A Holistic Method for Improving Software Product and Process Quality

By Elli Georgiadou

Director of Studies: Professor Richard Comley
Supervisor: Professor Anthony White

Submitted for the partial fulfillment of the Doctorate of Philosophy Degree

School of Science and Technology

Middlesex University

London
United Kingdom

August, 2018

CANDIDATE DECLARATION FORM

Name of Candidate: Elli Georgiadou

Student Number: M00388540

Thesis Title: **A Holistic Method for Improving Software Product and Process Quality**

Degree for which thesis is submitted: Doctorate of Philosophy

Statement of Associated Studies undertaken in connection with the programme of research (Regulation G3.1 refers). While a registered student of the university, I attended the following courses/workshops/ conferences where I presented the following peer reviewed papers:

Georgiadou, E., Siakas K.V., Berki E., (2003): Quality Improvement through the Identification of Controllable and Uncontrollable Factors in Software Development, 11th EuroSPI 2003 (European Software Process Improvement Conference), Graz, Austria.

Siakas K.V., **Georgiadou, E.** (2005): PERFUMES: A Scent of Product Quality Characteristics, The 13th Software Quality Management International Conference, SQM 2005, March 2005, Gloucestershire, UK.

Georgiadou, E. & George, C. (2006) "Information Systems Failures: Whose responsibility?" Proceedings of the 11th **IN**ternational conference on **S**oftware **P**rocess **I**mprovement - **R**esearch into **E**ducation and Training, (INSPIRE 2006), April, Southampton, UK, ISBN 1-902505-77-8.

Georgiadou, E., "A framework for the design and execution of controlled experiments in Software Engineering", Software Quality Management International Conference, Special 50th Anniversary of the BCS, Tampere, Finland, 2007.

Georgiadou, E. (2008) "GEQUAMO II Verification, Validation and Improvement of a Generic, Multilayered, Customisable, Software Quality Model ", Software Quality Management, 2008, Ulster University, March 2008.

Georgiadou, E., Sheriff, M. (2008) "Reconciling stakeholder conflicts by analysing apparently contradicting notions of value in SE projects", In: Software Quality Management International Conference, 2008, Belfast, Northern Ireland.

Georgiadou, E. "Navigating the labyrinth of software Re-words", 17th Software Quality Management International Conference, Southampton, UK, April 2009.

Georgiadou, E., Siakas, K., Balstrup, B. (2010), "The I^5P Visualisation Framework for Performance Estimation through the Alignment of Process Maturity and Knowledge Sharing", International Journal of Human Capital and Information Technology Professionals (IJHCITP) Vol. 2 No 2. (extended version of paper presented at EuroSPI, 2009)

Georgiadou, E., Siakas, K. (2013) "$VALO_5$ – Innovation, Maturity Growth, Quality and Valorisation", Systems, Software and Services Process Improvement Systems, Software and Services Process Improvement Communications in Computer and Information Science, Springer, Volume 364, 2013, pp 294-299.

Georgiadou, E., White, A., Comley, R. (2017) " CoFeD: A Visualisation Framework for Comparative Quality Evaluation", in Achieving Software Quality in Development and in Use, P Marchbank, M Ross, G Staples (Eds), 25th Software Quality International Conference, 2017.

Georgiadou, E., "Is the Composite Software Metric ρ (rho) a Good Metric?" in Computing and Quality, 26th Software Quality International Conference, 2018 ISBN: ISBN 978-0-9932889-9.

Georgiadou, E. "Reflections on the need for Disambiguation of Terminology for software Process Improvement", EuroSPI 2018, in Systems, Software and Services Process Improvement, Volume 896, Communications in Computer and Information , Springer, 2018 (in print – to be presented in September 2018).

I declare that while registered as a candidate for the university's research degree, I have not been a registered candidate or an enrolled student for an award of another university, academic or professional institution. I declare that no material contained in this thesis has been used in any other submission for academic award.

Signature of the candidate:

[illegible]

ABSTRACT

The concept of quality in general is elusive, multi-faceted and is perceived differently by different stakeholders. Quality is difficult to define and extremely difficult to measure. Deficient software systems regularly result in failures which often lead to significant financial losses but more importantly to loss of human lives. Such systems need to be either scrapped and replaced by new ones or corrected/improved through maintenance. One of the most serious challenges is how to deal with legacy systems which, even when not failing, inevitably require upgrades, maintenance and improvement because of malfunctioning or changing requirements, or because of changing technologies, languages, or platforms. In such cases, the dilemma is whether to develop solutions from scratch or to re-engineer a legacy system. This research addresses this dilemma and seeks to establish a rigorous method for the derivation of indicators which, together with management criteria, can help decide whether restructuring of legacy systems is advisable.

At the same time as the software engineering community has been moving from corrective methods to preventive methods, concentrating not only on both product quality improvement and process quality improvement has become imperative. This research investigation combines Product Quality Improvement, primarily through the re-engineering of legacy systems; and Process Improvement methods, models and practices, and uses a holistic approach to study the interplay of Product and Process Improvement. **The re-engineering factor rho (ρ), a composite metric was proposed and validated.**

The design and execution of formal experiments tested hypotheses on the relationship of internal (code-based) and external (behavioural) metrics. In addition to proving the hypotheses, the insights gained on logistics challenges resulted in the development of **a framework for the design and execution of controlled experiments in Software Engineering.**

The next part of the research resulted in the development of the novel, generic and, hence, customisable Quality Model **GEQUAMO,** which observes the principle of

orthogonality, and combines a top-down analysis of the identification, classification and visualisation of software quality characteristics, and a bottom-up method for measurement and evaluation. **GEQUAMO II** addressed weaknesses that were identified during various GEQUAMO implementations and expert validation by academics and practitioners.

Further work on Process Improvement investigated the Process Maturity and its relationship to Knowledge Sharing, resulted in the development of the **I^{5}P Visualisation Framework** for Performance Estimation through the Alignment of Process Maturity and Knowledge Sharing. **I^{5}P** was used in industry and was validated by experts from academia and industry. Using the principles that guided the creation of the GEQUAMO model, the CoFeD visualisation framework, was developed for comparative quality evaluation and selection of methods, tools, models and other software artifacts. CoFeD is very useful as the selection of wrong methods, tools or even personnel is detrimental to the survival and success of projects and organisations, and even to individuals.

Finally, throughout the many years of research and teaching Software Engineering, Information Systems, Methodologies, I observed the ambiguities of terminology and the use of one term to mean different concepts and one concept to be expressed in different terms. These practices result in lack of clarity. Thus my final contribution comes in my reflections on terminology disambiguation for the achievement of clarity, and the development of a **framework for achieving disambiguation of terms** as a necessary step towards gaining maturity and justifying the use of the term "***Engineering***" 50 years since the term *Software Engineering* was coined.

This research resulted in the creation of new knowledge in the form of novel indicators, models and frameworks which can aid quantification and decision making primarily on re-engineering of legacy code and on the management of process and its improvement. The thesis also contributes to the broader debate and understanding of problems relating to Software Quality, and establishes the need for a holistic approach to software quality improvement from both the product and the process perspectives.

Dedication

This work is dedicated to the haunting memory of my unborn child, and to the memory of four remarkable people who gave me life, unbounded love, and quality of life:

- my beloved **parents George and Elizabeth** who had no opportunity to study (my father finished primary school and my mother was illiterate) but the two of them enabled me to dream, instilled in me the love of learning, devotion to my teachers and dedication to duty and social justice;

  and

- my **uncle Lazaros**, an autodidact (who only attended 2 years of primary school), a polymath, thinker and philosopher who practised and taught me the dialectic method of enquiry, and problem-solving, and my **aunt Angeliki** who taught me patience and who, together with uncle Lazaros, offered me (a village girl) the opportunity to come to their house and family in town in order to study at secondary school. They also introduced me to the arts and art appreciation which continue to enrich my life.

Acknowledgements

My research journey was long, difficult, interesting, and rewarding. During this journey many people have accompanied me, inspired me, collaborated with me and helped me. Since I presented my first research paper in the First Software Quality Management International Conference in 1993 two remarkable people Professor Margaret Ross and EurIng Geoff Staples inspired me, trusted me to join the scientific committees of the SQM and INSPIRE conferences, and later on to help organise and chair both conferences on three occasions. They inspired and mentored me enthusiastically and generously throughout our collaboration on the International Committee of the British Computer Society Quality Specialist Group. For all these and for your friendship dear Margaret and Geoff, I owe you my deepest thanks and gratitude.

More recently my main supervisor and advisor Professor Anthony White provided immense inspiration through his own amazing research achievements and generosity to nearly 30 PhD completions. His continuing thirst for study and learning, as well as his ability and willingness to encourage and support researchers, like myself, to achieve their PhD completion, despite the various obstacles, delays and digressions, are remarkable. My deepest thanks go to you Tony for the scientific guidance and the trust in my research work and for the enormous encouragement you provided me with, without which I would not have been able to submit my thesis. My completion brings the total number of your PhD completions to 30. This number speaks for itself.

My Director of Studies Professor Richard Comley has given me valuable scientific advice and encouragement, as well as practical support not only for this research but also for my European Activities and the management of 6 European Tempus projects on Research and Knowledge Transfer many of which run simultaneously! Richard, without your endorsement and support I would not have been able to build a large network of research contacts and collaborators across the world. Many thanks for your understanding especially at difficult times (and there have been many).

Special thanks go to my two Universities (University of North London (now London Metropolitan University)) and Middlesex University for their generous support to serve as a European and International Affairs Faculty Co-ordinator, and to attend a large number of international meetings and conferences where I presented my work, received feedback, validated my models and collaborated on many research projects and initiatives. Professor Martin Loomes, you always encouraged me to pursue my research (saying it is never too late), and you appreciated my mentoring of younger colleagues through collaborative research. Thank you Martin for respecting and trusting me to represent the University at European and International levels for recruitment, forging collaborative links and carrying out joint research. Professor Balbir Barn you spurred me on not long ago to concentrate on my PhD by what felt like a hurtful remark at the time. You said “you have no passion for your research”. Balbir, your remark re-ignited my passion which had always been there but had been masked by adversity, illness and an overwhelming amount of work. Thank you for focussing my efforts!

My European Funded Projects including Socrates/Erasmus, Tempus on Curriculum Development and Research and Knowledge Transfer brought me into contact with many colleagues from various countries; too many to mention here but special thanks are due to Dr Juri Valtanen from Finland for the stimulating discussions and subsequent collaboration and joint publications on Problem Based Learning. Also Professor Rita Gevorgyan from Armenia for over 10 years of collaboration and co-authorship on Educational Quality Management where I was able to apply models and knowhow from Software Engineering to Process Improvement in Education and Educational Management in countries of transition following the collapse of the Soviet Union.

I want to thank all my co-authors and especially all my colleagues who volunteered their various groups of students to participate in my experiments, and Professor Witold Suryn for his knowledge on Software Quality Engineering and incisive comments during our joint work on the evaluation of Quality Models. Also I owe thanks to the many journal and conference reviewers who provided valuable feedback for which I am grateful.

Going back to the early years of my research I had patient support by Mr Chris Sadler who not only was my supervisor, he was also my Head of Department for many years at the University of North London. Chris, during my early research steps our discussions steered me towards reverse engineering which reinforced my focus on re-engineering. Thank you Chris for all the years of collaboration, and for those stimulating discussions and witty comments especially during the latest part of our service at Middlesex University when we shared an office. Dear Chris, you trusted me to join the DESMET research project team and you sent me to attend my first conference (Eurometrics in 1992) where I met Professor Barbara Kitchenham. She encouraged me to continue delving deeper into concepts of software measurement, and later on gave me feedback on my experimental designs. Thank you Barbara – your work inspired me to focus further on software metrics. This in turn pointed me towards the then young Professor Norman Fenton who, for a brief time, was my external advisor, and who gave me valuable feedback on my first experiment. Thank you Norman for the respect and encouragement you showed me.

