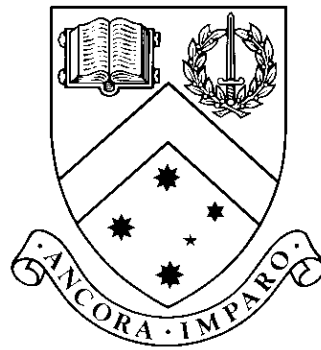


Model-based Software Testing Technology Change Management

by

Daniel Luke Rowley



Thesis

Submitted by Daniel Luke Rowley

in partial fulfillment of the Requirements for the Degree of
Bachelor of Software Engineering with Honours (2150)

in the School of Computer Science and Software Engineering at

Monash University

Monash University

November, 2002

© Copyright

by

Daniel Luke Rowley

2002

Contents

List of Figures	v
Abstract	vii
Acknowledgments	ix
1 Introduction	1
2 Software Testing Process Improvement	3
2.1 Defect Prevention	3
2.2 Software Testing Process Change Management	3
2.3 Software Testing Technology Change Management	4
2.3.1 Software Testing Technology Evaluation Approaches	4
3 Process Mapping	8
4 Accelerating Software Testing Technology Adoption	12
4.1 Three-layer Process Map Architecture	12
4.1.1 Activity Diagram Notation	12
4.1.2 Topmost-layer: Policy Management	14
4.1.3 Intermediate-layers: Mechanism Management	16
4.1.4 Bottommost-layer: Work Product Management	17
4.2 Constructing Software Testing Technology Change Management Process Maps	18
4.3 Planning Software Testing Technology Evaluations and Adoptions	19
4.4 Managing Software Testing Technology Change	19

4.5	Attesting Software Testing Technology Evaluations and Adoptions	19
4.6	Communicating Software Testing Technology Change	20
4.7	Expediting Software Testing Technology Change Management	20
4.8	Maintaining Software Testing Technology Change Management Process Maps	20
5	Case Studies	21
5.1	Constructing a Policy Map	21
5.2	Constructing a Mechanism Map	22
5.3	Modelling Organisational Software Testing Technology Change Management	22
5.3.1	ISO 9001 Software Testing Technology Change Management	22
5.3.2	CMM Software Testing Technology Change Management	25
5.3.3	CMMI Software Testing Technology Change Management	31
5.4	Modelling Team and Individual Software Testing Technology Change Man- agement	33
5.4.1	TSPi Software Testing Technology Change Management	33
5.5	General Observations	40
6	Conclusions	44

List of Figures

4.1	Three-layer Process Map Architecture	13
4.2	Activity	13
4.3	Start State	14
4.4	End State	14
4.5	Transition	14
4.6	Decision Point	15
4.7	Swimlanes	15
4.8	Fork	15
4.9	Join	16
4.10	Course-plotting	17
5.1	ISO 9001 Software Testing Technology Change Management Policy Map . .	24
5.2	ISO 9001 Software Testing Technology Change Management Policy Map with Ordering	26
5.3	ISO 9001 Software Testing Technology Change Management Mechanism Maps	27
5.4	ISO 9001 Software Testing Technology Change Management Mechanism Maps	28
5.5	Capability Maturity Model (CMM) Software Testing Technology Change Management Policy Map	30
5.6	Capability Maturity Model (CMM) Software Testing Technology Change Management Policy Map with Ordering	32
5.7	Capability Maturity Model Integration (CMMI) Software Testing Technology Change Management Policy Map	34
5.8	Capability Maturity Model Integration (CMMI) Software Testing Technology Change Management Policy Map with Ordering	35

5.9	Capability Maturity Model Integration (CMMI) Software Testing Technology Change Management Mechanism Maps	36
5.10	Capability Maturity Model Integration (CMMI) Software Testing Technology Change Management Mechanism Maps	37
5.11	Capability Maturity Model Integration (CMMI) Software Testing Technology Change Management Mechanism Maps	38
5.12	Capability Maturity Model Integration (CMMI) Software Testing Technology Change Management Mechanism Maps	39
5.13	Introductory Team Software Process (TSPi) Software Testing Technology Change Policy Map	41
5.14	Introductory Team Software Process (TSPi) Software Testing Technology Change Policy Map with Ordering	42

Model-based Software Testing Technology Change Management

Daniel Luke Rowley, BSE(Hons)
Monash University, 2002

Supervisor: Dr. SitaRamakrishnan

Abstract

A significant part of software testing process improvement effort pertains to defect prevention, software testing technology change management, and software testing process change management. Moreover, a large part of software testing technology change management effort pertains to establishing commitment and ability to perform software testing technology evaluations and adoptions.

“While technology is important and is the long-term key to improving the software process, it is rarely of much help in solving problems that are not precisely understood. Since most people object to having someone else’s solutions to undefined problems imposed on them, the unplanned introduction of technology generally causes resistance to change.” - Managing the Software Process (Humphrey, 1989).

While process documentation is able to provide the greatest visibility into the software testing technology change management process, process documentation can often be too verbose to catch onto. This research presents and justifies a mode for circumspect management of software testing technology change using Unified Modelling Language (UML) activity diagram notation [4] and Design by Contract [16] notions. Essentially, this thesis maintains that a diagramming notation that assimilates multiple levels of abstraction and connectives to narrative explanations allows an individual, project team, or organisation to effectually plan (or delineate), enact, attest, expedite, communicate, and maintain (correct, adapt, and perfect) software testing technology evaluations and adoptions. In other words, administration of software testing technology evaluations and adoptions is made easier by taking a structured, model-based approach to policy, mechanism, and work product management.

Model-based Software Testing Technology Change Management

Declaration

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.

Daniel Luke Rowley
November 5, 2002

Acknowledgments

I would like to acknowledge the generous support of my supervisor, Dr Sita Ramakrishnan.

Daniel Luke Rowley

Monash University
November 2002

Chapter 1

Introduction

A significant part of technology change management effort pertains to establishing commitment and ability to perform technology evaluations and adoptions. This research presents and justifies a mode for circumspect management of software testing technology change using Unified Modelling Language (UML) activity diagram notation [4] and Design by Contract [16] notions. Essentially, this thesis contends that a diagramming notation that assimilates connectives to narrative explanations allows an individual, project team, or organisation to effectually plan (or delineate), enact, attest, expedite, communicate, and maintain (correct, adapt, and perfect) software testing technology evaluations and adoptions. In other words, this research aims to help lay audiences compass the product of software testing technology change management group agency. In fact, an important part of forensic software engineering effort relates to evidencing (or chronicling) and communicating the identification, evaluation, and adoption of software testing technologies used in software failure investigations [19].

Although natural language constraint expressions prop up circumspect interpretation of model elements (that is, activity requirements and expectations), nested activity diagrams (in combination with connectives to narrative explanations) are used to explicate activity detail. In addition, swimlanes are used to demarcate activity context whereas colour coding is used to distinguish activity types.

Basically, this thesis propounds that accelerated interpretation of circumspect software testing technology change management practices necessitates a modelling language that assimilates constraint notation and connectives to narrative explanations. In fact, this research proposes a technique for constructing software testing technology change management process maps using a three-level architecture (or structural design) and constraint elicitation. Essentially, topmost-level diagrams are used to model policies, midway-level diagrams are used to model mechanisms, and low-level connectives are used to reference work products. In other words, this thesis contends that software testing technology change management is facilitated by a structured approach to policy, mechanism, and work product modelling.

Chapter 2 of this thesis explains software testing process improvement whereas Chapter 3 describes alternative process mapping technologies. Chapter 4 describes a methodology and rationale for constructing software testing technology change management process maps. Chapter 5 expounds on model-based software testing technology change management using case studies of prescriptive organisational software process improvement frameworks while Chapter 6 presents conclusive remarks and recommendations for further work.

Chapter 2

Software Testing Process Improvement

According to the Capability Maturity Model (CMM) [5], a significant part of software process improvement effort pertains to defect prevention, technology change management, and process change management. Furthermore, the Capability Maturity Model Integration (CMMI) [17] mandates that software process improvement is facilitated by effectual causal analysis and resolution, and organisational innovation and deployment.

2.1 Defect Prevention

According to the CMM [5], the purpose of defect prevention is to identify the cause of defects and prevent them from recurring. Although defect prevention is an important part of software test process improvement regimes, this thesis concentrates on improving the management of software testing technology change.

2.2 Software Testing Process Change Management

The CMM maintains that Process Change Management pertains to improving the software processes used in the organisation with the intention of improving software quality, decreasing the cycle time for product development, and increasing productivity. Clearly, software testing process change management relates to improving the software testing processes used in the organisation so as to improve software test documentation quality, decrease the cycle time for product testing, and increase testing productivity.

2.3 Software Testing Technology Change Management

Paulk et al. [5] comment that Technology Change Management involves identifying, selecting, and evaluating new technologies and incorporating effective technologies into the organisation. Paulk et al. allege that the objective of Technology Change Management is to improve software quality, increase productivity, and decrease the cycle time for product development. Moreover, Paulk et al. maintain that particular emphasis is placed on technology changes that are likely to improve the capability of the organisation's standard software process. Clearly, software testing technology change management relates to identifying, selecting, and evaluating new software testing technologies and incorporating effective technologies into the organisation.

2.3.1 Software Testing Technology Evaluation Approaches

Pfleeger [6] asserts that there are four types of evaluation techniques: feature analysis, survey, case study, and formal experiment. Furthermore, Brown and Wallnau [10] propound that technology evaluations are generally ad hoc and heavily reliant on the evaluation staff's skills and intuition. Brown and Wallnau argue that software development organisations use a variety of technology evaluation approaches to evaluate new technologies. Brown and Wallnau maintain that technology evaluation in practice involves

1. obtaining objective technology data by documenting case studies at other organisations;
2. (gathering) subjective opinions and experiences with the technology by attending trade shows and by conducting interviews with, or sending questionnaires to, technology vendors and users;
3. (conducting) focused experiments to mitigate high-risk aspects;
4. (demonstrating) the technology's feasibility with a pilot project;
5. (comparing) the technology to existing practices by conducting a shadow project and examining the results of both approaches; and/or
6. (initiating) demonstrator projects in the organisation to acquire phased exposure to the technology.

Brown and Wallnau assert that a typical organisation applies one or more of the approaches and, on the basis of the resultant data, forms an intuition about that technology's value. Brown and Wallnau argue that much of the informality in interpreting any evaluation's results is due to the absence of

1. well-defined goals before starting the evaluation;

2. controlled, rigorous techniques for data gathering during the evaluation;
3. a conceptual framework for analysing the resultant data in the context of existing technologies.

Brown and Wallnau claim that technology evaluation approaches can be classified as either: product-oriented or process-oriented. Brown and Wallnau argue that many organisations are able to competently conclude product-oriented decisions, however often face difficulty evaluating the potential impacts of new technologies on organisational processes. Essentially, the framework they propose to systematically evaluate the impact of software technologies on productivity and product quality relies on examining technologies in relation to their peers and predecessors. Furthermore, their framework has only be applied to component integration technologies. Whilst the technology delta evaluation framework does provide germane organisational guidance on evaluating software technologies, this thesis proposes a much simpler and more domain-specific framework that can be used by organisations to manage the impacts of new software testing technologies on organisational software testing processes. In other words, this thesis aims to present a conceptual framework for mapping prescriptive software testing technology change management frameworks at multiple levels of abstraction.

Feature Analysis

Glickman and Becker [7] propose a methodology for evaluating software tools that uses four matrices to represent negative and positive impacts of product characteristics on process characteristics such as maintainability, quality, cost, schedule, and productivity. While the methodology provides a practicable evaluation reporting format, it relies heavily upon subjectivity (or implicit opinion elicitation) to derive impact determinations. Moreover, it does not provide any specific guidance on evaluating the impact of software testing tools on productivity and quality. However, it does facilitate human understanding and communication by using normalised (numerical) ranges to rate and rank impacts according to degree of influence (that is, perform feature analysis). Rezagholi and Frey [8] also propose a method for evaluating organisational technologies and software technology change management processes (entitled MEPT (Managing Engineering and Product Technology)). Like the method proposed by Glickman and Becker, MEPT also relies heavily upon subjective (numerical) evaluation grades to conclude technology assessments. Futhermore, MEPT does not mandate any specific guidelines on assessing the impact of software testing tools on project productivity and product quality. Nevertheless, Rezagholi and Frey show that portfolios and cobweb diagramming notations are apposite methods for representing organisational technology profiles.

Surveys

Surveys are also an effectual evaluation approach to conclude software testing technology evaluation decisions.

Case Studies

Bruckhaus et al. [9] contend that software tools have long been recognised as an effective way to improve software development variables such as productivity and product quality. Bruckhaus et al. argue that effectual tool adoption necessitates understanding how a tool affects productivity and product quality in a software project. Bruckhaus et al. maintain that software tools can facilitate increased productivity by supporting software development and maintenance activities that are normally carried out with little or no tool support. Bruckhaus et al. also argue that software tools can help improve development and maintenance processes by facilitating new activities. Nevertheless, Bruckhaus et al. also warn that tool adoption can decrease productivity. Essentially, Bruckhaus et al. argue that when a tool increases your effort on specific activities or introduces new activities into the process, productivity can fall. Furthermore, Bruckhaus et al. maintain that using a tool may require more effort than working with a less sophisticated tool (or technology) such as operating system commands or pen and paper.

As maintained by Bruckhaus et al., software productivity can only improve if you can save more effort on the facilitated activities than you have to invest in activities that require more effort. Bruckhaus et al. propound that because individuals, project teams, and organisations do not know how to analyse a tool's impact on specific projects, software tools are usually adopted based on an intuitive understanding of their expected results. More importantly, Bruckhaus et al. propose that a tool's impact is not solely governed by its inherent properties, but also by the characteristics of the adopting project. Bruckhaus et al. assert that two characteristics - project size and development activities - are particularly important because they govern which tool functions you use and how often you use them. Bruckhaus et al. argue that to increase productivity, you should select a tool with your project size and development process in mind. Essentially, Bruckhaus et al. advise that tool evaluation/selection should ensure that adoption does not regress productivity to the point where you cannot afford to use it. Bruckhaus et al. state that regardless of which is more important in your project - quality or productivity - quantitative analysis of a tool's impact can help you decide if you should adopt a tool or pass it by.

