 1061 Standard for a Software Quality Metrics Methodology 1062 Recommended Practice for Software Acquisition 1063 Standard for Software User Documentation 1074 Standard for Developing Software Lifecycle Processes 1220 Application and Management of the Systems Engineering Process[13] 1228 Standard for Software Safety Plans 1233 Guide for Developing System Requirements Specifications 1362 Guide for Information Technology-System Definition – Concept of Operations 1471 Recommended Practice for Architectural Description of Software-Intensive Systems 1490 A Guide to the Project Management Body of Knowledge 1517 Software Lifecycle Processes – Reuse Processes 2001 Recommended Practice for Internet Practices – Web Page Engineering[14]
SEI	Capability Maturity Models – for example CMM Integrated: CMMI
Standards New Zealand	NZMP 6653 APEC-TEL Information Systems Security Standards Handbook (contains references to a wide range of IT security standards)
Standards Australia	HB 90.9 Software Development Guide to ISO 9001 HB 231 Information Security Risk Management Guidelines AS 8015 Corporate Governance of Information and Communications Technology

12 *IEEE Std 1058* is being merged with *ISO/IEC TR 16326* on the same subject. The result will be published by both IEEE and ISO/IEC with the number 16326 (J Moore, personal communication).

13 *IEEE Std 1220* has been fast-tracked into JTC 1. The ISO/IEC number is not yet known (J Moore, personal communication).

14 *IEEE Std 2001* has been fast-tracked into JTC 1 as *ISO/IEC/IEEE 23026* (J Moore, personal communication).

For more information, contact:

IPENZ
ENGINEERS NEW ZEALAND

National Office
PO Box 12 241
Wellington 6144
New Zealand

T 64 4 473 9444
F 64 4 474 8933
E ipenz@ipenz.org.nz
W www.ipenz.org.nz