At the University of North London I had valuable support from Professor Ian Haines (my Dean of Faculty), Professor Jim Yip my supportive Head of Department (and very briefly my supervisor), Professor John Charalambous who was not only inspiring to me but to so many with his energy and enthusiasm, his love of Education and love for Cyprus. John, you have been instrumental in my engaging in research and in international recruitment for the University. Professor Mikis Stassinopoulos gave me valuable advice on selecting suitable experimental designs and on analysing the experimental data. Many thanks to Dr George Karakitsos whose static analyser enabled me to analyse legacy code when considering re-engineering. Finally Nick Silvester provided effective and valuable on-going technical support. Thank you all for everything.

While at the University of North London I was privileged to meet a group of Visiting Professors who are brilliant Ukrainian scientists and warm people who became my friends: the eminent Professor Igor Kovalenko (a student of the eminent mathematician Kolmokorov) and his colleagues Doctors Nicolai Kuznetsov, Valentyn Shpak, and Lyidmila Zavadskaya. I enjoyed and benefited greatly from our many discussions on Statistics, Mathematics and Cybernetics here in London during your various visits and during my visits to Kiev. Thank you for inviting me and hosting me in Kiev, where I presented my early research findings receiving crucial feedback, but also for introducing me to the work of the Institute of Cybernetics, and to the Ukrainian history and culture.

I thank Professor Inger Eriksson who in the late 90s invited me and two of my then PhD students to design and deliver a short course on Software and Systems Quality at the University of Turku, and later on invited me together with Professor Gary Richardson to visit them at North Carolina University where I presented my research and gained feedback.

My sincere thanks go to my colleague Maya Milankovic-Atkinson for stimulating discussions in the early years on Object-Orientation and for generously offering her various student cohorts, all novice programmers, to act as the subjects of my many experiments at the University of North London. We continued our collaboration at Middlesex University. Thank you dear Maya for 30 years of collaboration (research and teaching) and true friendship born out of common approaches to teaching and research, and based on our shared culture, personal struggles and understanding reached through wars and injustices perpetrated on our two countries.

Many thanks go to Professor Darren Dalcher with whom I shared an office for 4 years. Our discussions on project failures strengthened my interest in Systems and Projects failure.

Also many thanks go to the late Professor Tully for inviting me to successfully co-supervise two PhD students.

My interest and focus on Process Improvement pointed me to the EuroSPI (now EurasiaSPI) Conference and to Dr Richard Messnarz, another polymath, a scientist, a commentator, a critic of socio-cultural and historical events and developments. Thank you Richard for creating such a thriving network of researchers and practitioners within the Software Process Improvement movement. Thank you also for the stimulating discussions and for the opportunities you gave me to attend the EuroSPI conference, present my work and validate my models by conference delegates from industry and academia.

My deepest thanks go to the thousands of undergraduate and hundreds of Masters students (over 33 years of teaching) who often pilot-tested and validated my various models and frameworks which were incorporated in modules and programmes (some of which are still being taught).

Most importantly I thank my 10 PhD students who in an unorthodox, (prothysteron) manner completed their studies under my guidance well before myself. I learned as much from all of them as I taught them. Many of them continue to carry out collaborative research with me evidenced by our many joint publications. Our meetings and discussions were always stimulating, rewarding, and inspirational. You are all now enjoying successful and satisfying careers which make me very happy and proud. I feel privileged to have been given a chance to supervise you all, to teach you but also to learn from you, to co-investigate, to innovate and to co-create. Lastly thank you all for urging me to study for my own PhD – and this is the reason I call this process a prothysteron (i.e. putting last what should have been first) which may be unorthodox but which I do not regret.

In particular, I want to thank deeply Dr Eleni Berki (one of my early PhD students), a creative and inspirational person, a brilliant scientist, a deep thinker and a charismatic, multifaceted and multitalented human being and researcher. Eleni, I am proud of your commitment to the struggle and for your significant scientific contribution starting from your undergraduate studies when you endeavoured to integrate Soft Systems Methods and Formal Methods, proceeding to CDM-FILTERS, your PhD and now surrounded by your own research team working on diverse subjects including Open Source, Security, Creativity, Social Impact of Technologies to list but a few of your many areas of interest, knowledge and expertise.   You have been my student, my colleague, my friend and as a daughter to me, always urging me to complete my PhD studies. Thank you very much for everything. I owe many thanks to Professors Mike Holcome, and Kalle Lyttinen; I learned a lot from you, your research and your way of thinking especially during our co-supervising Eleni Berki.

Another of my early PhD students is Professor Kerstin Siakas, a multicultural polyglot, with whom I co-authored in excess of 40 papers (since the mid-1990s) on a large range of topics including Quality Management, Process Improvement, and the influence of Culture on Systems and Organisations, as well as numerous pedagogic matters such as Distance Education, Information Literacy and Social Media. Kerstin, you have been such a constant, effective, organised and supportive co-researcher! Above all you have been a real friend especially at times of adversity (and there have sadly been many such times). I cannot thank you enough for the encouragement and support to make one last effort to complete my thesis.

There is also a very special, indefatigable, socially and politically committed human being Dr Janet Shapiro who has been in my life since October 1970. It is you Janet, my Statistics Lecturer in the 70s, my colleague in the 80s and 90s, and in the last 25 years my close and reliable friend (more like family to me) with whom I share ideology, commitment to social justice, interest in the Arts and in Radical Statistics! Thank you Janet for always reminding me to re-start my studies and to be kind to myself! Thank you above all for being there for me, especially in recent years.

Warm thanks go to my colleague, research collaborator and very dear friend Geetha (Dr Abeysinghe), and her husband Vijay, who have been supportive to me in so many ways. Geetha and Vijay, I admire and love you for your gentle calmness and philosophical life attitude even in the face of adversity. You always enable me to draw courage and energy by your example.

I owe deep thanks and gratitude to the compassionate and supportive colleague Professor Gill Whitney. Dear Gill thank you for standing by me in so many ways at the time of terrible loss and bereavement.

I want to also thank all my colleagues, research collaborators, co-authors and good friends: Drs Nawaz Khan, Mohamed Sheriff, Carlisle George, Elke Dunker, Elizabeth Stokes and Carlisle George; also Professor Chris Huyck, Chris Kindberg and Ed Currie. Warm thanks go to my compatriot, ex-colleague and very dear friend Professor Stylianos Hatzipanagos for his kindness, wit, and shared research and socio-cultural interests.

Special thanks to my dear colleague Dr Harjinder Rahanu, a bibliophile and an excellent lecturer, for our on-going collaboration on EU projects, researching and co-authoring papers primarily on Ethics, Pedagogy, Information Literacy, and on Process Improvement. Harjinder, I am thankful for our shared worldview (Weltanschauung or Κοσμοθεωρία) on education, politics and society at large, and love of family and books of course!

I want to thank deeply the ingenious and untiring Leonard Miraziz. Thank you dear Leonard for providing me with endless technical support throughout the years. Without your help, I would not have been able to work, especially since I nowadays work mostly from home.

Thank you all dear colleagues for the research collaborations and for the many stimulating discussions on research, teaching and on socio-cultural matters. I enjoyed, and still enjoy, working with many of you, teaching with you, thinking with you, writing with you. If nothing else, Middlesex University has given me such a wonderful bunch of friends for which I will always be grateful.

I want to give deep and warm thanks to another special person Dr Nikos Paltalidis who over the last 21 years has been a constant and a very special presence in my academic, intellectual, research and family life. Niko, you are the son I never had, and as such I watched your journey from the first year of your undergraduate studies, through to your Masters and PhD, and now your University teaching position, with interest and pride for your achievements. Thank you for your kindness and for being there for me especially at my hours of need.

Also many thanks go to Ms Annita Zirki, a secondary school Computing teacher, who was my student and supervisee during her undergraduate studies, and now my friend who shares my love of music, and who provides solid technical support whenever she visits me in London! Annita, I hope that my difficulties in completing my thesis do not put you off embarking onto your own PhD studies.

I also thank my most recent American friend, Adam Dufour, for his kindness but also for many stimulating discussions on the latest technologies and methodologies, crossing over from Software Engineering to Accounting, Finance, Government, Politics, and the Arts. Our discussions show how things that seem isolated within certain knowledge domains are organically inter-connected by thought processes, and can indeed be addressed by similar problem-solving methods.

My penultimate thanks go to Professor Mary Plastira-Valkanou and Nikos Valkanos and their four daughters, for a deep and long friendship of over 25 years built on our shared understanding of the world, the meaning of human creativity, our commitment to research in our respective fields and the struggle for a fairer society, as well as the education of future generations. I cherish our discussions and exchange of ideas on research, education, culture, civilisation, history, language and poetry (of course), and your four children whom I have known as babies and now four intelligent, beautiful, successful and simply delightful and lovable young women! Thank you for sharing so many precious times with all of you.

Last but not least, my loving thanks go to all my family, my siblings Maria, Michael and Menelaos and their children, grandchildren and great-grandchildren. It was with your love and practical help, however small, that I managed to study (the first one from our family who had the opportunity to go to University). I often wonder with sadness what our lives/your lives might have been like had you had indeed such an opportunity…? But my warmest and biggest thanks go to my dearest, humorous, and brilliant polymath brother Melis (Menelaos) and his amazingly generous, organised, warm and kind wife Despina. The two of you have always been patient, supportive and understanding especially during testing times for our parents and for me. A thousand thankyous for everything (Χίλια ευχαριστώ για όλα) go to you, and your beautiful children and grandchildren for always making me welcome in Cyprus. I conclude with many heartfelt wishes for the future of the younger generations, and for peace in the family and in our long suffering country, Cyprus. I love you all as always/Σας αγαπώ όλους όπως πάντα.

Elli/Ελλη/Γιαγιά ‘Αγγλία’, London August, 2018

List of Figures

Figure **Page**

List of Tables

TABLE OF CONTENTS

PROLOGUE

The research reported in this thesis is the result of work carried out over a period of 35 years during which the results of the research were published in journal papers and refereed international conference papers as well as book chapters. Over this period major technological developments and paradigm shifts in information systems development have informed and shaped the direction, experience and the outputs of this research. Increasingly deeper insights into the various issues relating to software quality were gained through the study of the literature, the design and execution of formal experiments, and the development of models which were validated by experts in academia and industry. The developed models were also integrated into several academic programmes and modules, and offered at two UK universities where the author worked as a Principal Lecturer in Software Engineering and Information Systems. Elements of Quality Management Processes have been applied to the Quality Management and Quality Evaluation of three European Tempus Projects for Knowledge Transfer.

The structure of the thesis is as follows:

Chapter 1 is the **Introduction** which presents the motivation, context, scope, objectives, formulation of research hypotheses and research methods employed.

Chapter 2 provides a discussion of research philosophy, research methods, their strengths and weakness, and the selection of the research methods used in this research.