Although Bruckhaus et al. do not explicitly mandate any software testing technology change management principles, the results of their case study have allowed them to present four tentative rules pertaining to generic tool adoption. Bruckhaus et al. suggest that

1. the less complex your process and the larger your project, the higher the probability that tool insertion will curtail your productivity;

2. when you adopt a more rigorous process, be prepared to increase resources even if you are improving tool support at the same time;
3. a tool's performance can peak when used in projects of a certain size, while being less productive in larger or smaller projects;
4. in terms of effort, adopting a rigorous process can be substantially less expensive if you also adopt the appropriate tools.

In summary, Bruckhaus et al. argue that tool insertion seems best justified when you plan to adopt a sophisticated process and when you are prepared to put in the extra effort that (the) process requires.

Kitchenham, Pickard, and Pfleeger [12] propound that because case studies do not have a well-understood theoretical basis, individuals, project teams, and organisations necessitate guidelines to organise and analyse case studies. Kitchenham, Pickard, and Pfleeger also recommend that personal (individual) and project team software technology evaluations are suited to case studies (because formal experiments necessitate replication) whereas organisational software technology evaluations can use case studies, formal experiments, or surveys. Essentially, Kitchenham, Pickard, and Pfleeger advise that good case studies involve:

- specifying the hypothesis under test;
- using state variables for project selection and data analysis;
- planning case studies properly;
- and using appropriate presentation and analysis techniques to assess the results.

Formal Experiments

Undoubtedly, formal experiments are rigorous controlled investigations that would be highly apposite for examining the effect of software testing tools or techniques on software testing productivity and product quality.

Chapter 3

Process Mapping

Curtis, Kellner, and Over [11] have identified five categories of desirable properties for process modelling tools and techniques:

1. facilitates human understanding and communication
2. supports process improvement
3. supports process management
4. provides automated guidance in performing the process
5. supports automated process execution

While all five categories are important, this thesis concentrates on developing process modelling technique properties pertaining to facilitating human understanding and communication and supporting software testing process improvement. More importantly, Pfleeger [6] reports that Kellner et al. contend that a process modelling technique should be validated according to its ability to include

- multiple levels of abstraction
- control flow, sequencing, and constraints on sequencing
- decision points
- iteration and feedback to earlier steps
- user creativity
- object and information management, as well as flow through the process
- object structure, attributes, and interrelationships

- organisational responsibility for specific tasks
- physical communication mechanisms for information transfer
- process measurements
- temporal aspects (both absolute and relative)
- tasks executed by humans
- professional judgement or discretion
- connection to narrative explanations
- tasks invoked or executed by a tool
- resource constraints and allocation, schedule determination
- process modification and improvement
- multiple levels of aggregation and parallelism

A diagramming notation to model software testing technology change management processes would necessitate the ability to include (no less than)

- multiple levels of abstraction
- control flow, sequencing, and constraints on sequencing
- decision points
- iteration and feedback to earlier steps
- professional judgement or discretion
- connection to narrative explanations
- process modification and improvement
- multiple levels of aggregation and parallelism

to facilitate human understanding and communication and software testing technology change management process improvement. To recapitulate, this research aims to devise a diagramming notation to model operational software testing technology change management (software testing technology identification/evaluation and adoption) practices. Clearly, individuals, project teams, and organisations aiming to assume software testing innovations necessitate a diagramming notation to communicate and visualise software testing technology evaluation and adoption process steps at different levels of abstraction. Thus, this research aims to devise a symbolic process notation to communicate and visualise software

testing technology evaluation and adoption decision-making processes at multiple levels of abstraction. This research also aims to use tenets to connect notation primitives and aggregates to narrative explanations - such guidelines will help ensure that individuals, project teams, and organisations are able to model and interpret technology change management activities at abstract and concrete levels of human understanding/communication. Most importantly, a diagramming notation to model software testing technology change processes assists an individual, project team, or organisation to reuse and maintain software testing technology change process specifications/implementations.

Essentially, an effectual diagramming notation to model software testing technology change management practices/processes allows an individual, project team, or organisation to

1. communicate, visualise, and plan/design software testing technology evaluation, adoption processes at different levels of abstraction;
2. verify and validate (that is, properly enact) software testing technology evaluation, adoption, and diffusion activities;
3. attest software testing technology evaluation, adoption, and diffusion decisions;
4. systematically reuse software testing technology change management process specifications/implementations (that is, expedite software testing technology evaluation and adoption decisions); and/or
5. maintain (correct, adapt, and perfect) software testing technology evaluation and adoption process specifications/implementations.

Above all, this research aims to develop a framework to help individuals, project teams, and organisations plan, enact, attest, expedite, and maintain (correct, adapt, perfect) software testing technology evaluations and adoptions. For the most part, the framework proposed by this research is sustained by (or necessitates) an effectual software testing technology evaluation and transfer diagramming notation design that assimilates multiple levels of abstraction and connectives to narrative explanations.

According to Anjard [1], a process map is a visual aid for picturing work processes which show how inputs, outputs, and tasks are linked. Solimon [2] regards process maps as the most important and fundamental elements of business process re-engineering. In other words, Solimon argues that process maps are the most effective means of realising process change.

Peppard et al [3] argue that there are two advantages of process mapping: usability, and greater process understanding. Effectually, Peppard et al contend that process maps are able to give a clearer explanation of a process than verbose process script. Moreover, Peppard et al argue that process maps promote a greater understanding of a process through the teaching effects of process map construction. However, Peppard et al also argue that process maps can, in some cases, prove to be too distracting to organisations. Furthermore, Peppard

et al argue that process maps can also lose relevance if the map construction process is not well understood and controlled.

Although there already exists many formal and ad hoc methods for producing process maps, this thesis propounds that Unified Modelling Language (UML) activity diagrams [4] are the most appropriate basis for constructing highly germane process maps.

While Structured Systems Analysis and Design Methodology (SSADM) [14] could be used, in part, to model software testing technology change management processes, it is best suited to analysis and design stages of systems development. The IDEF mapping standard [13] is a method designed to model the decisions, actions, and activities of an organisation or system. IDEF was derived from a well-established graphical language, the Structured Analysis and Design Technique (SADT). Basically, IDEF uses diagrams based on simple box and arrow graphics. English text labels are used to describe boxes and arrows whereas glossaries and text are used to define the precise meanings of diagram elements. Essentially, the description of the activities of a system or process can be easily refined into greater and greater detail until the model is as descriptive as necessary for the decision-making task at hand. While IDEF encompasses the means to construct software testing technology change management models that have a top-down representation and interpretation, it cannot express process execution sequence, iteration runs, selection of paths, parallel execution or conditional process flow.

The IDEF3 [15] Process Description Capture Method has been developed to overcome these weaknesses, however the IDEF3 means of collecting and documenting processes are too complex to easily model software testing technology change management processes. The ASME (American Society of Mechanical Engineers) Mapping Standard [3] is a mapping standard that is suited to detailed level mapping however it relies on the difficult evaluation of whether or not a step is value adding.

Although IDEF, IDEF3, or ASME Mapping Standard could be used for constructing software testing technology change management maps, these techniques are too intricate to be easily applied to the modelling of software testing technology change management processes. Activity diagrams are generic flow diagrams that can be easily extended to provide additional expressiveness. Moreover, because activity diagram notation is simple and well-understood, it can be easily used, controlled, and maintained. To recapitulate, the aim of this research is to construct a simple software testing technology change management process map architecture that can easily be used to guide and manage software testing technology evaluations and adoptions.

Chapter 4

Accelerating Software Testing Technology Adoption

4.1 Three-layer Process Map Architecture

Essentially, any type of software testing technology change management requires goals (or policies) that need to be satisfied through the enactment of activities or processes (mechanisms). Essentially, this research maintains that modelling of software testing technology change management policies and mechanisms is best managed by taking a layered-approach to process mapping. While policy and mechanism separation provides opportunity for a two-layer process map architecture, it is important that software testing technology change management process maps incorporate connectives to work products. Work product connectives enable software testing technology change management groups to recount the activities and actions that led to a particular software testing technology evaluation or adoption.

The three-layer process map architecture proposed in this thesis (see Figure 4.1) is composed of Unified Modelling Language (UML) [4] activity diagram notation and facilitates the administration of software testing technology change management policies, mechanisms, and work products.

4.1.1 Activity Diagram Notation

Essentially, activity diagrams consist of activities, states, and transitions between activities and states. According to Bennett, Skelton, and Lunn [4], all aspects of activity diagram notation need not be used at any one time. Furthermore, they contend that in the early stages of analysis, business workflows may not need the full notation for activity description and might better be described in simple text in a separate document. Activity diagram notation is very rich and not all aspects of the notation need to be used in constructing software testing technology change management maps.

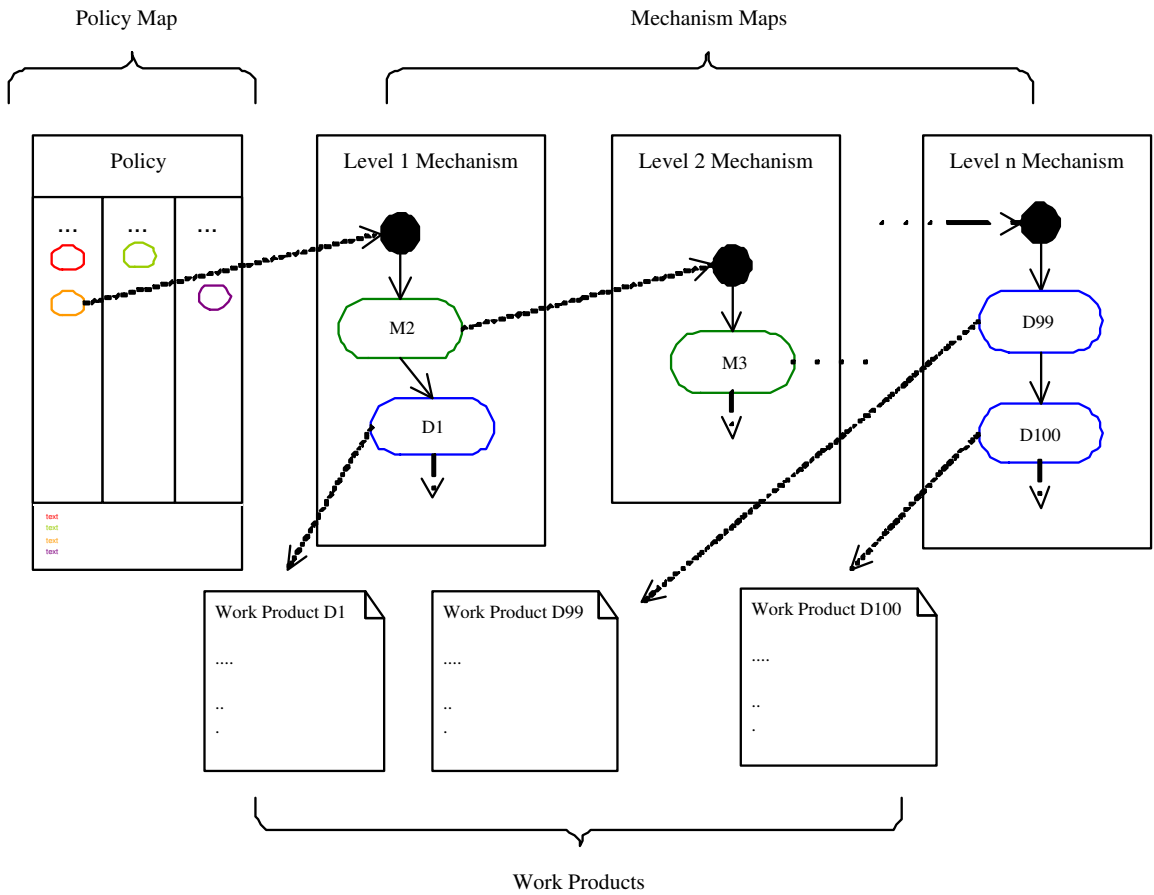


Figure 4.1: Three-layer Process Map Architecture

Essentially, the three-layer process map architecture proposed in this thesis uses:

- **activities** (see Figure 4.2) to represent units of work that need to be carried out;

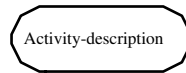


Figure 4.2: Activity

- **start and end states** (see Figures 4.3 and 4.4) to represent process entry and exit points;
- **transitions** (see Figure 4.5) to represent movement from one activity to another (or the movement between a start or end state and an activity);



Figure 4.3: Start State



Figure 4.4: End State

- **decision points** (see Figure 4.6) to represent a point in a mechanism where the exit transition from an activity may branch in alternative directions depending on a condition;
- **swimlanes** (see Figure 4.7) to indicate the context of policies (top-level activities);
- **forks and joins** (see Figures 4.8 and 4.9) to allow activities to be run in parallel; and
- **nested activity diagrams** to manage mechanism complexity.

4.1.2 Topmost-layer: Policy Management

Raising a swimlane for each management goal facilitates modelling software testing technology change management policies. Essentially, swimlanes are apposite for delineating the context of software testing technology change management activities.

As established by the Capability Maturity Model (CMM) [5], there are, by and large, five types of software testing technology change management activity:

1. **Commitment to Perform (Commitment)**: activities pertaining to establishing commitment to perform software testing technology evaluations and adoptions;
2. **Ability to Perform (Ability)**: activities pertaining to establishing ability to perform software testing technology evaluations and adoptions;
3. **Activities Performed (Activity)**: activities pertaining to performing software testing technology evaluations and adoptions;
4. **Measurement and Analysis (Measurement)**: activities pertaining to measuring and analysing software testing technology evaluations and adoptions; and



Figure 4.5: Transition



Figure 4.6: Decision Point

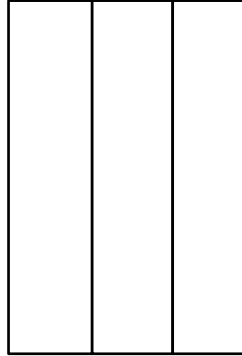


Figure 4.7: Swimlanes

5. Verifying Implementation (Verification): activities pertaining to verifying software testing technology evaluations and adoptions.