Chapter 3 is a critical review of lifecycle models, information systems development methodologies, process models, standards, and quality models, and the development of the generic, multilayered, and customisable software quality model GEQUAMO. Some deficiencies of GEQUAMO were identified during its use resulting in GEQUAMO II which was validated by experts from academia and industry.
(Papers I, II, III, IV, V)

Chapter 4 identifies controllable and uncontrollable factors in software development and proposes improvements to the ISO 7126 quality model.
(Papers VI, VII)

Chapter 5 describes the derivation of the composite metric rho (ρ**) which is an indicator** for answering the question on whether deficient or malfunctioning systems should be scrapped or re-engineered. Achieving improvements through the manipulation of legacy code was carried out through the design and execution of formal experiments for testing the hypotheses. Gradual understanding of the problems with logistics encountered during the design and execution of the experiments lead to the development of a Framework for the Management of similar experiments.
(Papers VIII, IX, X, XI, XII, XIII)

Chapter 6 starts with a discussion of the main drivers of process improvement and presents the development, use and validation two frameworks and one model:

- the **I^5P Visualisation Framework** for Performance Estimation through the alignment of Process Maturity and Knowledge Sharing;
- the **$VALO_5$ Model** of innovation, maturity, quality and valorisation; and
- The **CoFeD – visualisation framework** for comparative quality evaluation.

(Papers XIV, XV, XVI)

Chapter 7 presents reflections on metrics validity through revisiting the composite metric rho (ρ). It also considers the ethical and legal dimensions of systems failures as well as the potential value gains if we can r*econcile stakeholder conflicts by analysing apparently contradicting notions of value in SE projects* concludes with the recurring problem of terminology management/mismanagement and the need for terminology disambiguation for creating clarity.

(Papers XVII, XVIII, XIX, XX)

Chapter 8 is the overall conclusion which summarises the Contributions to Knowledge and the Significance of the Study. This research contributes to a broader understanding of information systems and the elements within the environment that influence the quality of process and product. The research also explored the use of formal experiments to test hypotheses and beliefs across the software engineering and information systems research and practitioner communities. The need for a legal and ethical stance towards software failures and their social implications is highlighted. The need for disambiguation of terms so that Software Engineers 'speak the same language' is also discussed. Limitations of the research as well as directions of future work complete the thesis.

The Epilogue is a personal reflection on my research journey, my quest for knowledge and understanding, the creation of new knowledge, and the development of models and frameworks as mechanisms for product and process improvement, the application of the author's research findings in industry, as well as in curriculum development, and European Projects for Research & Knowledge Transfer. It concludes with a brief discussion on the role of educators in raising awareness and the sense of social responsibility in their students, who will be the future software engineers. It outlines the benefits of improved quality in software products and processes.

Submitted Publications in Thematic Groupings

Theme 1: Software Quality, Software Crisis and Attempts to Address it

I. **Georgiadou E**., Sadler C., (1995) "*Achieving quality improvement through understanding and evaluating Information Systems Development Methodologies*", 3rd International Conference on Software Quality Management, SQM'95, Seville, Spain ISBN 1-853812-416-8,

Number of citations 2

II. **Georgiadou, E. "**Software Process and Product Improvement - A Historical Perspective", International Journal of Cybernetics, Volume 1, No1, Jan 2003 pp172-197.

Number of citations 60

III. Marc-Alexis Côté, Witold Suryn, **Elli Georgiadou**: "*In search for a widely applicable and accepted software quality model for software quality engineering"*. Software Quality Journal 15(4): 401-416 (2007).

Number of citations 45

IV. **Georgiadou, E.** (2003) GEQUAMO: a generic, multilayered, customisable, software quality model. Software Quality Journal, 11 (4). pp. 313-323.

Number of citations 42

V. **Georgiadou**, E. (2008) "GEQUAMO II Verification, Validation and Improvement of a Generic, Multilayered, Customisable, Software Quality Model ", Quality Issues for Business Software Quality Management, 2008, Ulster University, March 2008. ISBN 978-1-906124-05-2

Number of citations 1

Theme 2: Measurable and Controllable Factors in Software Development

VI. **Georgiadou, E.,** Siakas K.V., Berki E., (2003): Quality Improvement through the Identification of Controllable and Uncontrollable Factors in Software Development, 11th EuroSPI 2003 (European Software Process Improvement Conference), Graz, Austria, 10-12.12.2003.

Number of citations 14

VII. Siakas K.V., Georgiadou, E. (2005): PERFUMES: A Scent of Product Quality Characteristics, The 13th Software Quality Management International Conference, SQM 2005, Gloustershire, UK Current Issues in Software Quality, ISBN 1-902505-67-0

Number of citations 13

Theme 3: A suite of controlled experiments

VIII. **Georgiadou E.,** Karakitsos G., Sadler, C., Stasinopoulos D (1993). *"An experimental examination of the role of re-engineering in the management of software quality*", 1st Software Quality Management International Conference II Vol., Computational Mechanics Publications, ISBN 1-85312-225-4.

Number of citations 1

IX. **Georgiadou, E.,** Karakitsos G., Sadler C., (1994) *"Improving the program quality by using the re-engineering factor metric ρ",* the 10th. International Conference of the Israel Society for Quality, Jerusalem, ISBN1-85312-225-4.

Number of citations 2

X. **Georgiadou, E**., Karakitsos, G., Sadler C., Stasinopoulos D, Jones, R. (1994) *Program maintainability is a function of structuredness,* 2nd International Software Quality Management, Computational Mechanics Publications, Edinburg, Scotland

Number of citations 1

XI. **Georgiadou, E.,** Keramopoulos, E. (2001) "*Measuring the Understandability of a Graphical Query Language through a Controlled Experiment"*, 9th International Conference on Software Quality Management, University of Loughborough, UK. Pathways to Software Quality, ISBN 1-902505-40-9

Number of citations 9

XII. **Georgiadou, E., (2007)** "A framework for the design and execution of controlled experiments in Software Engineering", Software Quality Management International Conference, Special 50th Anniversary of the BCS, Tampere, Finland, Software Quality in the Knowledge Society, ISBN 978-1-902505-96-1

Number of citations 1

Theme 4: Performance Estimation, Process Maturity

XIII. **Georgiadou, E.,** Siakas, K., Balstrup, B. (2010), "*The I^5P Visualisation Framework for Performance Estimation through the Alignment of Process Maturity and Knowledge Sharing"*, International Journal of Human Capital and Information Technology Professionals (IJHCITP) Vol. 2 No 2.

Number of citations 5

XIV. **Georgiadou, E**., Siakas, K. (2013) "*VALO$_5$ – Innovation, Maturity Growth, Quality and Valorisation"*, Systems, Software and Services Process Improvement Systems, Software and Services Process Improvement Communications in Computer and Information Science, Springer, Volume 364, 2013, pp 294-299.

Number of citations 5

XV. **Georgiadou.,** White, A., Comley, R. (2017) " *CoFeD: A Visualisation Framework for Comparative Quality Evaluation"*, in Achieving Software Quality in Development and in Use, P Marchbank, M Ross, G Staples (Eds), 25th Software Quality International Conference, Achieving Software Quality in Development and in Use ISBN: 978-0-9932889-6-8

Number of citations 1

Theme 5: Reflections Metrics Validity, Ethical Dimension of Systems Failures, Value Gains and Terminology Disambiguation

XVI. **Georgiadou**, E., (2018) "*Is the Composite Software Metric ρ (rho) a Good Metric?" in* Computing and Quality, 26th Software Quality International Conference, 2018, ISBN: 978-0-9932889-9.

Number of citations 0

XVII. **Georgiadou, E.** (2009) "Navigating the labyrinth of software Re-words", Software Quality in the 21st Century 17th Software Quality Management International Conference, Southampton, UK, ISBN 978-1-906124-22-917

Number of citations 2

XVIII. **Georgiadou, E**. *"Reflections on the need for Disambiguation of Terminology for software Process Improvement",* EuroSPI 2018, in Systems, Software and Services Process Improvement, Volume 896, Communications in Computer and Information , Springer, 2018

Number of citations 0

XIX. **Georgiadou, E**. & George, C. (2006) *"Information Systems Failures: Whose responsibility?"* Proceedings of the 11th INternational Conference on Software Process Improvement - Research into Education and Training, (INSPIRE 2006), April, Southampton, UK, ISBN 1-902505-77-8.

Number of citations 2

XX. **Georgiadou, E**., Sheriff, M. (2008) "*Reconciling stakeholder conflicts by analysing apparently contradicting notions of value in SE projects*", In: Quality Issues for Business Software Quality Management International Conference, 2008, Belfast, Northern Ireland. ISBN 978-1-906124-05-2

Number of citations 0

Additional papers

Papers 1 to 10 (below) report on collaborative research which contributes to various aspects of the on-going Software Quality debate and practice.

1. **Georgiadou. E.,** Evaluating the evaluation methods: Data Collection and Storage System using the DESMET Feature Analysis, the tenth international conference of the Israel Society for Quality, Nov 1994, pages467-474

2. Milankovic-Atkinson, M., **Georgiadou, E.** "*Object-Oriented Metrics for reusability*", Software Quality Management International Conference, Cambridge, UK, Apr. 1996.

3. Siakas, K.V., Berki, **Georgiadou, E.,** Sadler, C. *"The complete alphabet of quality software systems",* 7th World Congress for Total Quality Management, New Delhi, India, Feb. 97.

4. Berki, E. **, Georgiadou, E**. "*A comparison of qualitative frameworks for information systems development methodologies",* In Proceedings of The Twelfth International Conference of The Israel Society for Quality (Jerusalem, ISRAEL, December 1998.

5. **Georgiadou, E.,** Milankovic-Atkinson, M. "A formal experiment to verify Object-Oriented Metrics", 4th INSPIRE'99, Crete, September 1999.

6. Barbor, N &**Georgiadou, E**. [2002] Investigating the applicability of the Taguchi Method to Software Development, Proceedings of Quality Week, San Francisco. USA, July 2002

7. Berki, E., **Georgiadou, E.,** Holcombe, M.[2003a]: "Process Metamodelling and Method Engineering as Tools for Improved Software Quality Management - *A Chronological Review and Evaluation Critique Considering the Need for a New Scientific Discipline*", In the Proc. of Ross, M., Staples, G. (Eds.) 11th International Conference on Software Quality Management, SQM 2003, Glasgow, April 2003 Process Improvement and Project Management Issues, 1-902505-53-0

8. Siakas, K.V. &**Georgiadou, E.** (2008) "*Knowledge Sharing in Virtual and Networked Organisations in Different Organisational and National Cultures"*, eds. Ettore Bolisani, Building the Knowledge Society on the Internet, Idea Publishing ISBN: 978-1-59904-816-1, Part 1, Chapter 3.

9. Berki E., Siakas K.V., **Georgiadou, E.,** (2006): "*Agile Quality or Depth of Reasoning? Applicability versus Suitability Respecting Stakeholders' Needs"*, eds. Stamelos Ioannis and Sfetsos Panagiotis, Agile Software Development Quality Assurance, Idea Publishing ISBN: 9781599042169, pp. 23-55.

10. Estdale, J**., Georgiadou, E**. *Applying the ISO/IEC 25010 Quality Models to Software Product*, EuroSPI 2018, in Systems, Software and Services Process Improvement, Volume 896, Communications in Computer and Information , Springer, 2018 (in print)

CHAPTER 1 INTRODUCTION

1.0 Introduction to the chapter

This chapter presents the motivation, context, scope, objectives, formulation of research hypotheses and research methods employed. This thesis contributes a holistic approach to the broader debate and understanding of problems relating to Software Quality.

1.1 Background

This research focuses on the interdependencies of software product and software process; also on the quest for continuous improvement. It provides new knowledge in the form of a novel composite metric/indicator, models and frameworks which can aid quantification, and hence decision making, primarily on the efficacy of re-engineering legacy code and the management of process improvement.

As many researchers, practitioners and members of the public know, quality in general is transparent (and could even be taken for granted) when present, but is easily and immediately recognised in its absence. Software quality is no exception. However, defining and measuring quality is extremely difficult. Kitchenham (1996) and Kitchenham and Pfleeger (1996) emhpasised that: “quality is an elusive concept”.

Software artifacts even 'small programs' are among the most complex artifacts that humans produce, and software development projects are among our most complex undertakings. Nowadays our lives are governed by computers, communications and computer-based systems. Computer Systems and Information Systems play a very central role in organisations and the demands on information systems are continuously increasing. At the same time as information systems become more complex, more people are involved with software development and the need for repeatable processes have become necessary. Deficient quality of software systems result in failures which lead to significant economic losses but more importantly to loss of human lives.

There are many different causes of failures in computer-based systems including physical faults, maintenance errors, design and implementation, mistakes resulting in hardware or software defects, and user or operator mistakes. The causes of these failures are varied but often they are not foreseen and hence they are discovered too late in the process and invariably unexpectedly. In 1968 at the land mark Software Engineering Conference (NATO, 1968) the term Software Crisis was coined as a mark of recognition that there was a serious problem of systems failures. The quality of systems therefore became questionable.

Similarly most software projects can be considered at least partial failures because few projects meet all their cost, schedule, quality, or requirements objectives. Many surveys have indicated that as many as 50 % -75% of information systems projects are total or partial failures (Polymenakou and Serafeimidis, 1995).

According to the Standish Group reports CHAOS published regularly since 1995, (CHAOS, 2015), Liebowitz, J., (2015), (Dalcher, 2017) many of these deficient systems are never used or abandoned after release. Researchers and practitioners have sought to understand the reasons for failures and have developed a plethora of methods, techniques and tools for alleviating the likelihood of failure.

The concerns about late deliveries of software, with low reliability and high maintenance costs have directed most efforts to improve software quality. There is universal agreement that although it may be difficult to understand, define and measure quality, we all readily recognise its absence.

Since the 1950s Deming (1986) recognised that quality is difficult to define mainly because it is difficult to express future user needs into measurable attributes. Users want a product to satisfy their needs within specified time and cost limits. Nearly 30 years later, Juran et al. (1981) defined quality as "fitness for purpose”. Customers hold different opinions, for different reasons about quality. In the same year Crosby extended this definition to "Quality is conformance to requirements” (Crosby, 1979).

A decade later the emphasis shifted to specification with Cullen asserting that “Quality is conformance to specification” (Cullen, 1989). The most all-encompassing definition is provided in ISO 9001 second edition (1994), and later on ISO9126 namely “Quality is the totality of features and characteristics of a product or service that bear on its ability to satisfy specified or implied needs”. This definition recognises that many different aspects make up and affect quality. These factors are often interdependent and tradeoffs may be necessary to achieve an acceptable level of compromise.

Gillies proposed that “*Quality is generally transparent when present, but easily recognised in its absence* “(Gillies, 1992). He elaborated as follows:
Quality is *not absolute***.** It means different things in different situations. In one situation the safety of the car can be the most important attribute for quality, and in another situation it can be the speed and the acceleration. It depends on what is important for us in a specific moment.

Quality is *multidimensional:* It depends on many different factors. There is seldom only one factor that decides the quality of a product or a service. There are usually many details that make us to consider a product as a quality product.
Quality is *subject to constraints***:** Some products are usually considered as quality products (for example Rolls Royce) but because of the high price we choose to consider some other products as quality products.
Quality is about *acceptable compromises***:** Some criteria may be sacrificed because of constraints such as prohibitive costs settling for a product or service of lower quality.

Quality criteria are *not independent*: They interact with each other causing conflicts. For example the more complex a system is, the more unreliable it is likely to be. That is complexity is likely to affect reliability.

Gillies proceeded to define five views of quality which may be in conflict with each other. These are:
The transcendent view: The classical definition of quality meaning "elegance".
The product-based view: The economist's view, higher quality = higher cost.
The user-based view: It is meeting the users’ requirements and fitness for purpose.
The manufacturing view: Measures quality in terms of conformance to requirements.
The value-based view: Provide what the customer requires at a price they can afford.

Kitchenham (1996) encapsulated many of the issues relating to the understanding, definition and measurement of Software, arguing that *"Quality is a complex concept* *that means different things to different individuals. It can be highly context dependent. This means that there can never be any simple measure of quality that will be accepted by everyone. If you are interested in assessing or improving quality in your organisation, you must ensure that you define what aspect of quality you are interested in and how you are going to measure it. In fact, if you define quality in a measurable way, it is usually easier for other people to understand your viewpoint. “*

More than 75 % of software projects run late and as many as 30% of projects are cancelled (CHAOS, 1994; Dalcher, 2005, Jones, 2010; Chaos, 2016 Dalcher, 2017).

A constant quality improvement quest resulted in the development of numerous approaches to building and maintaining software, from new technologies to progressive processes and frameworks as reported by among others Jayaratna (1994), Jackson (1995), Beck (2001), Vandierendonck & Mens (2011), Wolf (2016), Dalcher, (2017).

New languages were often believed to have almost magical powers for resolving the Crisis. Automated tools, formal methods, object-orientated methods have been proposed as alternative 'religions' with many software engineers becoming almost fanatical followers of one or the other approach (Georgiadou & Sadler, 1995), (Siakas *et al* 1997). Despite these efforts systems continue to fail with dramatic frequency and impact.

At this juncture it is necessary to introduce some fundamental terms in the context of Software Engineering and the scope of this research namely Quality, Measurement and Metrics, Product, Process, Resources and Projects.

1.2 Fundamental Terms Pertinent to this research

Quality according to Kitchenham (1996) is an elusive concept, difficult to define and even more difficult to measure. Quality means different things to different people. The approach of this research is to look at quality in a holistic manner studying the inextricable interaction of software product and software process quality.

Based on Kitchenham's observation, the ISO 9001 (1994) definition for quality and my philosophy of providing for generalisation and specialisation/customisation, the following definition of software quality is proposed:

Software Quality is the totality of product as well as process characteristics, and their interaction and measurement (whether qualitative or quantitative) that satisfy different stakeholder requirements.

Software products or artifacts are also known as the deliverables or outputs of the software process. These products may be plans, functional specifications, process models, procedure manuals, coding, test data, test results and so on (Whitmire 1997). In the early years the Software Engineering community adopted an end-of-cycle quality inspection regime just like the early manufacturers who inspected finished products. Inspections in turn resulted in three categories of finished product namely the accepted, the rejected and those products requiring rework. The last two 'heaps' namely the *rejects* and the *reworks* gave a measure of the losses which every manufacturer needed to reduce for survival and competitive advantage (Gilb & Graham, 1993), (Burr and Georgiadou 1995).

Software process is a set of activities that begin with the identification of a need and conclude with the retirement of a product that satisfies the need; or more completely, as a set of activities, methods, practices, and transformations that people use to develop and maintain software and its associated products (e.g. project plans, design documents, code, test cases, user manuals). Pfleeger (1998) emphasizes that "we must learn how to use software process to enhance products without stifling creativity and flexibility. We must also learn which processes work best in which situations, and understand what characteristics of the products and of the people building them are the most important in process choice".

Resources include people, tools, materials, methods, time, money, training (or generally knowledge and skill) and products from other projects (Whitmire, 1997). In essence resources are the inputs to the processes used on a project. Resource usage together with size measures allow productivity to be measured (Kitchenham 1996).

A software project is the relationship between instances of a problem to be solved, internal and external goals and standards, processes, methods and techniques, constraints and finally a product (one or more deliverables) (Whitmire, 1997). The goals and constraints particularly on resources affect the outcome, the nature of the product and the success or failure of the whole project. Therefore the study of failed projects is complex but necessary if the software engineering community is to address, rectify and eventually prevent future failures.

Software metrics: Fenton and Neil (1999) asserted that 'Software metrics' "is the rather misleading collective term used to describe the wide range of activities concerned with measurement in software engineering". A *software metric* is a standard of a measure of a degree to which a software system or process possesses some property. Even if a metric is not a measurement (metrics are functions, while measurements are the numbers obtained by the application of metrics), often the two terms are used as synonyms.

According to IEEE 1061 (1998) "Software metrics measure properties of software and are loosely defined as functions whose inputs are software data and whose output are single numerical values that can be interpreted as the degree to which software possesses a given attribute that affects its quality". A standard way of measuring some attribute of software is known as a metric. According to ISO 9126 (2001) and the ISO/IEC 25010 (2014) a software quality metric is a quantitative scale and method that can be used to determine the value which a feature takes for a specific software product.
The definition used in this research is *"A software metric is a measurable property which is an indicator of one or more of the quality attributes."*

1.3 Attempts to address the Software Crisis

For nearly 50 years the Software Engineering community has recognised the necessity to address systems and project failures which came to be known as the Software Crisis (Sommerville, 2010); (Standish Group, CHAOS Report, 2015); (Dalcher, 2017).

[Britain: The health service's IT problem; Computerising the NHS] and [The Economist, Oct 19, 2002, Vol. 365(8295), pp.51-52].

Researchers and practitioners endeavoured to identify methods for improving the productivity and the quality of product. Due mainly to lack of awareness and the strong desire to achieve these aims, myths and often unsubstantiated claims plagued the industry. It was not long ago that program generators were hailed as the answer to the problem of late deliveries and therefore high costs. New languages were often believed to have almost 'magical' powers of resolving the crisis. Automated tools, formal methods, object-orientated methods have been proposed as alternative 'religions' with Software Engineers becoming almost fanatical followers of one or the other approach (Georgiadou & Sadler, 1995), (Siakas et al 1997).

This quest resulted in the development of numerous approaches to building and maintaining software, from new technologies to progressive processes and frameworks as reported by among others the various CHAOS reports (1994 -2015) by the Standish Group, Jackson (1995), Jayaratna (1994), Beck (1999), Avison & Fitzgerald (2003), (Eveleens and Verhoef, (2010), Vandierendonck & Mens (2011), and Dalcher (2017).

Although the CHAOS reports have come under some criticism e.g. by Eveleens and Verhoef (2010) they nevertheless provide a benchmark for further study as despite the many developments such as the adoption of agile methods the rates of failures have not improved significantly.

This research sought to gain increasingly deeper insights into the ways in which the Software Engineering community has attempted to address the software crisis by examining the role of Software Development Lifecycles, Information Systems Development Methodologies, and Quality models which researchers and practitioners proposed, developed and used over the last 50 years. A historical perspective was adopted tracing major developments and their contribution to this endless quest for improvement.

1.4 Learning from the Manufacturing Industry

Already
The automobile manufacturer's eighth model
Reposes on top of the scrap iron
But we
Are travelling in the ninth
Thus we have decided
In ever new vehicles - full of flaws
Instantly destructible
Light, fragile
Innumerable -
Henceforward to travel.

(from Bertolt Brecht : The Impact of the Cities 1925-1928
Still, when the automobile manufacturer's eighth model)

The Japanese approach to quality control was initiated by Shewhart in 1939 (Shewhart W A. Statistical method from the viewpoint of quality control. Mineola, NY: Dover Publications, 1986 (1939) and continued by Deming (1986). Shewhart wrote *Statistical Method from the Viewpoint of Quality Control* and gained recognition in the statistical community. Logothetis and Wynn (1989) also reported that Kaoru Ishikawa who is considered the father of 'Total Quality Control' and who received the 'Deming Prize', advocated that Statistics should become a common language that can be used at all levels in the organisation providing the information to anticipate, identify and correct mistakes, and also used to reduce wasteful variability in the system by 'doing it right the first time'.