Although any orthogonal activity classification scheme can be used for policy modelling, this thesis agrees with the activity classification schemes propounded by the CMM and the Capability Maturity Model Integration (CMMI) [17]. While the CMMI proposes a similar activity classification scheme to the CMM, the CMMI mandates only four activity classifications. According to the CMMI, software testing technology change management activities can be classified as:

1. Commitment to Perform (CO): activities pertaining to establishing commitment to perform software testing technology evaluations and adoptions;
2. Ability to Perform (AB): activities pertaining to establishing ability to perform software testing technology evaluations and adoptions;
3. Directing Implementation (DI): activities pertaining to directing software testing technology evaluations and adoptions; or

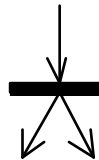


Figure 4.8: Fork

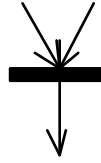


Figure 4.9: Join

4. Verifying Implementation (VE): activities pertaining to verifying software testing technology evaluations and adoptions.

While swimlanes can be used to delineate activity context, colour coding can be used to demarcate activity types. For example, CMMI Commitment to Perform activities could be tinted red whereas CMMI Ability to Perform activities could be coloured blue. The advantage of using colour to demarcate activities is that it helps technology change management groups easily identify and compare particular activities or families of activities.

While the CMM and CMMI mandate sub-practices for some top-level activities, these sub-practices are best treated/modelled as topmost-level mechanism diagrams. This not only provides a good starting point for implementing mechanisms, but also ensures that policy maps are one-tier (or one layer deep). Essentially, sub-practices should be modelled at one layer below the parent practice - this also applies to sub-practices of a policy (top-level activity).

4.1.3 Intermediate-layers: Mechanism Management

While topmost-level diagrams are used to model policies (or top-level activities), intermediate-level diagrams are useful for explicating policy mechanics (or implementations). Although intermediate-level diagrams are apposite for providing varying degrees of abstraction, connectives to work products and/or narrative explanations are needed to stop up diagram nesting. Moreover, it is easy to see that mechanism model depths are often not uniform simply because some policies necessitate more complex mechanics than others. Furthermore, some mechanism elements are better expressed using narrative explanations in place of abstract diagram notation.

Nested activity diagrams are a suitable means for modelling policy mechanisms at multiple levels of abstraction. More importantly, taking a layered or incremental approach to mechanism modelling allows corrections, adaptations, and perfections to be performed in more compendious and expeditious ways. In other words, nested diagrams provide a way of constructing reusable and maintainable software testing technology change management process (sub-) maps that are modular and robust in design.

Another important benefit of modular mechanism maps is that they provide an easy way to attest software testing technology change management decisions and work products.

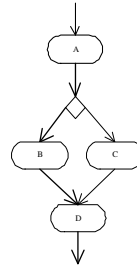


Figure 4.10: Course-plotting

Through the use of course-plotting, software testing technology change management groups are able to chronicle and recount the activities and actions that produced a particular work product. Moreover, because mechanism models are often nested (or multi-level), there exists an opportunity to communicate or analyse course-plots at multiple levels of abstraction. Clearly, course-plots are useful for directing causal analyses and resolutions, or more generally, process improvements.

Lastly, natural language constraints are apt for ensuring that mechanism interpretation is circumspective. Where appropriate, constrictions need to be formulated, expressed and attached to activity transitions. Clearly, pre- and post-conditions provide a means for persuading careful interpretation of activity requirements and expectations.

Obviously, to chronicle the activities and actions that produced a particular work product (that is, a particular technology evaluation or adoption), a copy of the mechanism needs to be made available for course-plotting. While any type of tag can be used to denote activities on a mechanism path, this thesis proposes that activity transition lines be thickened to indicate course-plots (see Figure 4.10).

4.1.4 Bottommost-layer: Work Product Management

At some model depth, it becomes obvious that documentation connectives are more suited to explicating activity or work product detail. By connecting model elements to work products or narrative explanations, a software testing technology change management group is more easily able to trace activities to work products. In essence, documentation connectives stop up diagram nesting.

However, linking up mechanisms to work products is not easy without a system or form of abbreviated or formulaic reference. Constructing a concise reference requires formal rules to ensure that connective interpretation is safe. Moreover, it is important to ensure that connectives are complete, consistent and correct. A notation or colour code is needed to let model readers know if an activity is closing (that is, in need of no further explanation) or connected to a document or nested diagram. This thesis proposes a three-code colour coding system for marking activity connectives:

1. closing activities are tinted red;
2. document-connected activities are tinted blue; whereas
3. activities connected to a nested diagram are tinted green.

Essentially, any three-code colour coding system can be used to characterise connectives.

While colour codes are able to explicate connective types, descriptive activity names are used to reference relatives (nested activity diagrams or documents). It is important to note that connectives to documents need to be constructed on mechanism copies (where course-plotting is undertaken) and not on mechanism masters - this is simply to ensure that mechanisms remain reusable.

4.2 Constructing Software Testing Technology Change Management Process Maps

Activity diagram swimlanes are used to communicate and delineate the context of software testing technology change management policies (or top-level activities) whereas nested activity diagrams are drawn on to demarcate policies from mechanisms (or sub-practices). In other words, nested activity diagrams are apposite for separating what activities are to be performed from how those activities are to be performed, or in other words, separating intent from realisation. Moreover, because activity diagrams are generic flow diagrams, they can be easily extended to provide additional expressiveness. While swimlanes delineate activity context, colour coding is used to demarcate activity and connective types. Forks and joins enable activities to be enacted in parallel whereas decision nodes bump up the number of alternative courses of action. Connectives to narrative explanations can stop up diagram nesting.

While software testing technology change management policy and mechanism maps coordinate, policy maps are able to stand alone. On the other hand, mechanism maps should not exist without a guiding policy.

Essentially, constructing a software testing technology change management process map requires that a policy map be created first of all. To this end, a swimlane needs to be raised for each software testing technology change management goal, and each policy (top-level activity) needs to be classified and colour coded. Each policy in the policy map can then be connected to a nested diagram or narrative explanation (document) using a formulaic or abbreviated reference. To recapitulate, intermediate-level diagrams are useful for explicating policy mechanics at multiple levels of abstraction. Again, at some model depth, it becomes obvious that documentation connectives are more suited to explicating activity or work product detail. Lastly, subjective review is needed to identify corrections, adaptations, and perfections to the software testing technology change management map. The refinement of

software testing technology change management process maps is continuous and iterative throughout the construction process.

4.3 Planning Software Testing Technology Evaluations and Adoptions

By taking a structured, model-based approach to software testing technology change management, software testing technology change management groups are more easily able to plan technology evaluation and adoption activities. Clearly, modular software testing technology change management process maps enable change management groups to plan change management activities at multiple levels of abstraction. In effect, software testing technology change management process maps can be used as a tool for designing and visualising change management activities before verbose process scripts are written. Moreover, process map designs can enable change management groups to ensure that technology evaluation and adoption processes are well-defined and validated (through subjective peer-review) before they are institutionalised.

4.4 Managing Software Testing Technology Change

Once a software testing technology change management process map has been produced and accepted, change management group members can use the map to guide their technology change management actions. Clearly, script-based software testing technology change management removes doubtfulness or uncertainty as regards interpretation whereas model-based software testing technology change management facilitates accelerated interpretation at the expense of some uncertainty. By using constraints on mechanism maps, map readers are able to easily determine the pre- and post-conditions (or requirements and expectations) for each activity that needs to be enacted. In other words, constraints can help ensure that mechanism map reading is circumspective. In essence, pre- and post-conditions on activity transitions helps change management groups verify and validate their acts. Clearly, constraints can alert change management groups to issues that necessitate causal analysis and resolution.

4.5 Attesting Software Testing Technology Evaluations and Adoptions

As software testing technology change management groups follow the process maps, a chronicle of their actions and work products can be created. Again, modular mechanism maps provide an easy way to attest software testing technology change management decisions and work products. Through the use of course-plotting, software testing technology change

management groups are able to chronicle and recount the activities and actions that produced a particular work product. Moreover, because mechanism models are often nested (or multi-level), there exists an opportunity to communicate or analyse course-plots at multiple levels of abstraction. Clearly, course-plots are also useful for directing causal analyses and resolutions, or more generally, process improvements.

4.6 Communicating Software Testing Technology Change

Model-based software testing technology change management enables software testing technology change management groups to more easily communicate technology evaluation and adoption policies, mechanisms, and work products. In fact, because mechanism maps are often composed of multiple levels, there exists the opportunity for change management groups to communicate mechanisms at different levels of abstraction. As mentioned above, course-plots can also be used to communicate activities and actions that lead to a particular evaluation or adoption.

4.7 Expediting Software Testing Technology Change Management

By taking a structured, model-based approach to software testing technology change management, software testing technology change management groups can more easily reuse technology evaluation and adoption policies and mechanisms. While it is difficult to propose reasons for reusing a technology change management policy, a software testing technology change management group can easily reuse a mechanism provided that the pre- and post-conditions of the mechanism satisfy the policy goal (or invariant) or parent mechanism goal. Essentially, Design by Contract [16] constraint notions can help ensure that mechanism reuse is justified.

4.8 Maintaining Software Testing Technology Change Management Process Maps

As process maps are traversed (that is, as constraints are interpreted and course-plots are recorded), software testing technology change management groups are alerted to imperative and anticipatory corrections, adaptations, and perfections to technology change management processes (or more specifically, mechanism maps). After causal analysis, resolutions need to be incorporated into the software testing technology change management process map. To this end, policies, mechanisms, connectives and/or constraints can be replaced, removed, or amended. Again, a modular process map structure makes it easy to replace process implementations.

Chapter 5

Case Studies

To expound the three-layer software testing technology change management map presented in Chapter 4, four case studies were undertaken: three organisational software process improvement frameworks (CMM [5], CMMI [17], and ISO 9001 [5]) and one team software process improvement framework (TSPi [18]).

The Capability Maturity Model (CMM) is an organisational software process improvement framework whereas the Capability Maturity Model Integration (CMMI) provides guidance for improving an organisation's processes and its ability to manage the development, acquisition, and maintenance of products and services. Essentially, the CMM Integration was conceived to sort out the problem of using multiple Capability Maturity Models (CMMs). ISO 9001 is a requirement standard for assessing an organisation's ability to meet customer and applicable regulatory requirements. In contrast, the Introductory Team Software Process (TSPi) is a team software process improvement framework that assimilates the concepts of CMM Level 5 (Optimising Level) key process areas (defect prevention, technology change management, process change management). The TSPi was designed to provide both a strategy and a set of operational procedures for using disciplined software process methods at the individual and team levels.

5.1 Constructing a Policy Map

To recapitulate, constructing a policy map (or mechanism map catalogue) requires defined management goals and activity types.

Although there are many possible orderings of activities, it is clear that software testing technology change management processes need to be implemented before they are institutionalised. In other words, activities in a software testing technology change management process need to be defined before the commitments and abilities needed to perform those activities can be realised. Moreover, practices for measuring and analysing and verifying

the implementation of software testing technology change management activities cannot be defined until the software testing technology change management process has been implemented.

Once the software testing technology change management process is implemented and institutionalised, map-readers can follow the mechanisms pertaining to the process implementation to chronicle and recount the activities that lead to a particular software testing technology evaluation or adoption. Because software process improvement frameworks are not limited to any type of software testing technology evaluation, mechanism maps are able to implement any type of software testing technology evaluation technique provided that the pre- and post-conditions of the implementation satisfy the goal of the parent mechanism or policy.

While software testing technology evaluations and adoptions are the principal activities of a software testing technology change management process, it is also important that the software testing technology change management process is measurable and controllable so that it is lasting.

5.2 Constructing a Mechanism Map

For any policy in the policy map, a mechanism map can be constructed. By using activity diagram activities, transitions, decision points, and forks and joins, a policy implementation can be modelled. Where appropriate, an activity in the mechanism map can be marked as a connective to a nested mechanism map or document. Alternatively, an activity can be marked as closing (that is, in need of no further explanation). To recapitulate, marking an activity as a connective requires a three-code colour code system. While any three-code colour code system can be used for this purpose, this thesis advises that activities connected to a nested diagram be tainted green, activities connected to a document be tainted blue, whereas activities that are closing be tainted red. Again, colour codes help map-readers more easily determine whether further detail pertaining to an activity is available.

5.3 Modelling Organisational Software Testing Technology Change Management

5.3.1 ISO 9001 Software Testing Technology Change Management

According to Paulk et al [5], the ISO 9000 series of standards developed by the International Standards Organisation shares a common concern about quality and process management with the CMM.

Although ISO 9001 does not clearly mandate goals for software testing technology change management, two were elicited from the standard: inspection equipment is controlled; and

the software testing technology change management process is institutionalised as a defined process.

Clause 4.11 (Control of Inspection Equipment) prescribes requirements for implementing a software testing technology change management process whereas Clauses 4.1 (Management Responsibilities), 4.2 (Quality System Requirements), 4.16 (Control of Quality Records), 4.18 (Training Requirements), and 4.20 (Statistical Techniques) prescribe requirements for institutionalising a software testing technology change management process.

Clearly, Clause 4.11 (Control of Inspection Equipment) requirements satisfy Goal 1 (Inspection equipment is controlled) whereas Clauses 4.1 (Management Responsibilities), 4.2 (Quality System Requirements), 4.16 (Control of Quality Records), 4.18 (Training Requirements), and 4.20 (Statistical Techniques) satisfy Goal 2 (The process is institutionalised as a defined process).

Although it not clear how or if the ISO 9001 distinguishes activity types, it is easy to see that the procedures for establishing commitment to perform software testing technology evaluations and adoptions are prescribed in Clause 4.1 (Management Responsibilities). Moreover, it is somewhat evident that requirements for establishing ability to perform software testing technology evaluations and adoptions are prescribed in Clause 4.1 (Management Responsibilities) and 4.18 (Training Requirements). Clause 4.16 (Control of Quality Records) and 4.20 (Statistical Techniques) suggest requirements for measuring and analysing software testing technology change management activities whereas Clause 4.1 (Management Responsibilities) and 4.2 (Quality System Requirements) prescribe requirements for verifying the implementation of a software testing technology change management process. On the other hand, it is clear that Clause 4.11 (Control of Inspection Equipment) prescribes the requirements for implementing a software testing technology change management process.

Using a similar activity classification scheme as the CMM, it is easy to classify ISO 9001 institutionalisation activities as Commitment to Perform, Ability to Perform, Measurement or Analysis, or Verifying Implementation activities. In summary, Clause 4.1 prescribes Commitment to Perform activities, Clause 4.1 and 4.18 prescribe Ability to Perform activities, Clause 4.16 and 4.20 prescribe Measurement and Analysis activities, whereas Clause 4.1 and 4.2 prescribe Verifying Implementation activities. Moreover, it is easy to see that Clause 4.11 prescribes Activities Performed.