Hagime Karatsu (cited by Logothetis & Wynn, 1989) explained the benefits from a high quality manufacturing process as follows: *"If it is aimed to produce quality products, there will be great financial benefits. Withdrawal and return of products are reduced. Higher productivity will be achieved because the necessity to stop machines in order to replace materials will be less frequent. This means it is possible to reduce the operation rate. As the manufacturing system itself improves in quality, the cost will be minimised. That will give rise to the company's reputation and will increase its sales."*

Deming is regarded as the founder of the third wave of the Industrial Revolution. He claimed that if a company tries to obtain short term profit, it would lead to business failure, and suggested ceasing dependence on inspection in order to achieve quality, and eliminating the need for mass inspection by building quality into the product in the first place. This emphasises the importance of the stage of design of a product, which is common from Deming to Taguchi.

Taguchi's philosophy began taking shape in the early 1950s when he was recruited to help improve the postwar Japan's crippled telephone system. Finding deficiencies in traditional trial-and-error approaches to identifying design problems, he eventually developed his own complete, integrated methodology for designing experiments (American Supplier Institute, 1999) for process and product improvements.

Taguchi's philosophy is now well practiced in the manufacturing industry. In Japan the Taguchi Method (1985; 1986) is called 'hinshitsu kougaku' which translates to 'quality engineering'. The method has ensured the significant reduction of manufacturing costs together with increased product quality (Barbor & Georgiadou, 2001). Taguchi also spoke of social loss. If quality is high, society will get benefit from the product. If quality is low, society's current standards will decrease to cope with those deficient products. Therefore, it is desirable to strive for smaller social loss. The term 'social loss' (Logothetis & Wynn, 1989) implies:

- losses due to poor and varied performance of a product;
- failure to meet the customer's requirements of fitness for use or for prompt delivery;
- harmful side-effects caused by the product.

Examples of notorious failures that caused exorbitant financial losses but also loss of life include the space shuttle Challenger Disaster in 1986 (Dennis S. G. et al., 1986), the London Ambulances Dispatch Service failure in 1992, Beynon-Davies, P., (1999) The London ambulance dispatch system was cancelled in 1990 at 11.25 million pounds; the second attempt was abandoned after deployment costing another costing another 15 million pounds!

Business examples include the total failure of the London Stock Exchange – Taurus, (Charette, 2005), and the fiasco of the BA Terminal 5 Luggage Handling system (Winston, 2008).

Lessons that can be learned from the manufacturing industry are twofold: firstly the need to built-in quality at the early stages of the development process, avoiding product testing at the end of the lifecycle, and secondly the need to use techniques such as Statistical Process Control to monitor the quality of the process and hence the products.

In the case of Software Engineering considerable effort was expended in carrying out software testing namely exercising finished code with 'suitable' test data. Despite these efforts systems continued to fail and the Software Engineering community had to seek alternative or complementary methods for minimising the losses in terms of financial costs and loss of human life.

Maintenance of existing software is by far the most frequent, demanding and expensive process in software engineering. Legacy systems require constant maintenance: corrective to address failures in use, adaptive to accommodate changes in requirements and perfective to improve performance (Lehman, 1974; Sommerville, 2001; Pressman, et al., 2015). Modifying software is an integral part in the lifecycle of systems. Maintainability, the ease with which maintenance activities are carried out, impacts on productivity and costs. Through the study of the behaviour of existing systems it is aimed to help decide whether a failing system should be scrapped or improved by carrying out adaptive, corrective and perfective maintenance.

1.5 Motivation of the Researcher

Reliability, Functionality, Usability, Portability, Efficiency, Maintainability are aspects of quality which, when deficient, can cause failures resulting in disasters and financial losses. All aspects of life nowadays depend on complex and interconnected systems: transport, health, the economy, government, education. Managers need accurate information and guidelines to help them make important decisions, plan and schedule activities to allocate resources for the different software activities that take place during software development. Therefore understanding and controlling the process is imperative if we are going to enhance the quality of the product and the success of project(s).

1.6 Research Scope and Research Questions

This investigation is multifaceted. It views Software Quality from different perspectives. Different stakeholders have a different understanding of software quality and hence different requirements. Their requirements and opinions may be synergistic or conflicting. The broad topic therefore covers a big range of issues relating to systems and project failures and the efforts of the community (both theoreticians and practitioners) engage in a continuous quest for identifying and resolving the issues.

According to Lehman's first law software systems must be continually adapted, or they become progressively less satisfactory (Lehman, 1980), (Lehman, et al., 1997). At the same time, software is becoming more and more complex and expensive than before. "As a software system evolves, its complexity increases unless work is done to maintain or reduce it". Lehman (1998) observed that systems continue to evolve over time. As they evolve, they grow more complex unless some action, such as code refactoring, is taken to reduce the complexity. In the late 1970s, a widely cited survey study by Lientz and Swanson (1981) established that maintenance and enhancement of software consume between 75% and 80% of the lifecycle cost. They categorised maintenance activities into four classes:

- Adaptive – modifying the system to cope with changes in the software environment
- Perfective – implementing new or changed user requirements which concern functional enhancements to the software
- Corrective – diagnosing and fixing errors, possibly ones found by users
- Preventive – increasing software maintainability or reliability to prevent problems in the future.

This research combines an investigation and implementation of Product Improvement, (and primarily on software maintenance of existing (legacy) systems), and investigation of Process Improvement methods, models and practices, and interplay of Product and Process Improvement.

1.7 Focussing on Product Improvement

The manifestations of what came to be known as the Software Crisis demand quality improvements to rectify errors and even catastrophic failures. Additionally even if systems are not failing, they inevitably require maintenance because of changing technologies or changing requirements. Therefore a major challenge is how to deal with legacy systems. The question is *"**whether to develop new solutions from scratch or to re-engineer systems?**"* This investigation addresses this dilemma and seeks to establish a rigorous methods and associated measures for supporting management criteria for deciding whether code restructuring is advisable. As the area is broad it is necessary to narrow the scope and focus on the topic as shown in Figure 1.1.

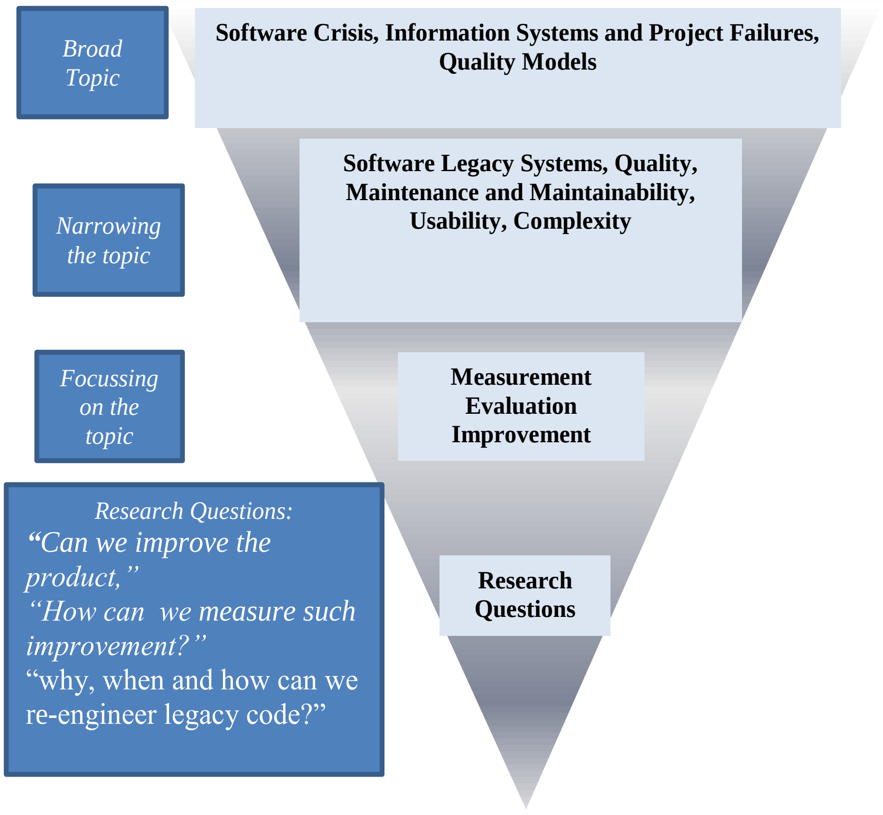
Broad Topic
Software Crisis, Information Systems and Project Failures, Quality Models
Narrowing the topic
Software Legacy Systems, Quality, Maintenance and Maintainability, Usability, Complexity
Focussing on the topic
Measurement Evaluation Improvement
Research Questions: "Can we improve the product," "How can we measure such improvement?" "why, when and how can we re-engineer legacy code?"
Research Questions

Figure 1.1: Part 1 research questions funnel (adapted from Chilakanti, 2013)

Software development has shifted from corrective to preventive methods and quality improvements have shifted from product improvement to process improvement emulating the manufacturing industry.   Preventive methods in all fields of human activity (such as preventive health care) are preferable and in the long run less costly.
Shifting the effort to process improvement relies on the assumption that an improved process is likely to result in improved outputs (products) (Shewhart, 1986); (Deming, 1986); (Taguchi, 1985 and 1986).

1.8 Focussing on Process Improvement

Process improvement initiatives in other industries as well as in the Software Engineering industry have argued and demonstrated that process improvement enhances the chances of product improvement (Kitchenham, 1995, Burr and Georgiadou, 1995, Pleeger-Lawrence, 1996).

Thus product improvement should not be separated from process improvement as the two are interconnected.
Considering the research from a process improvement approach we start from the broad topic of Total Quality Management and Continuous Process Improvement to proceed to Process Maturity, Knowledge Sharing and Performance Measurement. This is the second Pillar of this research.

Figure 1.2 depicts additional research questions funnel.

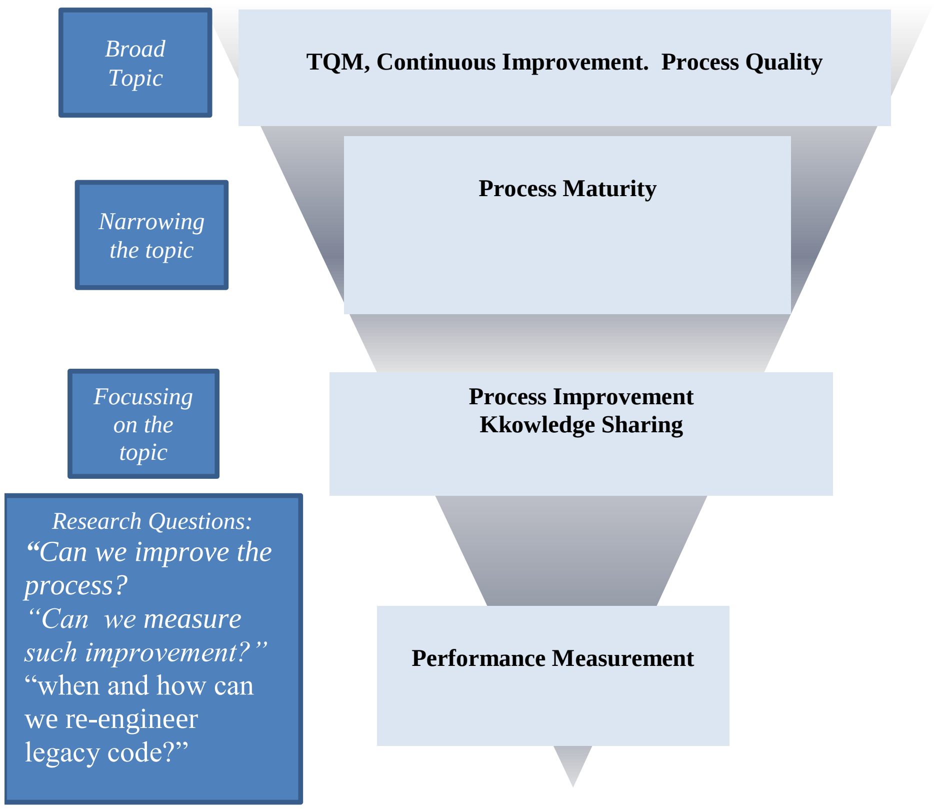
Broad Topic
TQM, Continuous Improvement. Process Quality
Narrowing the topic
Process Maturity
Focussing on the topic
Process Improvement Kkowledge Sharing
Research Questions: "Can we improve the process? "Can we measure such improvement?" "when and how can we re-engineer legacy code?"
Performance Measurement

Figure 1.2**: The Pillar 2 research questions funnel (adapted from Chilakanti, 2013)**

The Research Questions (RQs) were:

RQ1 ***"Can we improve the quality of software by manipulating its structure, and, if the answer is yes, can we measure such improvement?"***

RQ2 ***"When and how is it preferable to re-structure existing code or develop from scratch?"***

RQ3 ***"if we can improve the process, can we measure such improvement?"***

In order to answer these questions it was decided to:

- identify and study the characteristics of software and its behaviour with the view to bring about improvements through manipulation (restructuring, reengineering, refactoring) of legacy software;
- investigate methods and models proposed for software process improvement;
- develop estimates, measures, models and frameworks for the achievement of improvements.