In summary, by using a similar activity classification scheme to the CMM, it is easy to map ISO 9001 software testing technology change management activities into five categories: Commitment to Perform, Ability to Perform, Activities Performed, Measurement and Analysis, and Verifying Implementation.

Determining the context of ISO 9001 software testing technology change management activities is not easy as the ISO 9001 does not clearly prescribe goals for software testing technology change management. Nevertheless, a swimlane was raised for the goal of Clause 4.11 (inspection equipment is controlled) and for the goal of institutionalising a defined software testing technology change management process.

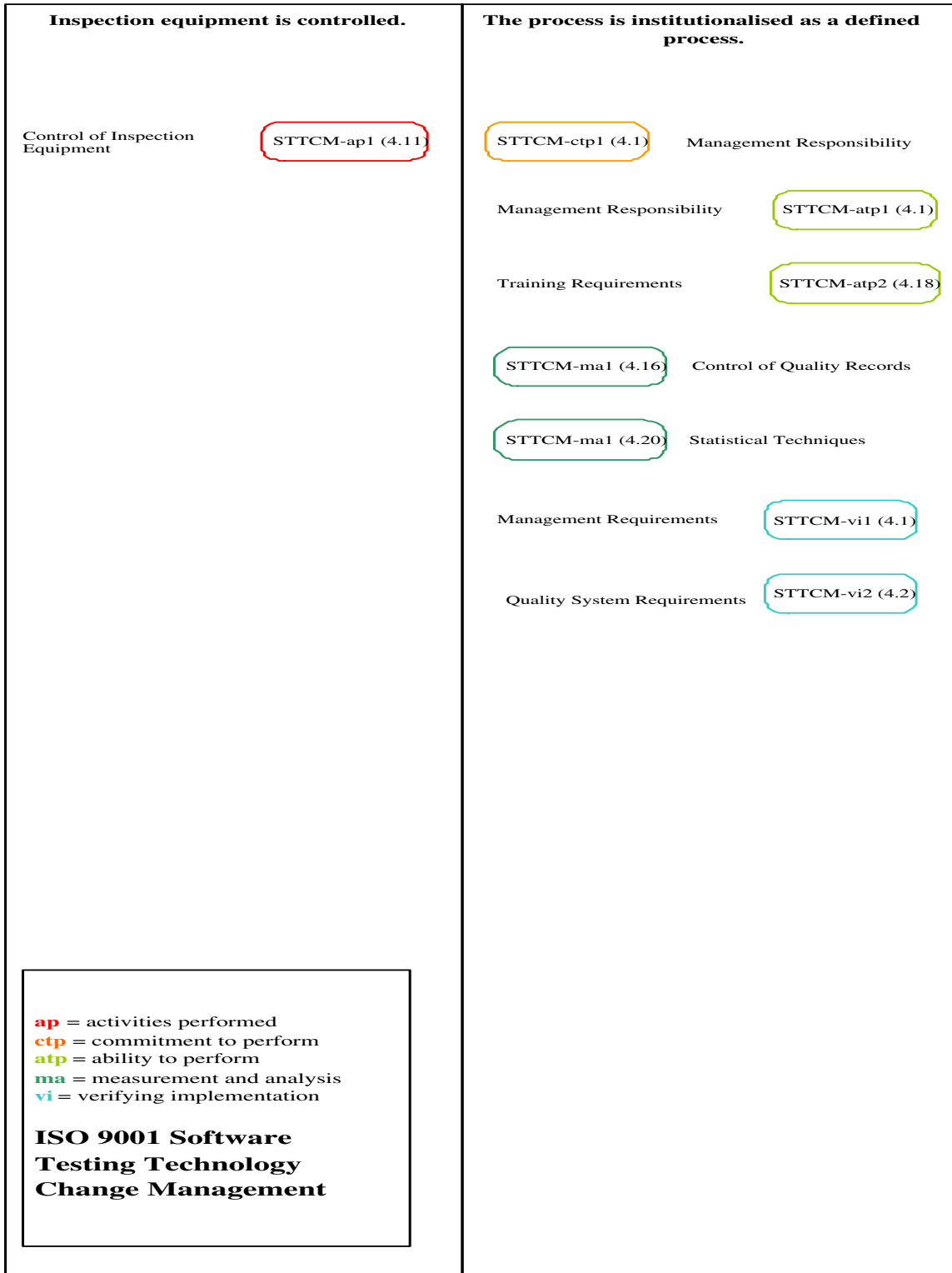


Figure 5.1: ISO 9001 Software Testing Technology Change Management Policy Map

A policy map for ISO 9001 software testing technology change management is depicted in Figure 5.1. As shown in Figure 5.1, there is a swimlane pertaining to each ISO 9001 software testing technology change management goal.

Figure 5.2 shows examples of activity orderings: an example of an ordering to enact or implement a software testing technology change management process and an example of an ordering to institutionalise software testing technology change management. Clearly, ordering of process implementation and institutionalisation activities is dependent upon project management concerns and scheduling.

In the case of ISO 9001, software testing technology evaluations and adoptions can be chronicled and recounted by following mechanism maps pertaining to Clause 4.11 (Control of Inspection Equipment). Because Clause 4.11 does not prescribe any particular way to evaluate software testing technologies, it is possible to implement any type of software testing technology evaluation technique (as described in Chapter 2) using mechanism maps. Although this thesis does not demonstrate how to construct a deep mechanism map for Clause 4.11 requirements, it does show top-level mechanisms for each of the policies in Figure 5.1 (and Figure 5.2) (see Figures 5.3 and 5.4). It is easy to see that the normative nature of the ISO 9001 standard makes it easy to construct a three-layer software testing technology change management model.

5.3.2 CMM Software Testing Technology Change Management

According to the Capability Maturity Model (CMM) [5], the object of technology change management is to identify new technologies and transition them into the organisation in an orderly manner.

The CMM mandates three goals for software testing technology change management: incorporation of technology changes is planned; new technologies are evaluated to determine their effect on quality and productivity; and appropriate new technologies are transferred into normal practice across the organisation.

Essentially, the CMM sets out policies pertaining to establishing commitment and ability to manage software testing technology change, managing software testing technology change, and measuring and verifying software testing technology change management.

Technology Change Management activities prescribes requirements for implementing a software testing technology change management process whereas commitments, abilities, measurements, and verifications prescribe requirements for institutionalising a software testing technology change management process.

The CMM mandates that effectual software testing technology change management necessitates three commitments: the organisation follows a written policy for improving its technology capability, senior management sponsors the organisation's activities for technology change management, and senior management oversees the organisation's technology change management activities.

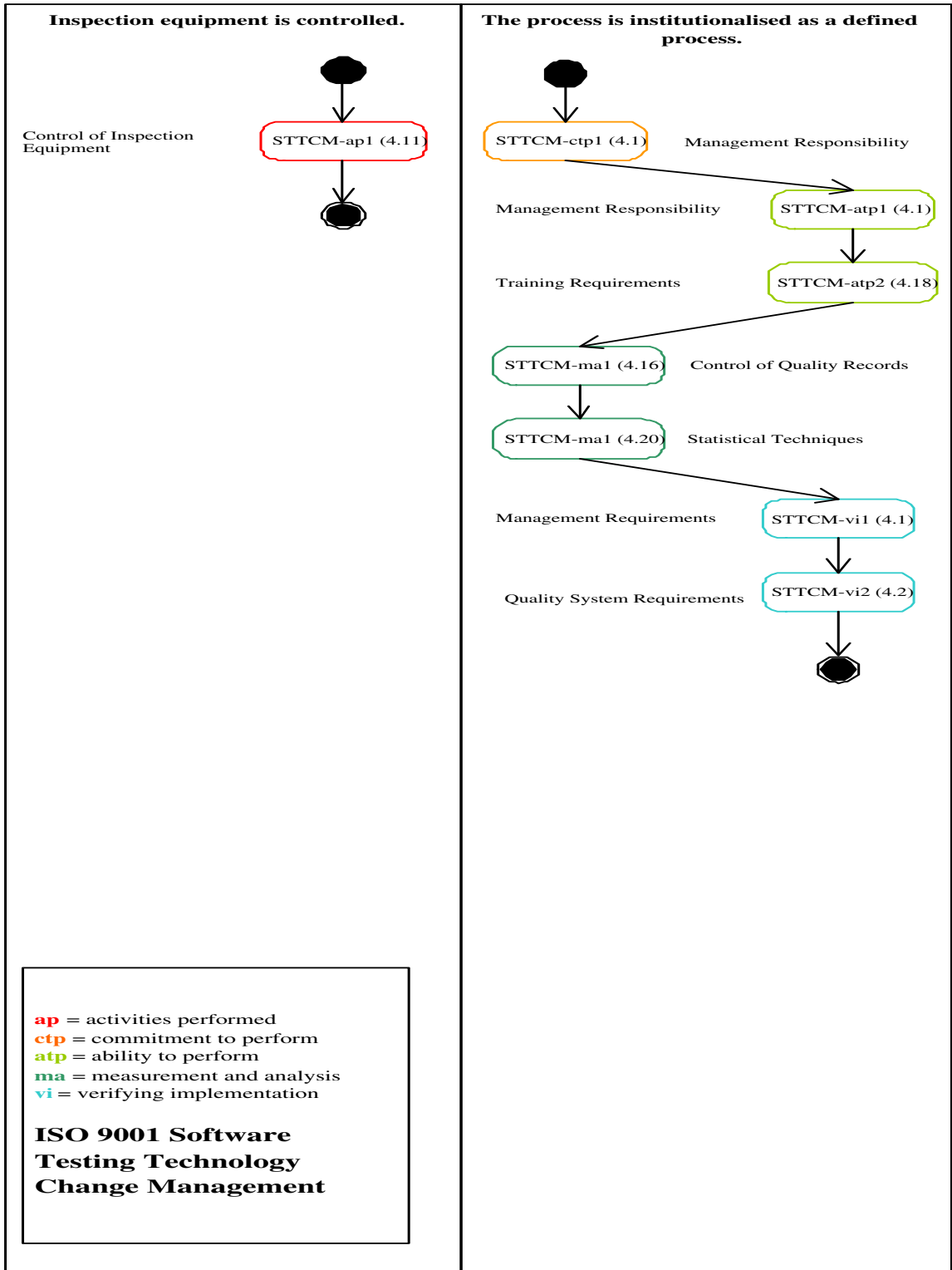


Figure 5.2: ISO 9001 Software Testing Technology Change Management Policy Map with Ordering

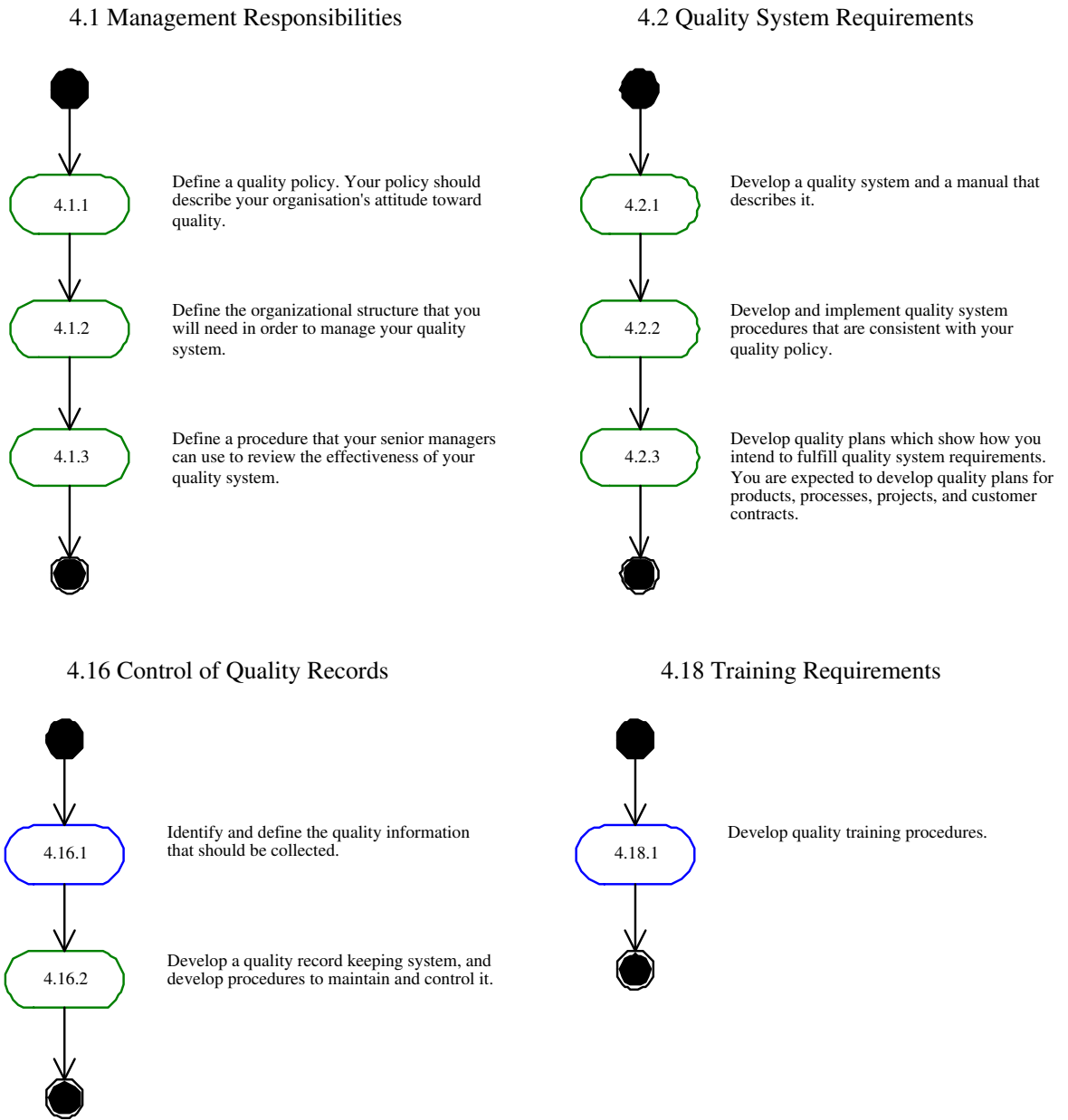


Figure 5.3: ISO 9001 Software Testing Technology Change Management Mechanism Maps