Software Quality has many characteristics both structural and behavioural. The focus of this study was covering the nexus of software product improvement (especially re-engineering of legacy code) and software process improvement, and the measurement of improvements as a means of substantiating and validating propositions, models, and results.

1.9 Research Objectives and Hypotheses

1.9.1 The objectives of this research were to:

- investigate methods and practices of addressing systems failures;
- identify software characteristics and their measures of the software product & manipulate/re-structure software and study empirically the effects of such interventions on behavioural characteristics (such as maintainability and usability);
- propose suitable re-engineering software product measures through the study of dependencies of internal characteristics and external characteristics of software;
- identify the factors and characteristics of the software process & develop process improvement and measurement mechanisms;
- construct an adaptable quality model suitable for different stakeholders (such as users, developers, sponsors) & develop a framework for the evaluation of items under selection.

1.9.2 Research Assumptions

Assumption 1: Process Improvement impacts directly on Product Improvement.
Assumption 2: Process Improvement depends on Maturity level and Knowledge Sharing.

1.8.3 The research hypotheses

H1: Structural software product characteristics impact on its behaviour

H2: The usability of software depends on design characteristics

In this investigation the product characteristics selected are maintainability and understandability.

Thus H1 is recast

H1.1 The maintainability of software depends on its complexity.
(i.e. Low module complexity results in high maintainability).

H1.2 The maintainability of software depends on its structuredness
(i.e. Highly structured programs are highly maintainable).

And H2 is recast as

H2.1 The understandability of software depends on the use of colour in the Human Computer Interface (HCI).

H2.2 The understandability of software depends on the use of non-symbolic naming of variables.

1.10 Summary of Chapter 1

The context of this research was the widespread problem of Information Systems failures and the manifestations of what has come to be known as the *Software Crisis*. Studying the attempts to address the crisis allowed for the focus and motivation to be expressed. The research rested on three pillars:

- Product Improvement (particularly Maintainability and Usability) of legacy code,
- Process Improvement which was expected to result in Product Improvement, and
- The connection between Product and Process Improvement and the measurability of Improvement.

The aims and objectives were specified, and research hypotheses were formulated.

CHAPTER 2

RESEARCH PHILOSOPHY, RISEARCH METHODS AND RESEARCH METHODS USED IN THIS RESEARCH

Disciple: "Rabbi, why do you answer a question with another question?"
Rabbi: "Is there another way?"   (Jewish Proverb)

Chapter 1 is the **Introduction** which presents the motivation, context, scope, objectives, formulation of research hypotheses and research methods employed.

2.1 Research Philosophies

The research questions posed here demand scientific proof of generally held beliefs, which often mark the beginning of many investigations. ***Doxai*** represent what is believed to be true whilst epistemic investigation results in something ***known to be true***. The purpose of science, then, is the process of transforming things *believed* into things *know*n i.e. transformation of ***doxa to episteme;*** δόξα **[dóxa]** : Greek term for opinion, belief, or judgment, as opposed to *επιστήμη*[epistêmê] Greek term for systematic knowledge.

Two major research philosophies have been identified in the tradition of science, namely positivist (sometimes called scientific), and interpretivist (also known as anti-positivist) (Galliers, 1992). According to Checkland (1981) scientific (empirical) approaches have: Repeatability, Reductionism and Refutability.

2.1.1 Positivism

Positivists believe that reality is stable and can be observed and described from an objective viewpoint (Sarkar, 1996) without interfering with the phenomena being studied. Positivists seek to isolate and observe phenomena. Positivism requires repeatability, and involves manipulation of "a single independent variable so as to identify regularities in, and to form relationships between, some of the constituent elements of the social world." Previous observations, explained realities and inter-relationships form the basis of predictions.

In the case of Software Engineering and Information Systems the factors involved in their development and operation are either not measurable or extremely difficult to measure. Conversely, considerable debate on whether positivism is a suitable paradigm for social science research is reported in, among others, (Hirscheim, 1982) and (Remenyi & Williams, 1996).

2.1.2 Interpretivism and the multi-paradigmatic approach

The positivist paradigm has been widely used as reported by researchers like Orlikowski and Baroudi (1991) in Information Systems research. A combination of research methods is preferable in order to improve the quality of research. The constructivist approach is concerned with developing frameworks, refining concepts or pursuing technical developments. The approach allows models and frameworks to be created that do not describe any existing reality or do not necessarily have any "physical" realisation (Cornford and Smithson 1996).

Lee (1991) developed a framework which can be used in order to combine qualitative and quantitative research in a study. Lee's framework is known as a multi-paradigmatic approach which identifies three levels of understanding as follows:
First level: subjective understanding belongs to the observed human objects. This understanding is the making of sense of everyday behaviour which manifests itself in social settings.
Second level: interpretive understanding belongs to the observer (researcher). This understanding is the reading or interpretation of the first level, common sense understanding.
Third level: positivistic understanding belongs to the researcher. This understanding involves the researcher creating and testing propositions in order to explain the empirical reality that he/she is investigating.

Lee (1991) suggests that research methods such as case study, action research and grounded theory can be used to develop the researcher’s second level understanding which in turn helps develop testable propositions addressing the social phenomena under investigation. This in essence constitutes the third level of understanding. A multi-paradigmatic approach can provide a basis for developing testable propositions.

2.2 Research Methodology Selection and Justification

This research is concerned with a search for, and evidence to support general laws or theories that will cover a whole class of cases. Such research emphasises systematic protocols and hypothesis testing within the scientific tradition and is known as ***nomothetic research (***Cornford and Smithson, 1996). Galliers (1992) summarised the main research methods in Information Systems.

Formal laboratory controlled experiments are primarily used for testing a hypothesis to establish the confidence with which you may predict the implications of a particular theory.

Laboratory experiments permit the researcher to identify precise relationships between a small number of variables that are studied intensively via a designed laboratory situation using quantitative analytical techniques with a view to making generalisable statements applicable to real-life situations.

The key weakness of laboratory experiments is the "limited extent to which identified relationships exist in the real world due to oversimplification of the experimental situation and the isolation of such situations from most of the variables that are found in the real world" (Galliers, 1992, p.150). However, as it is usually impractical to carry out formal experiments within industry due to the heavy demand on resources researchers frequently carry out experiments within academia.

Surveys enable the researcher to obtain data about practices, situations or views at one point in time through questionnaires or interviews. Quantitative analytical techniques are then used to draw inferences from this data regarding existing relationships. The use of surveys permits a researcher to study more variables at one time than is typically possible in laboratory or field experiments, whilst data can be collected about real world environments.

Case studies involve an attempt to describe relationships that exist in reality, very often in a single organisation. Case studies may be positivist or interpretivist in nature, depending on the approach of the researcher, the data collected and the analytical techniques employed. Reality can be captured in greater detail by an observer-researcher, with the analysis of more variables than is typically possible in experimental and survey research.

Model Development (MD) is an effective research method. MD assists researchers and scientists to describe, understand, predict, and test complex systems or events. *"Models can consist of actual objects or abstract forms, such as sketches, mathematical formulas, or diagrams. A model is an abstraction, a mental framework for analysis of a system" (Busha & Harter, 1980).* Although models can be an oversimplification and overgeneralisation their simplicity aids understanding of what and/or who are involved and of when and in what sequence processes take place.

The research reported here was based on the three levels of Lee's multi- paradigmatic framework provided understanding which is the making of sense (level 1), followed by interpretive understanding (level 2) , and the third level (positivist understanding) which enabled the formulation and testing of propositions and hypotheses.

In order to achieve the research aims and objectives, this research adopted a multi-paradigmatic method combining literature review (historical and underpinned by an interpretive stance), formal controlled experiments, qualitative analysis, quantitative evaluations, qualitative surveys, expert opinion, and model development.

2.3 Research Methods Used

Table 2.1 shows the research methods used and the main contributions under development of: taxonomy, indicator, model, or framework.

2.4 Summary of Chapter 2

This chapter presented the scope and focus of the research, the research objectives and assumptions. Finally the formulated research hypotheses and the research methods were presented.

Table 2.1a - Research Methods Used

PAPER	Literature Review, interpretivism & argumentation	Survey for validation & verification	Case studies	Formal, Controlled Experiment	Qualitative analysis , categorisation	Quantitative Measurement & Evaluation	Model Development: Taxonomy Indicator, Model, Framework.
I.	✓				✓		Indicator
II.	✓				✓		Taxonomy
III.	✓						Model extension
IV.	✓		✓		✓	✓	Feature Analysis, CFD: Composite Features Diagram GEQUAMO Model
V.	✓	✓	✓			✓	GEQUAMO II Model

Table 2.1b - Research Methods Used

VI.	✓				✓		
VII.	✓				✓		
VIII.	✓						Model extension
IX	✓			✓			
X	✓			✓			
XI	✓			✓			
XII	✓			✓			
XIII	✓						Framework for experimental design and execution

Table 2.1c - Research Methods Used

XIV	✓			✓			I^{5}PModel
XV	✓			✓			VALO$_5$ Model
XVI	✓	✓		✓			CoFeD framework
XVII	✓						
XVIII	✓	✓					Guidelines
XIX	✓				✓		
XX	✓				✓		Framework

CHAPTER 3

INFORMATION SYSTEMS DEVELOMENT METHODOLOGIES, LIFECYCLES, AND QUALITY MODELS

3.0 Introduction

This chapter presents a critical review of lifecycle models, information systems development methodologies, process models, standards, and quality models, and the development and validation of the generic, multilayered, and customisable software quality model GEQUAMO.

3.1 Information Systems Development Methodologies

Information Systems Development Methodologies (ISDMs) and associated tools aim to a systematic planning and control of the development process. Systems Development Methodologies have been proposed and used to address a number of problems including ambiguous user requirements, un-ambitious systems design, unmet deadlines, budgets exceeded, poor quality software with numerous 'bugs' and poor or non-existent documentation. This meant that software was difficult to maintain, and inflexible to future changes.