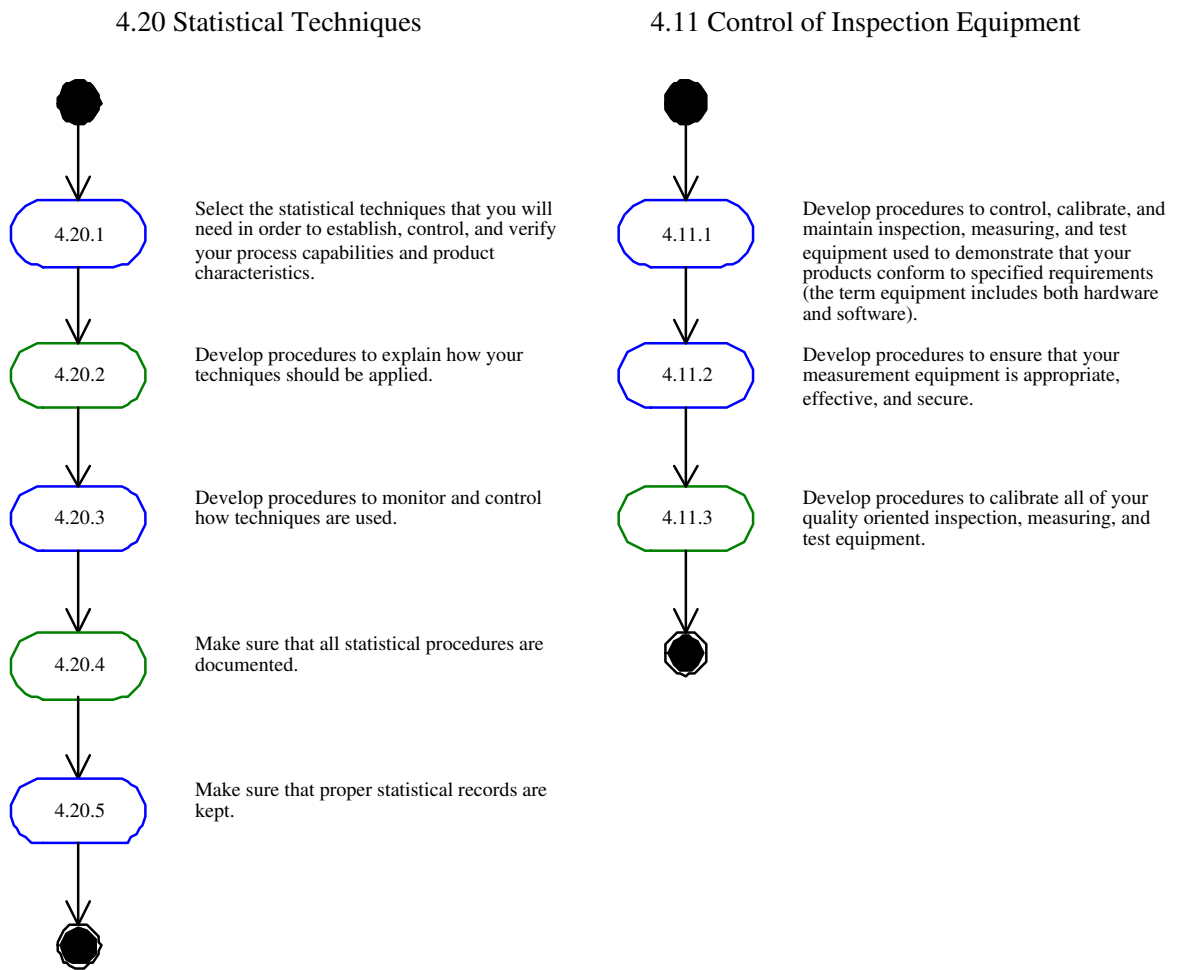


Figure 5.4: ISO 9001 Software Testing Technology Change Management Mechanism Maps

Moreover, the CMM advises that management of software testing technology change requires the satisfaction of five preconditions: a group responsible for the organisation's technology change management activities exists, adequate resources and funding are provided to establish and staff a group responsible for the organisation's technology change management activities, support exists for collecting and analysing data needed to evaluate technology changes, appropriate data on the software processes and software work products are available to support analyses performed to evaluate and select technology changes, and members of the group responsible for the organisation's technology change management activities receive training to perform these activities.

In addition, the CMM prescribes eight activities for managing software testing technology change: the organisation develops and maintains a plan for software testing technology change management, the group responsible for the organisation's software testing technology change management activities works with the software projects in identifying areas of software testing technology change, software managers and technical staff are kept informed of new software testing technologies, the group responsible for the organisation's software testing technology change management systematically analyses the organisation's standard software testing process to identify areas that need or could benefit from new software testing technology, software testing technologies are selected and acquired for the organisation and software projects according to a documented procedure, pilot efforts for improving software testing technology are conducted, where appropriate, before a new software testing technology is introduced into normal practice, appropriate new software testing technologies are incorporated into the organisation's standard software testing process according to a documented procedure, and appropriate new software testing technologies are incorporated into the projects' defined software testing processes according to a documented procedure.

The CMM also recommends that measurements be made and used to determine the status of the organisation's activities for technology change management. Moreover, the CMM advocates two practices for verifying software testing technology change management: the organisation's activities for software testing technology change management are reviewed with senior management on a periodic basis, and the software quality assurance group reviews and/or audits the activities and work products for software testing technology change management and reports the results.

Although this thesis does not present a total software testing technology change management map for the CMM, it does show that policy modelling makes easy implementing the prescriptive practices mandated by the CMM.

A policy map for CMM software testing technology change management is depicted in Figure 5.5. As shown in Figure 5.5, Ability to Perform activities are tinted red, Commitment to Perform activities are tinted orange, Ability to Perform activities are tinted green, Measurement and Analysis activities are tinted light blue, and Verifying Implementation activities are tinted aqua. Furthermore, as shown in Figure 5.5, there is a swimlane pertaining to each CMM software testing technology change management goal. Figure 5.6 shows examples of activity orderings: an example of an ordering to enact or implement a software

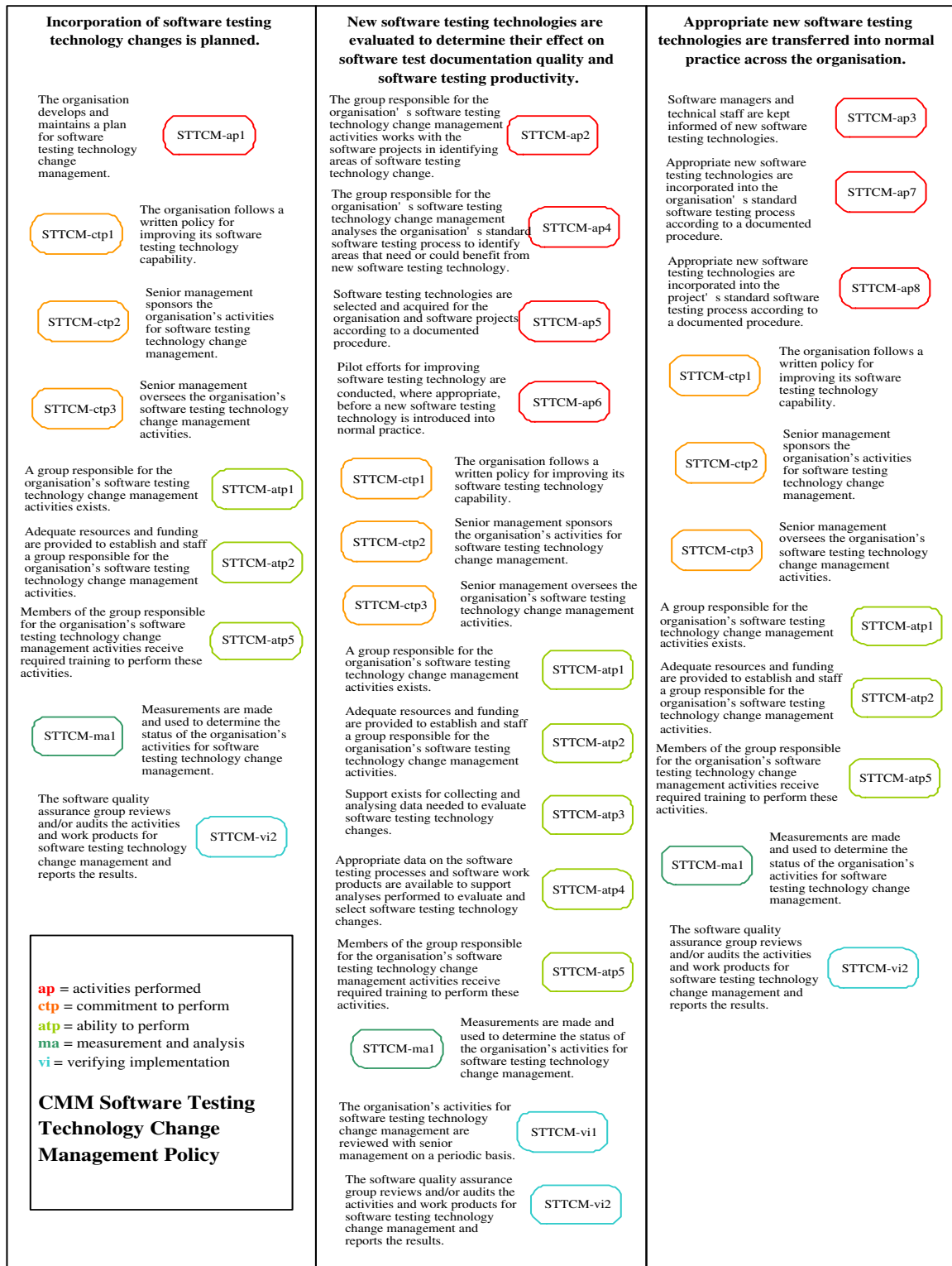


Figure 5.5: Capability Maturity Model (CMM) Software Testing Technology Change Management Policy Map

testing technology change management process and an example of an ordering to institutionalise software testing technology change management. Again, the ordering of process implementation and institutionalisation activities is dependent upon project management concerns and scheduling. Because of this, it is impossible to present all possible schedules.

Again, it has proven unrealistic to attempt to produce mechanisms for each policy mandated by the CMM in this research. Nevertheless, this research has shown that it is possible to produce a policy map for the CMM due to the prescriptive nature of the framework. Moreover, there is no reason to suggest that mechanism maps (and encompassing work product connectives) are not apposite or possible for representing CMM policy implementations.

5.3.3 CMMI Software Testing Technology Change Management

According to the Capability Maturity Model Integration (CMMI) [17], the purpose of Organisational Innovation and Deployment is to select and deploy incremental and innovative improvements that measurably improve the organisation's processes or technologies.

The CMMI mandates three goals for Organisational Innovation and Deployment: process and technology improvements that contribute to meeting quality and process-performance objectives are selected; measurable improvements to the organisation's processes and technologies are continually and systematically deployed; and the process is institutionalised as a defined process.

Organisational Innovation and Deployment specific practices prescribes requirements for implementing a software testing technology change management process whereas generic practices (commitments, abilities, implementation directives, and verifications) prescribe requirements for institutionalising a software testing technology change management process.

The CMMI sets out policies pertaining to selecting improvements, deploying improvements, and institutionalising a defined software testing technology change management process. Essentially, the CMMI mandates that to select improvements, a software testing technology change management group needs to collect and analyse improvement proposals, identify and analyse innovations, pilot improvements, and select improvements for deployment. The CMMI argues that deploying improvements necessitates mechanisms to plan the deployment, manage the deployment, and measure the effects of improvements. To institutionalise a defined software testing technology change management process, the CMMI suggests ten general practices: establish an organisational policy, establish a defined process, plan the process, provide resources, assign responsibility, train people, manage configurations, identify and involve relevant stakeholders, monitor and control the process, collect improvement information, objectively evaluate adherence, and review status with higher management.

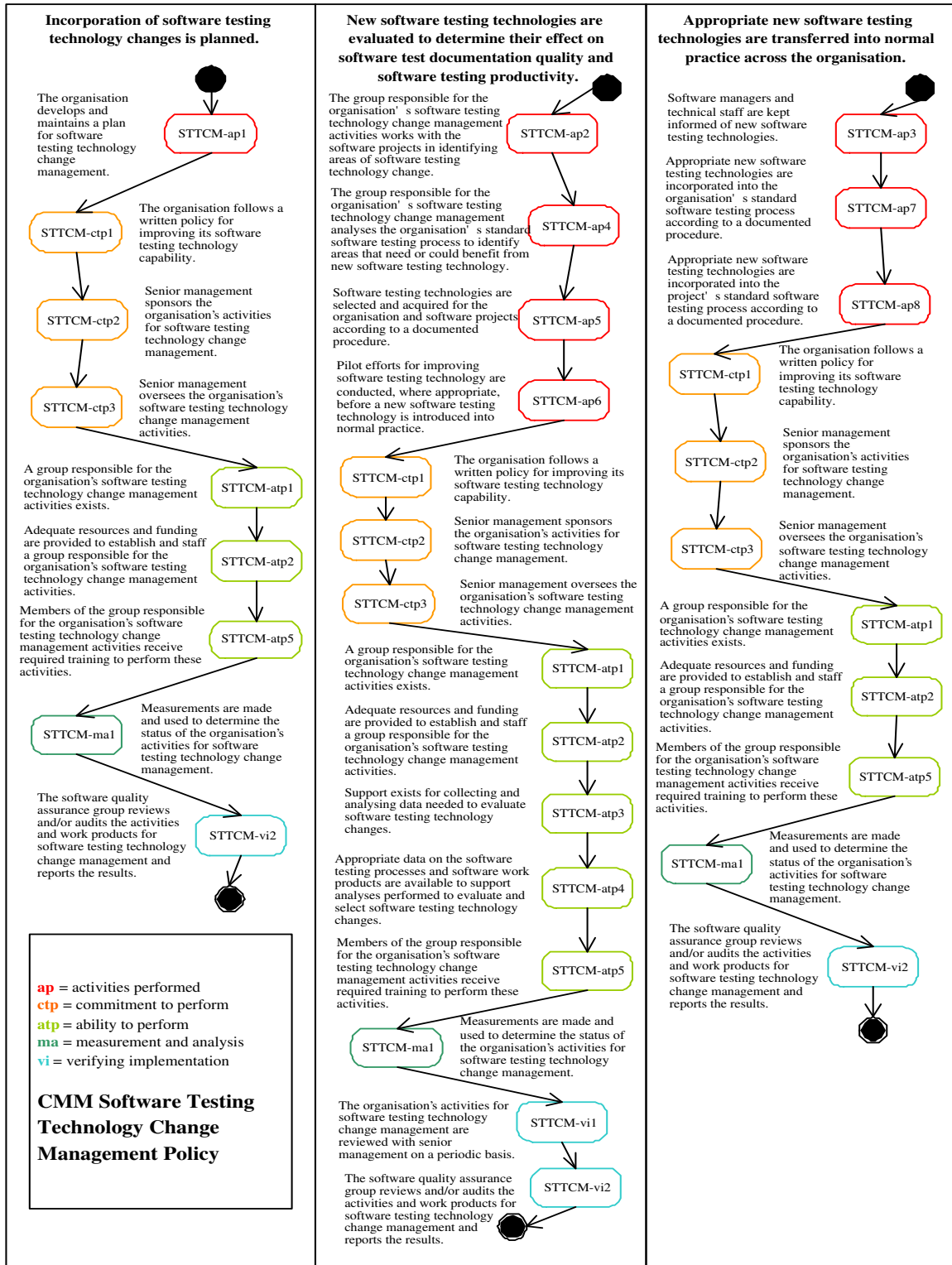


Figure 5.6: Capability Maturity Model (CMM) Software Testing Technology Change Management Policy Map with Ordering

Basically, construction of a policy map for the Organisational Innovation and Deployment key process area requires three swimlanes to be raised: one swimlane for the Select Improvements goal, another swimlane for the Deploy Improvements goal, and another swimlane for the Institutionalise a Defined Process goal.

As shown in Figure 5.7, Activities Performed are tinted purple, Commitment to Perform activities are tinted red, Ability to Perform activities are tinted green, Directing Implementation activities are tinted orange, and Verifying Implementation activities are tinted blue.

Figure 5.8 shows examples of activity orderings: an example of an ordering to enact or implement a software testing technology change management process and an example of an ordering to institutionalise software testing technology change management. Again, the ordering of process implementation and institutionalisation activities is dependent upon project management concerns and scheduling. Because of this, it is impossible to present all possible schedules.

Like the CMM, the CMMI mandates subpractices for each of the top-level activities (practices) of the Select Improvements and Deploy Improvements goals. Each of the practices (or policies) in the Select Improvements and Deploy Improvements swimlanes connect to a mechanism map (see Figures 5.9, 5.10, 5.11, and 5.12). Each top-level mechanism map for each of the policies can use colour coding to let map readers know if a mechanism is closing, or connected to a more detailed document or mechanism. Although this thesis does not present a total software testing technology change management map for the CMMI, it is easy to see that policy modelling also presents a good starting point for implementing CMMI policies.

Like with the CMM, it is not viable to attempt to produce deep mechanisms for each policy mandated by the CMMI in this research. Nevertheless, this research has also shown that it is possible to produce a policy map for the CMMI due to the prescriptive nature of the framework (see Figure 5.7). Moreover, there is no reason to suggest that mechanism maps (and encompassing work product connectives) are not apposite for representing CMMI policy implementations. In fact, Figures 5.9, 5.10, 5.11, and 5.12 show top-level mechanism maps for each of the policies contained in Figure 5.7 with example connectives to nested mechanisms or documents.

5.4 Modelling Team and Individual Software Testing Technology Change Management

5.4.1 TSPi Software Testing Technology Change Management

The Introductory Team Software Process (TSPi) mandates one goal for technology change management: the team has suitable tools and methods to support its work. This sole goal for technology change management is a goal of the TSPi Support Manager Role.

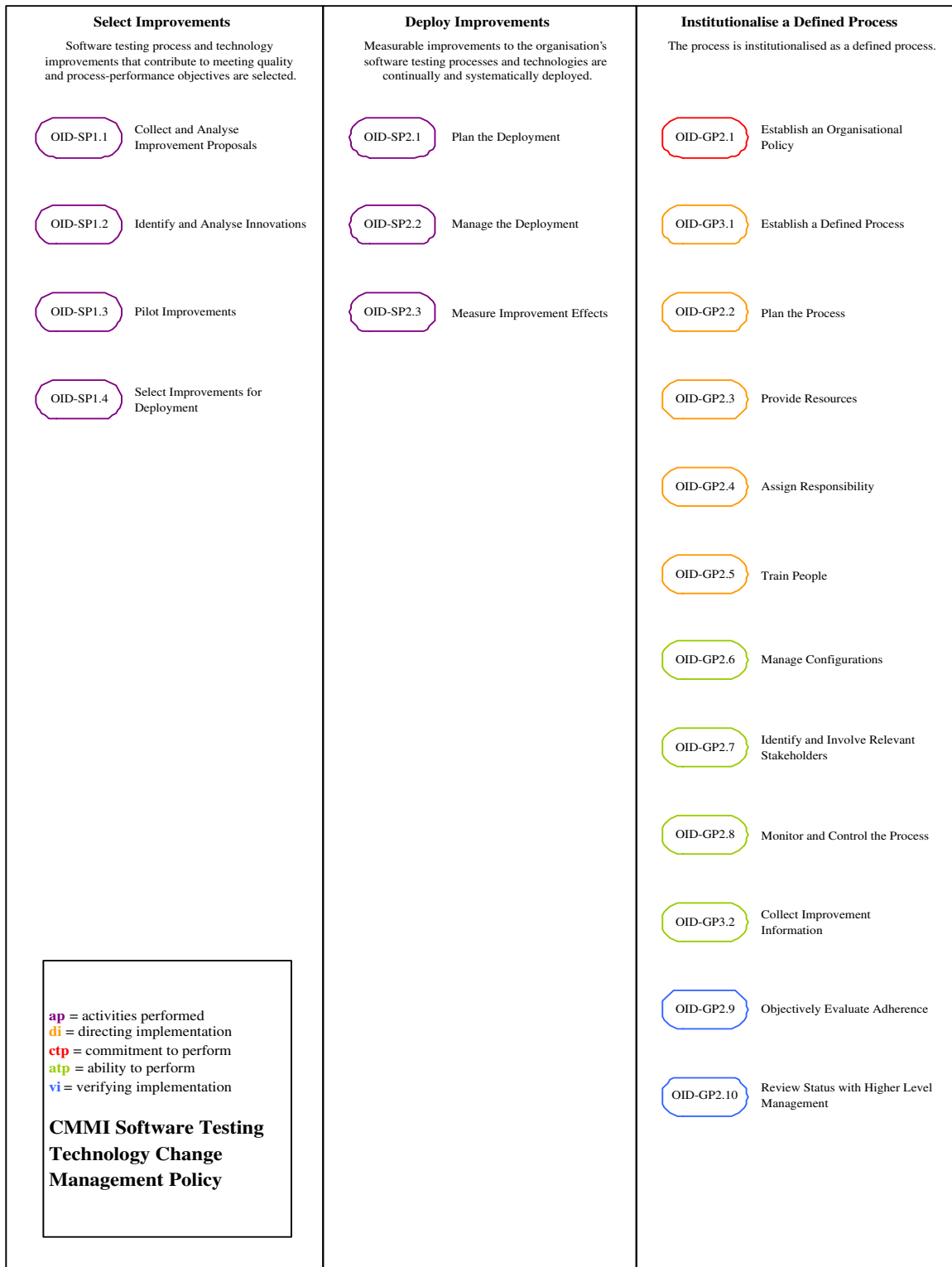


Figure 5.7: Capability Maturity Model Integration (CMMI) Software Testing Technology Change Management Policy Map

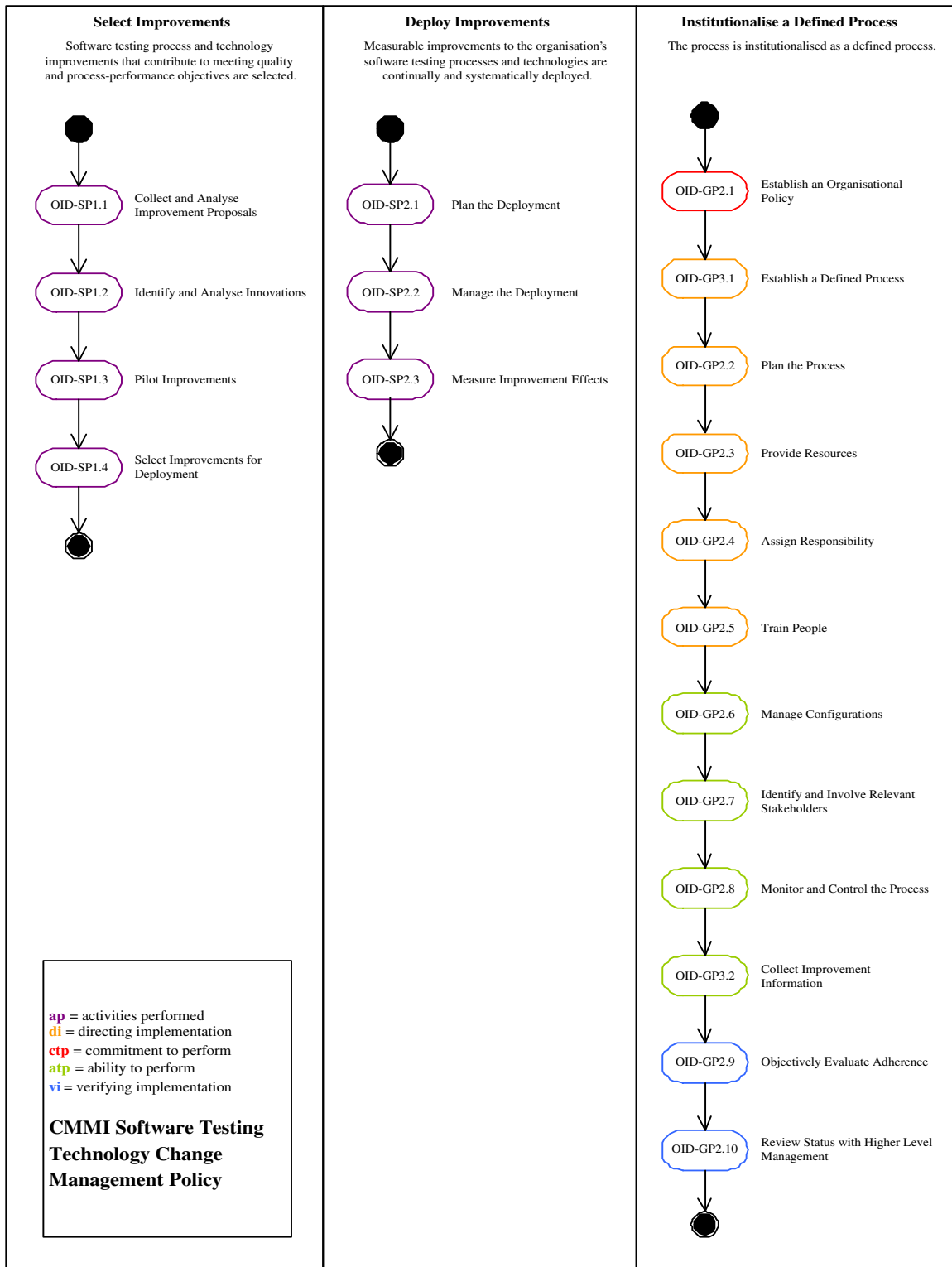
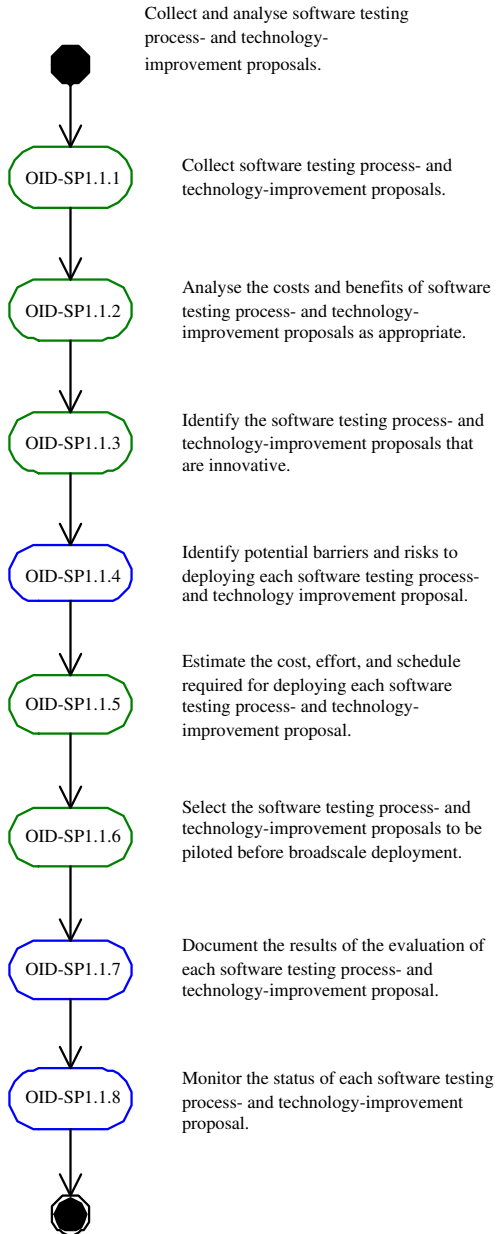


Figure 5.8: Capability Maturity Model Integration (CMMI) Software Testing Technology Change Management Policy Map with Ordering