It is important to note the erroneous use of the terms method and methodology as synonyms by both researchers and practitioners in the Information Systems and Software Engineering domains. Kerry Howell (2013), in her book "Introduction to the Philosophy of Methodology" clarifies that *"a methodology is the rationale for the research approach, and the lens through which the analysis occurs....A method is simply a tool used to answer your research questions—how, in short, you will go about collecting your data. Examples of research methods include: Contextual inquiry, Interview, Survey etc. The methodology should impact which method(s) for a research endeavour are selected in order to generate the compelling data."*

The use of method and methodology as synonyms has been perpetuated for over 40 years. Evidence of this issue can be found in many articles and acclaimed textbooks such as Jayaratna (1994) and Avison and Fitzgerald (2003). At this stage we will not challenge this ambiguity but we put a marker for subsequent investigation and action towards addressing such ambiguities. As the community continues to refer to methods as methodologies we did the same (see **Papers I and II**).

By applying a methodology to the development of software insights are gained into the problems under consideration and thus, these problems can be addressed more systematically. Methodologies provide the environment for repeatable procedures with specified deliverables at each stage of the system lifecycle. Software should comply with the important quality requirements of Timeliness, Relevance, Accuracy and Cost Effectiveness (TRACE)[1]Software Engineering aims to bring to bear the more rigorous methods used in the engineering world in the software development world.

Paper I (Georgiadou and Sadler, 1995) outlines the main characteristics and philosophy of the prevalent methodologies. Over 2000 methodologies (and brand names) are in existence (Jayaratna, 1994), each claiming to solve most, if not all, of the problems of systems development (Jackson 1994), (Berki et al, 1997). Since the 70s literally hundreds of different methods and tools have appeared each claiming to ease the life of the developer and the user by achieving improved productivity without compromising the quality of the software product.

These methodologies range from integrated collections of procedures to a single technique, notations, 4GLs and tools for supporting the process at the various stages of the systems lifecycle. Figure 3.1 depicts the approximate time of their introduction.

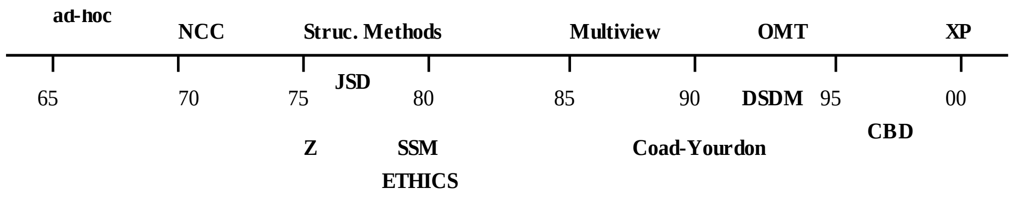

Figure 3.1 Information Systems Development Methodologies over 50 years

[1] TRACE is a term coined by the author (UNL Lecture Notes SDM 1993)

With the exception of Formal Methods (such as Z and VDM) which are not within the scope of this research all methodologies aspire to help organize the process of information systems development.

Using the techniques and tools of a methodology we can understand a problem situation by using abstraction, generalisation, classification and specialisation to model a system. The resulting models elucidate the functionality of the system and introduce a degree of structure and rigour enabling us to produce solutions satisfying the identified requirements.

3.2 A Taxonomy of Methodologies

The level of user participation is inherent in each methodology. Participative methodologies like SSM, ETHICS and Agile Methods such as XP, place a strong emphasis on the managerial and social issues and, usually, devote extra time and effort on preventative actions such as liaising with stakeholders and users in particular, during the early stages of the systems lifecycle. This shapes the development process and the type of solution(s) achieved. User involvement and user participation takes many forms such as discussions on requirements, walkthroughs, reviews, inspections, and validations.

Although there are thousands of information systems methodologies (brand names) they tend to form clusters or families with general characteristics dictated by their philosophy and type of problems they attempt to solve. Thus they have been divided into data-driven or process driven, hard and soft, large and small depending on the viewpoint of the researchers studying the whole area of methodologies.

Avison and Fitzgerald (1995) proposed a classification of ISDMs into Soft and Hard with the addition of Hybrid Methodologies. **Paper I** presents an enhanced classification by the addition of Agile Methods (XP), Formal Methods and Specialised Methods, shown in Figure 3.2.

This classification was used to underpin further investigations into requirements engineering and quality models (Siakas et al., 1997), (Berki et al., 1997).

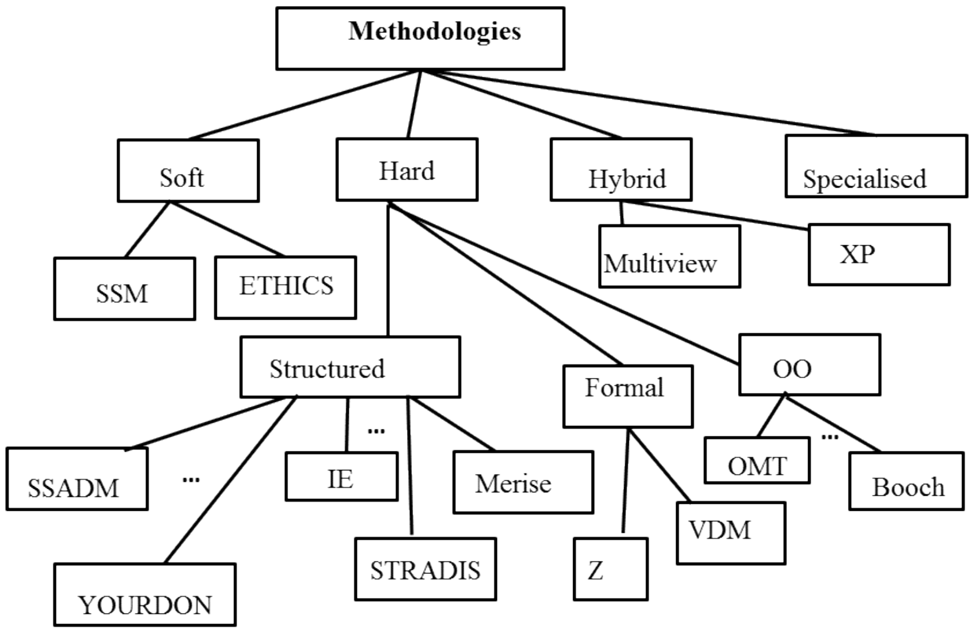

Figure 3.2 Methodology Classification Tree
[Adapted from Georgiadou &Sadler, 1995]

Information systems practitioners (developers) are mainly interested in selecting a methodology that is appropriate for their particular environment, capable to address their problems and finally enable them to enhance their productivity. It is thus important to develop ways of evaluating and selecting suitable methodologies according to requirements. *"Making the wrong choice of methods and tools can be a big risk to an organisation (and to those who made the choice), because the acquisition and installation costs can be very high and the consequential costs of project disruption and delay can also be very high"* (Law & Naeem, 1992). The choice of an appropriate method has been of concern to industry and academia alike. Making the wrong choice of method or tool can be very costly for a company. How are software engineers to succeed in making these choices?

3.3 Evolution of the Software Development Lifecycle

In addition to methodologies, the lifecycle of systems development has been depicted in numerous lifecycle models. Tom Gilb (1981), and Naumann and Jenkins (1982) were already referring to previously practiced methods as traditional and that new evolutionary methods were necessary and preferable.

The involvement of the user is an integral part of the prototyping paradigm. The cycle starts with the requirements gathering and goes through a number of refinements. Prototyping is a mechanism for clarifying the requirements and making improvements (Figure 3.3).

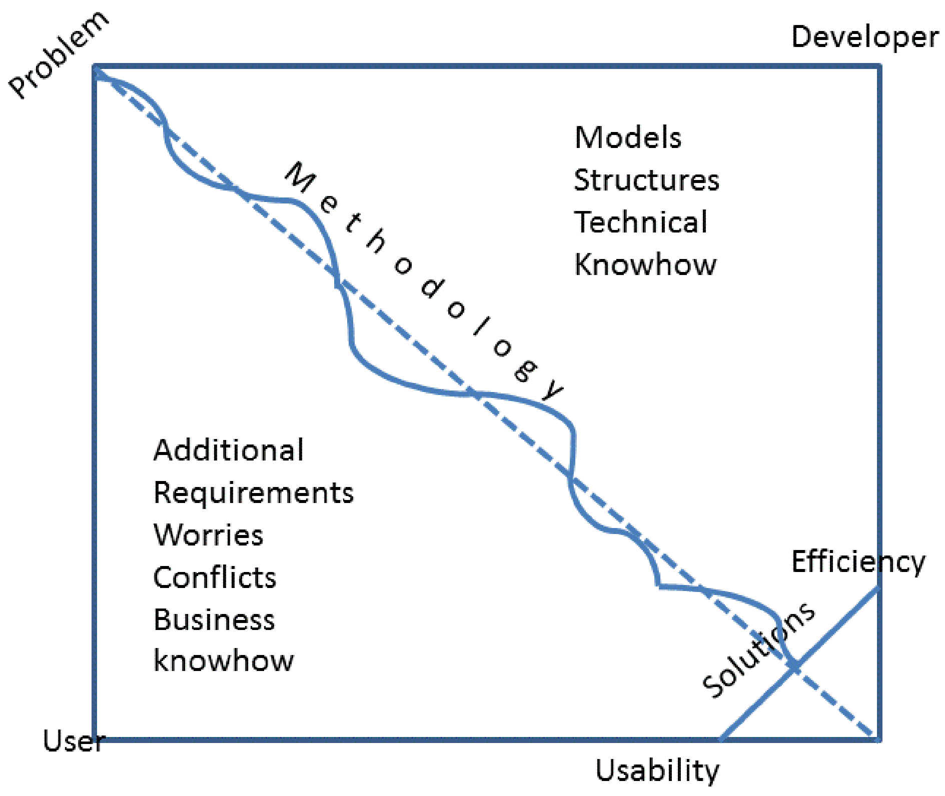

Figure 3.3 Prototyping: The 'Balanced' interaction (Paper II)

Paper II concluded that the prototyping development route is characterised by constant feedback from the users and team, collaborative development tending to achieve near-perfect solutions, trading off technical effectiveness to usability as shown on the solutions space. The whole system of transformation from problem to solution can be likened to a group of musicians playing their instruments and making music. The musicians are the systems developers, the instruments are the methods and the music the solutions produced.

It was argued **(in Paper II)** that **t**he role of the methodology as an instrument of understanding is provided by its underlying lifecycle namely the organised phases, steps, tasks and checklists which steer the developer towards identifying and specifying the components, sub-systems and their interactions. Therefore in trying to choose and apply a methodology valuable insights are gained into the original problem and the existing procedures, highlighting problem areas and additional requirements.

Software Lifecycles models are paradigms for guiding the development process and for aiding the planning, monitoring and controlling of projects. Hence lifecycles are process models with phases and deliverables at each phase. The nature of the problem, the methods and tools, the controls and the deliverables formulate the paradigm. Figure 3.4 shows the approximate time of introduction of the major software lifecycle models since 1970.

In **Paper II** a historical perspective was adopted to look at the major developments and introductions of lifecycle models, methodologies (revisited), and quality models. Linear, incremental, cyclic, Object-Oriented, and fractal models were discussed. Efforts for process improvement and particularly Continuous Improvement through Total Quality Management as well as Quality Models were presented and discussed.

Cyclic models such as the Spiral Model reflect the iterative nature of software development. Prototyping was primarily an iterative model. Models like the V, W, X include embedded quality assurance mechanisms through the development process emhpasising that code is not the only thing that needs ‘testing’. Object-oriented models came with the promise of development with re-use in mind.