SP 1.1 Collect and Analyse Improvement Proposals



SP 1.2 Identify and Analyse Innovations

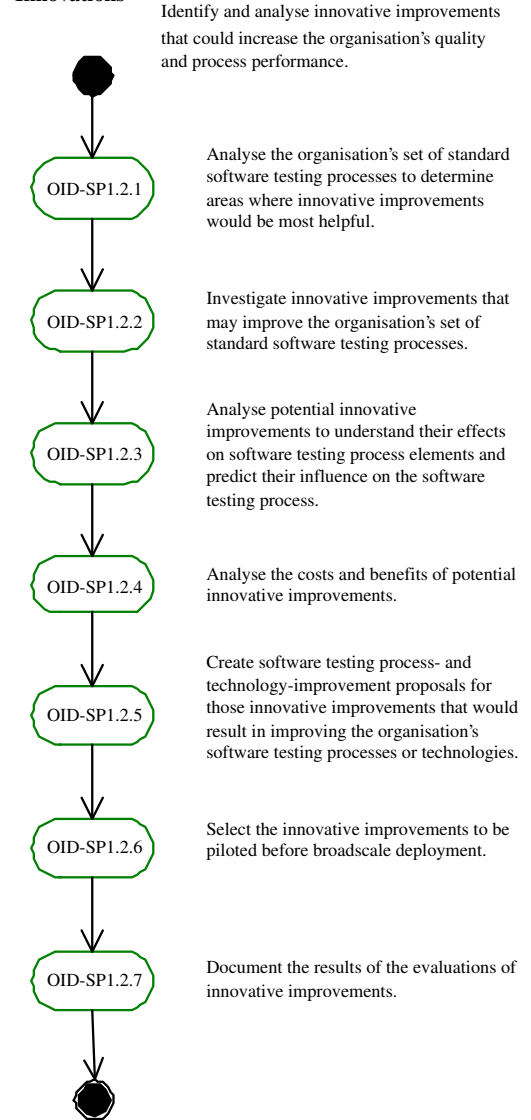
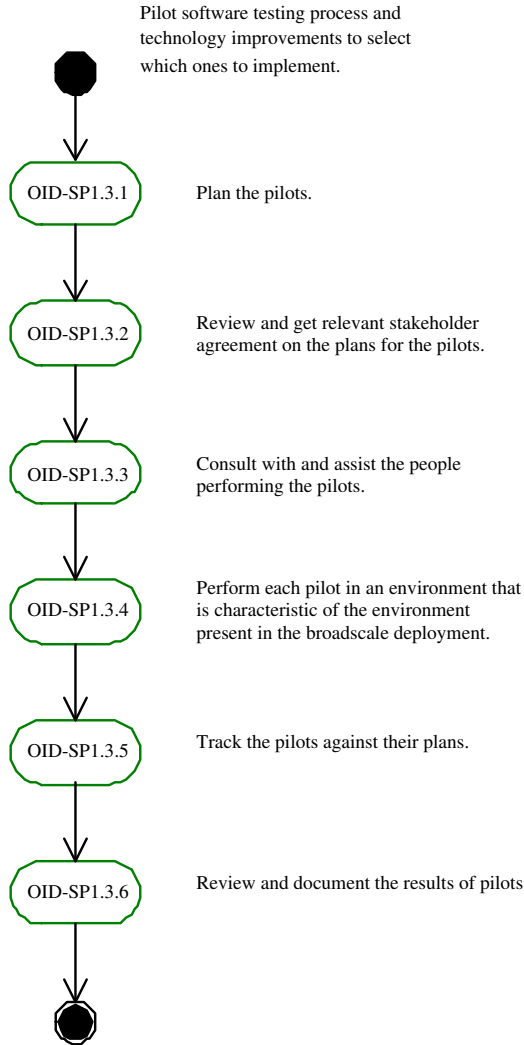


Figure 5.9: Capability Maturity Model Integration (CMMI) Software Testing Technology Change Management Mechanism Maps

SP 1.3 Pilot Improvements



SP 1.4 Select Improvements for Deployment

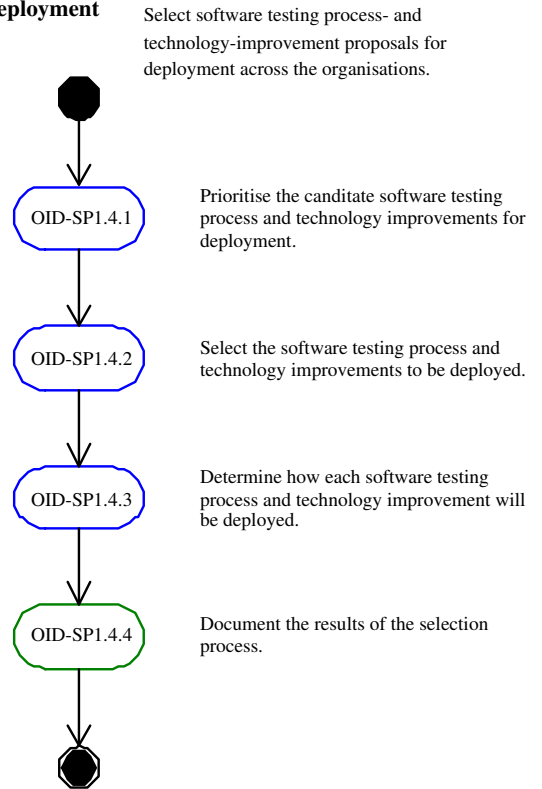
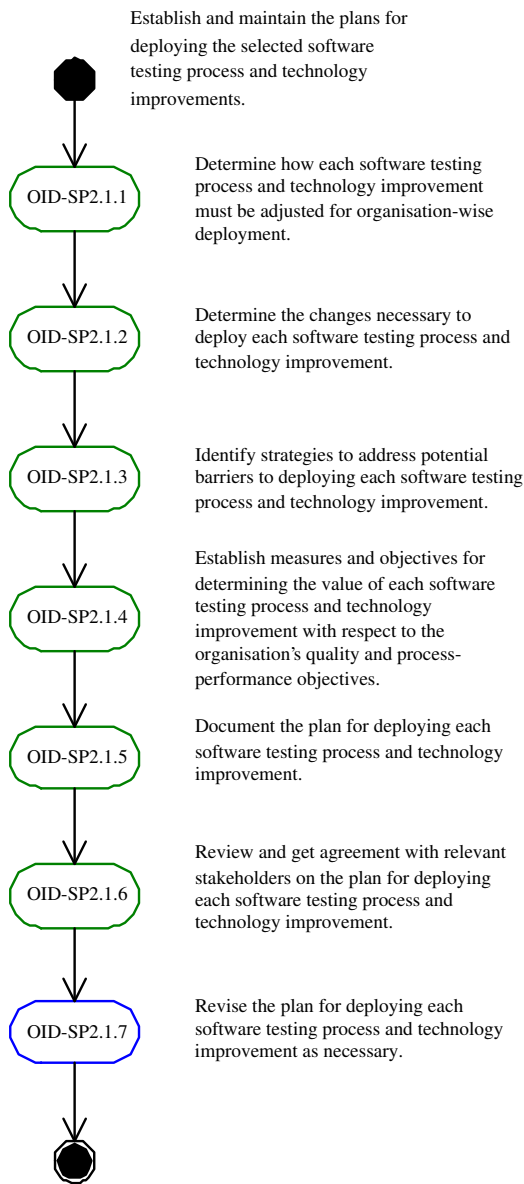


Figure 5.10: Capability Maturity Model Integration (CMMI) Software Testing Technology Change Management Mechanism Maps

SP 2.1 Plan the Deployment



SP 2.2 Manage the Deployment

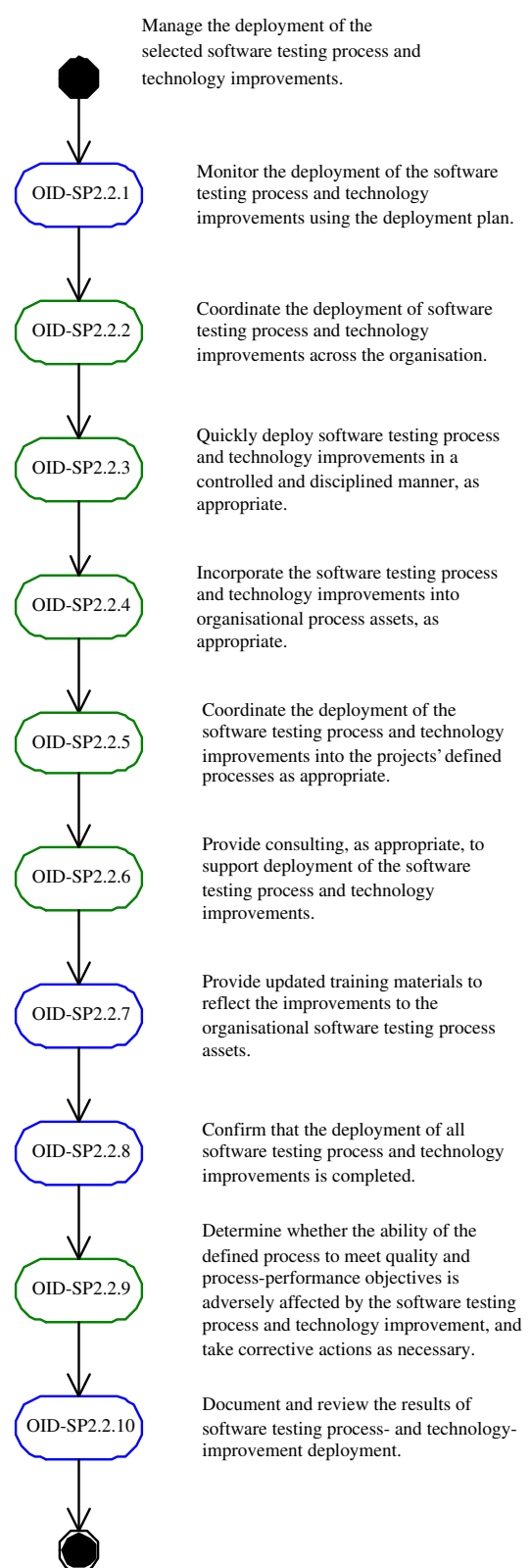


Figure 5.11: Capability Maturity Model Integration (CMMI) Software Testing Technology Change Management Mechanism Maps

SP 2.3 Measure Improvement

Effects

Measure the effects of the deployed software testing process and technology improvements.

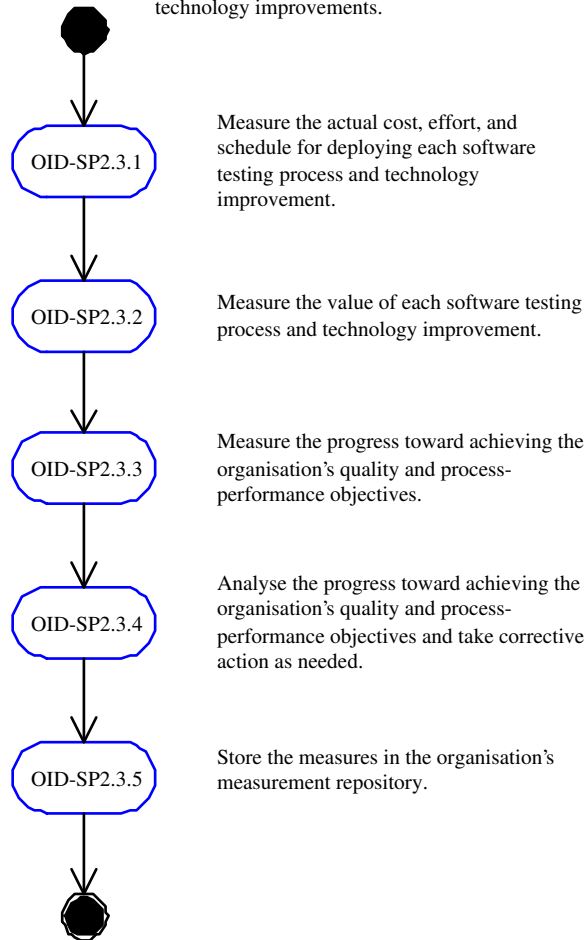


Figure 5.12: Capability Maturity Model Integration (CMMI) Software Testing Technology Change Management Mechanism Maps

Support Manager Principal Activity 1 (Lead the team in determining its software testing support needs and obtaining the needed software testing tools/facilities) specifies the requirements for implementing a software testing technology change management process whereas Support Manager Measures 1.1 (The team had appropriate software testing tools) and 1.2 (The team effectively used the software testing tools that it had) prescribe requirements for institutionalising a software testing technology change management process.

A policy map for TSPi software testing technology change management is depicted in Figure 5.13. As shown in Figure 5.13, there is a swimlane pertaining to the TSPi software testing technology change management goal. Figure 5.14 shows examples of activity orderings: an example of an ordering to enact or implement a software testing technology change management process and an example of an ordering to institutionalise software testing technology change management. Again, because the ordering of process implementation and institutionalisation activities is dependent upon project management concerns and scheduling, it is impossible to present all permutations of ordering.

Although the TSPi prescribes only one activity and two measures for implementing and institutionalising software testing technology change, this thesis does not develop any mechanism maps for team (or individual) software testing technology change management. Again, the reason for this is because there are an infinite number of possible mechanism maps (or more specifically, process implementations).

5.5 General Observations

The results of the case studies are summated in the list below.

ISO 9001 Clause 4.11 (Control of Inspection Equipment) implements software testing technology change management whereas other clauses (4.1 (Management Responsibilities), 4.2 (Quality System Requirements), 4.16 (Control of Quality Records), 4.18 (Training Requirements), and 4.20 (Statistical Techniques) institutionalise software testing technology change management.

Capability Maturity Model (CMM) Technology Change Management activities implement software testing technology change management whereas Technology Change Management commitments, abilities, measurements, and verifications institutionalise software testing technology change management.

Capability Maturity Model Integration (CMMI) Organisational Innovation and Deployment activities implement software testing technology change management whereas Organisational Innovation and Deployment commitments, abilities, implementation directives, and verifications institutionalise software testing technology change management.

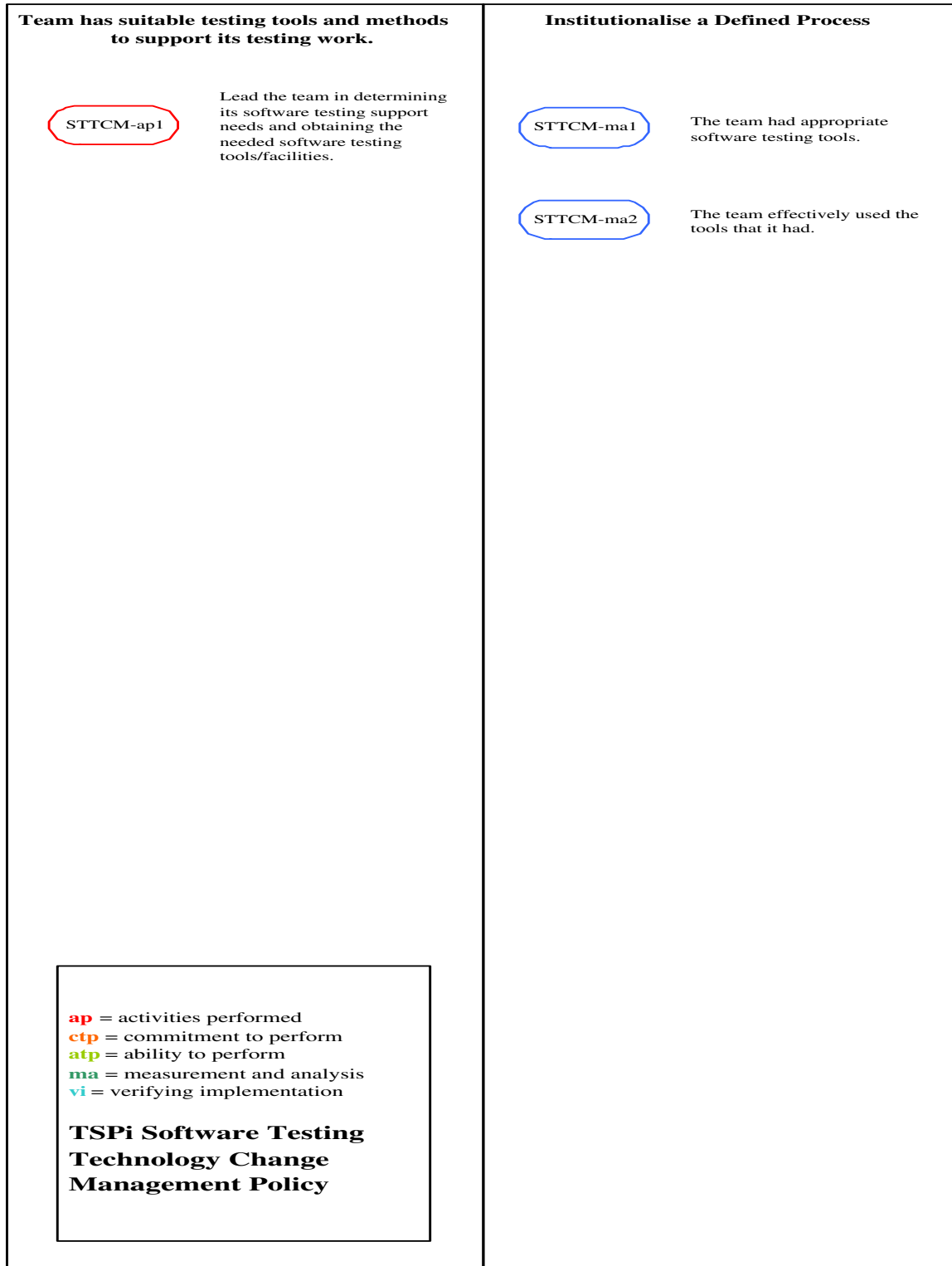


Figure 5.13: Introductory Team Software Process (TSPi) Software Testing Technology Change Policy Map

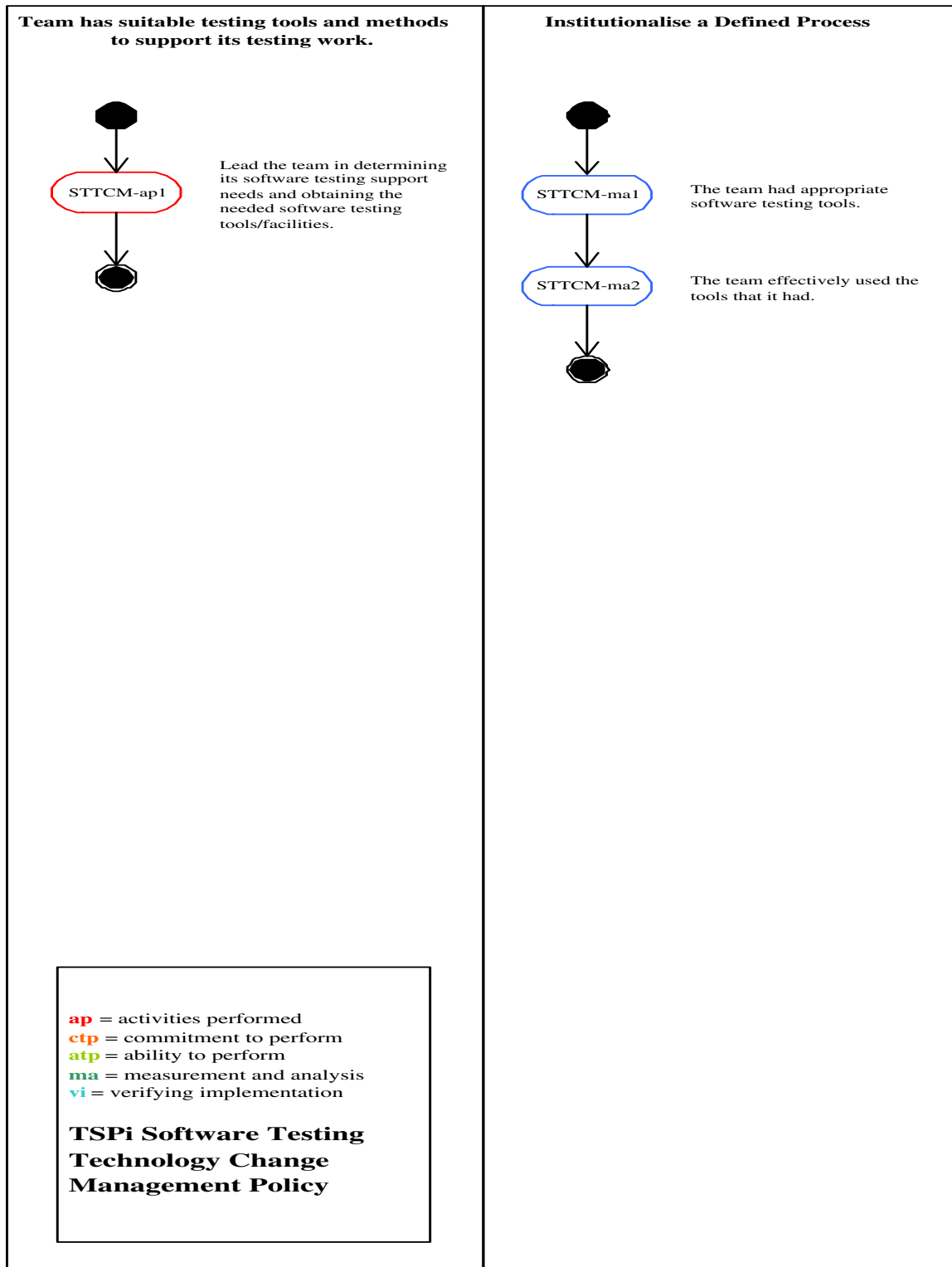


Figure 5.14: Introductory Team Software Process (TSPi) Software Testing Technology Change Policy Map with Ordering

Introductory Team Software Process (TSPi) Support Manager Principal Activity 1 implements software testing technology change management whereas Support Manager Measures 1.1 and 1.2 institutionalise software testing technology change management.

Because some activities pertain to more than one goal (that is, not all activities can be classified orthogonally according to context), it is important to ensure that connectives on policy maps incorporate activity context (where required). This is to ensure that a policy can have multiple mechanisms that differ in context.

As demonstrated with the ISO 9001, when activity types and/or activity contexts are not clear, the policy map construction process is made more difficult. Clearly, software testing technology change management process maps cannot be constructed expeditiously. One of the greatest difficulties in constructing process maps is that it is testing to elicit the ordering, and requirements and expectations (or pre- and post-conditions) of activities. The reason for this is that prescriptive software process improvement frameworks do not mandate project management schedules. Prescriptive software process improvement (or maturity) frameworks also do not mandate when measurements or verifications of technology change management activities are to be undertaken. Process maturity frameworks do however prescribe the requirements for implementing and institutionalising a software testing technology change management process. Once a process is institutionalised, software testing technology change management groups can use policy and mechanism maps to chronicle and recount their activities. In other words, once software testing technology change management groups (or individuals) are able to identify what policy is to be implemented or followed, they can use mechanism maps to plan or guide their activities.

Chapter 6

Conclusions

Basically, this research proposes a three-layer process map architecture for modelling software testing technology change management policies. The sandwich-like architecture facilitates circumspect policy and mechanism interpretation through the use of natural language constraints and documentation connectives. Essentially, documentation connectives are needed to stop up diagram nesting and explicate activity detail (or work product) whereas constraints can be used to ensure that model interpretation is circumspective. Moreover, a top-down modelling approach makes easy the progressive implementation of software testing technology change management processes. That is, skeletal process maps can be constructed and as more activity detail becomes available (or a need for further explanation becomes identified), the maps can easily be amended to assimilate the new detail.

While this research provides direct benefit to independent verification and validation (IV&V) organisations, it is general enough to be applied by any organisation that participates in software validation activities or more precisely, software testing technology evaluation and adoption. Moreover, effectual management of software testing technology change can help software testing organisations expeditiously equip themselves for software failure investigations.

Clearly, further work is needed to measure the effectiveness of or validate objectively software testing technology change management maps. At present, the only effective means of validating software testing technology change management maps is through subjective peer review (such as the Delphi technique [6]). Moreover, further work could include elucidating how process map course-plots can help direct software testing technology change management (or more generally, software testing process) improvement efforts or initiatives. More ambitiously, further work could also include constructing process maps for every key process area in the CMM [5] and/or CMMI. Alternatively, further work could include producing software testing technology change management process maps for other prescriptive software process improvement frameworks (such as Six Sigma). On a lesser note, further work is also needed to construct formal connectives to different work product types.

This research has shown that it is difficult to deduce policies and constraints from prescriptive software process improvement frameworks. Moreover, because software process maturity frameworks do not (and cannot) mandate exactly how to perform improvements, it is also difficult to elicit the ordering of activities and determine which activities can be effectively enacted in parallel. In other words, it is clear that mechanisms can not be diametrically obtained from outlines (or abstract policy statements or guidelines). However, the aim of this thesis was not to implement mechanics for each software testing technology change management policy but rather deduce a technique for constructing well-founded mechanisms. In fact, it often takes years for organisations to satisfy the prescriptive criterion set by software process improvement frameworks.

Although the three-layer process map architecture is a consequent of trying to manage software testing technology change management policies, mechanisms, and work products, it is general enough to be adapted to any type of technology change management.

Nevertheless, this thesis proposes one tactic for managing software testing technology change that facilitates the planning, enacting, attesting, expediting, communicating, and maintaining of software testing technology evaluations and adoptions (or more generally, software testing technology change management group work products). In summary, “Model-based Software Testing Technology Change Management” contends that software testing technology change management is facilitated by a structured approach to policy and mechanism modelling. In other words, administration of software testing technology evaluations and adoptions (or more generally, software testing technology change management group work products) is made easier by taking a structured, model-based approach to policy, mechanism, and work product management.

References

- [1] Anjard, R. P., (1998), "Process Mapping: a valuable tool for construction management and other professionals.", MCB University Press. Vol. 16, No 3/4, pp 79-81.
- [2] Soliman, F., (1998), "Optimum level of process mapping and least cost business process re-engineering.", International Journal of Operations & Production Management, MCB University Press, Vol. 18, No 9/10, pp 810-816
- [3] Peppard, J. and Rowland, P., (1995), "The essence of business process re-engineering.", Hemel Hempstead: Prentice Hall Europe
- [4] Bennett, S. and Skelton, J. and Lunn, K., (2001), "Schaum's Outlines of UML", McGraw Hill
- [5] Paulk, M. and Weber, C. and Curtis, B. and Chrissis, M., (1995), "The Capability Maturity Model", Addison Wesley
- [6] Pfleeger, S., (2001), "Software Engineering Theory and Practice", Prentice Hall
- [7] Glickman, S. and Becker, M., (1985), "A Methodology for Evaluating Software Tools", Conference on Software Tools, April 15-17 1985 New York, New York, USA, pp 190-196
- [8] Rezagholi, M. and Frey, M., (2000), "Managing Engineering and Product Technology: A Method for Technology Assessment", June 2000 Oulu, Finland, pp 181-192
- [9] Bruckhaus, T. and Madhavji, N. and Janssen, I. and Henshaw, J., (1996), "The Impact of Tools on Software Productivity", IEEE Software, September 1996 Volume 13 Number 5, pp 29-38
- [10] Brown, A. and Wallnau, K., (1996), "A Framework for Evaluating Software Technology", IEEE Software, September 1996 Volume 13 Number 5, pp 39-49
- [11] Curtis, B. and Kellner, M. and Over, J., (1992), "Process Modelling", Communications of the ACM, September 1992 Volume 35 Number 9, pp 75-90

- [12] Kitchenham, B. and Pickard, L. and Pfleeger, S., (1995), "Case Studies for Method and Tool Evaluation", IEEE Software, July 1995 Volume 12 Number 4, pp 52-62
- [13] "IDEF Family of Methods", <http://www.idef.com/default.html> (last accessed: 30/10/02)
- [14] Eva, M., (1999), "SSADM Version 4 - A Users Guide", McGraw-Hill
- [15] Mayer, R. and Menzel, C. and Painter, M. and DeWitt, P. and Blinn, T., and Perakath, B., "Information Integration for Concurrent Engineering (IICE) IDEF3 Process Description Capture Method Report",
http://www.idef.com/Downloads/pdf/Idef3_fn.pdf (last accessed: 30/10/02)
- [16] Meyer, B., (1992), "Applying Design by Contract", Computer (IEEE), Vol. 25, No. 10, October 1992, pp 40-51.
- [17] (2002), "CMMI(SM) for Software Engineering", Version 1.1, Staged Representation (CMMI-SW, V1.1, Staged),
<http://www.sei.cmu.edu/publications/documents/02.reports/02tr029.html> (last accessed: 30/10/02)
- [18] Humphrey, W., (2000), "Introduction to the Team Software Process", Addison Wesley
- [19] Rowley, D. and Ramakrishnan, S., (2002), "Forensic Applications of Software Analysis", 3rd International Conference - Law and Technology (LawTech 2002), November 6-7 2002, Cambridge, USA