For several years the community was engaged in corrective measures. Linear models, like the Waterfall model, placed Testing towards the end of the lifecycle which results in late discoveries of errors and omissions when it often proved to be very late and extremely expensive. This study revealed that as the lifecycle development itself became more mature, testing has been ‘moving’ towards the beginning of the lifecycle. Additionally other quality assurance techniques such as walkthroughs, inspections, reviews, design reviews etc., were used throughout the lifecycle giving many opportunities for early detection of errors and omissions. These are steps towards preventive quality assurance. With the introduction of agile methods such as Extreme Programming it was observed that Testing had moved to the beginning of the lifecycle as predicted in 1995 (Georgiadou and Sadler, 1995). .

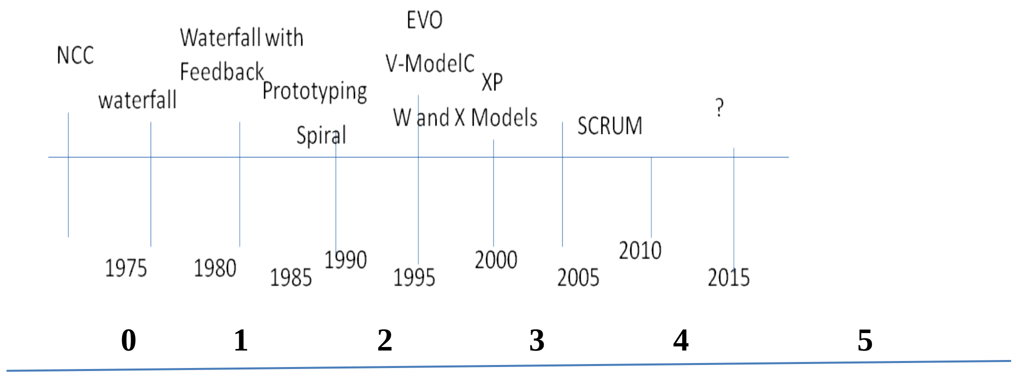

Figure 3.4 Growth of Lifecycle Models and Capability Maturity

The juxtaposition of the lifecycle timeline and the Capability Maturity level depicts the maturity growth of the lifecycle. The capability maturity of the process is expressed in recent years by the development and adoption of iterative and agile methods.

In the last 10 years studies on Agile Methods show increasing take up by the industry. In 2014 del Águila et al. (2014) provided a comparative review of Software Engineering (SE) and Knowledge Engineering (KE). They noted that these two Engineering Disciplines have been evolving in parallel but have not been learning from each other. They (del Aguila et al.) proposed an integration of the two, named SKEngineering, which "*allows the development of quality products using SE or KE methods, since there are many cases in which companies require deploying software systems that integrate components based and not based on knowledge in a transparent way.*" They additionally suggest that SKEngineering can be assisted by well-known Artificial Intelligence techniques as machine learning or fuzzy approaches.

In recent years there has been increasing focus on Model Driven Engineering (MDE). Clark et al. (2016) advised that MDE should be used for Model Driven Organisations (MDO) because modelling can provide a more scientific tractable alternative instead. Kulkani (2016) reports on the successful adoption of MDE within the large organisation TATA.

Information Systems Development Methodologies (ISDMs) have been changing, evolving and adapting to other changes in the field. As Software Engineering is maturing so do the methods, techniques and tools evolve to meet new requirements and challenges. However, systems failures persist.

3.4 Software Quality Models

In the last 40 years various product quality models have been developed. Software Quality Models have primarily been based on Top Down, hierarchical structures the most notable of which are: the Dromey model (Dromey, 1995); (Hyatt & Rosenberg, 1996), the McCall model (McCall et al., 1977); the Boehm model (Boehm et al., 1978); the FURPS Model (Grady &Caswell, 1987); and the ISO 9126-1 model (2001); also its standards for both external metrics: ISO / IEC 9126-2 and internal metrics: ISO / IEC 9126-3 in 2003 and quality in use: ISO / IEC 9126-4 in 2004. The ISO -9126 model incorporated various aspects from previous models and standards for assessing, controlling and measuring the quality of software.

These models suffer from lack of orthogonality between the quality factors and their sub-factor or constructs. For example the Boehm quality model proposes definitions and measures for a range of software quality attributes. It focuses on software quality form the developer's perspective and divides quality into 7 quality factors (intermediate constructs) namely Portability, Reliability, Efficiency, Human Engineering, Testability, Understandability, and Modifiability. Each of these intermediate constructs is further divided into primitive constructs It can be seen that the 7 top level quality factors are not orthogonal. For example reliability and human engineering share a common primitive construct of Robustness/Integrity. Therefore, this lack of orthogonality presents difficulties when considering quantification and evaluation

The ISO 9126 standard sets out a strict framework for designing an evaluation of quality characteristics. Six characteristics are specified and decomposed into several sub-characteristics. It is the only model that observes orthogonality – each sub-characteristic relates to only one characteristic.

In **Paper III** the major quality models were discussed, and ISO 9126 was judged to be *"a widely applicable and accepted software quality model for software quality engineering"*.

However, none of these models are customisable according to the different stakeholders and their specific requirements. Since the turn of the millennium the tendency has been to customise quality models. Another weakness of these quality models is that they rarely specify how quality attributes should be measured and how measurement results can be aggregated to achieve an overall quality assessment for a system.

3.5 GEQUAMO – A generic, multi-layered and customisable Quality Model

In **Paper IV** the GEQUAMO (**GE**neric, multi-layered and customisable) QUAlity **MO**del was proposed. GEQUAMO encapsulates the requirements of different stakeholders in a dynamic and flexible manner so as to enable each stakeholder (developer, user or sponsor) to construct their own model reflecting the emphasis/weighting for each attribute/requirement. Using a combination of the CFD (Composite Features Diagramming Technique) developed by the author, and Kiviat diagrams a multi-layered and dynamic model is constructed.

A generic CFD shown in Figure 3.5 each node is exploded into two, three or more sub-characteristics. The proposed Quality Model uses the CFD- Composite Features Diagram comprising a set of concentric circles showing increasingly lower details (sub-characteristics). The lowest branches in every case represent sub-characteristics which are measurable and hence controllable. It can be seen that the number of branches (siblings) emanating from each node can be 0, 2, 3 or more. CFDs can be used for generating visual profiles for items under comparison for the purpose of selection. R1, R2, R3 are the requirements of a specific stakeholder. Each tree represents decomposition into lower layers. The outer branches are usually measurable directly.

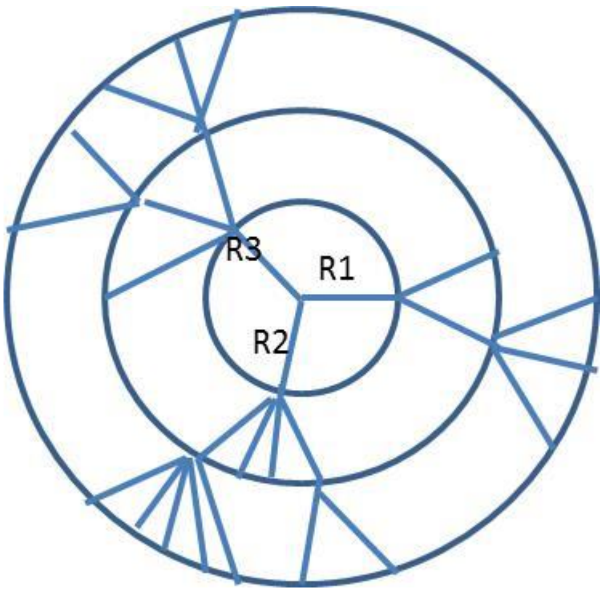

Figure 3.5 – A generic CFD- Composite Features Diagram

In **Paper IV,** the genericity and versatility of GEQUAMO were demonstrated through its application to various evaluation cases. Also the algorithm for top-down qualitative identification and decomposition of the requirements (features, sub-features, sub-subfeatures), and the bottom-up quantitative evaluation were presented. The construction rules do not allow one sub-characteristic, and it is also advisable that the maximum number of sub-characteristics should not exceed 7.

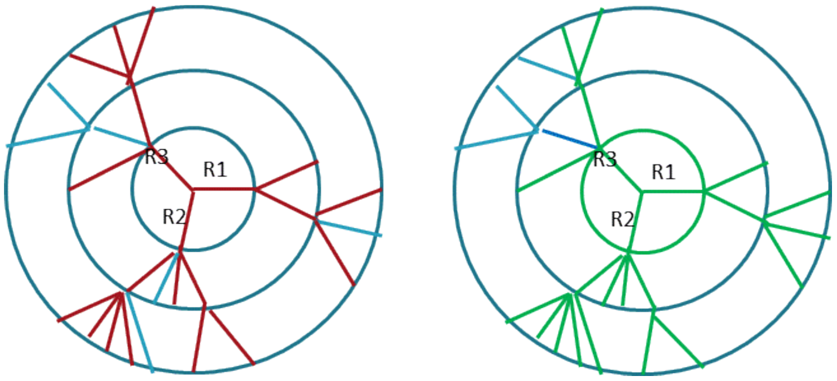

Figure 3.6 – A Visual Comparison between two items using CFDs

It can be seen in Figure 3.6 even before the numerical (bottom up evaluation) that the ítem profiled on the left (in red) has more deficincies than the one on the right in Green).

Miguel et al. (2003) included GEQUAMO as one of the early Tailored Quality Models as shown in Figure 3.7.

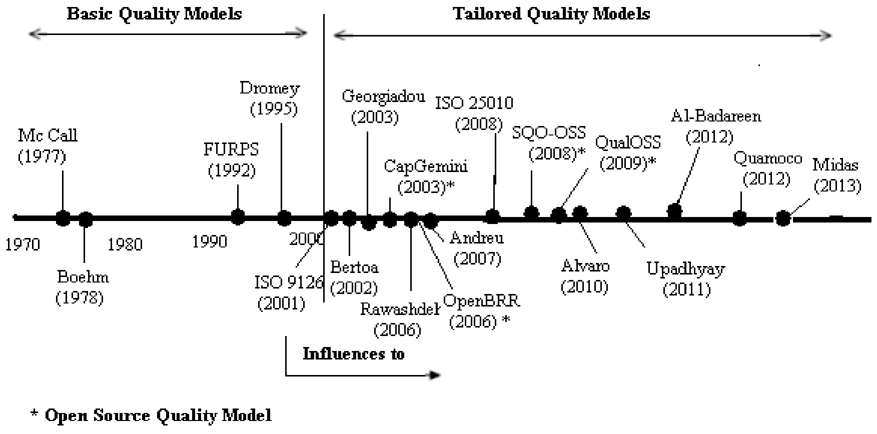

Figure 3.7 - Basic and Tailored Quality Models (Source Miguel et al., 2014)

In **Paper V** weaknesses of GEQUAMO were addressed resulting in GEQUAMO II. As one of the aims of the GEQUAMO (and GEQUAMO II) is the visual representation which enables immediate and easy comparison especially for comparative evaluation of items, it was found that decomposition to various sub-characteristics and over several layers may result in the braches running into each other making the picture confusing and potentially illegible. This is a limitation that needs to be recognised and managed through adhering to refined construction rules. Also through the validation exercise it became apparent that the combination of CFDs and the repeated use of Kiviat Diagrams caused the model to be unstable due to confounding errors. This was addressed by opting for a straight averaging of the terminal values (assuming they are equally weighted) as a more representative value to be propagated to the parent branch.

3.6 Summary of Chapter 3

This chapter studied the development of methodologies, lifecycles and quality models. The major contribution at this stage was the development of the GEQUAMO model (and its refinement in GEQUAMO II), which integrates top down (qualitative) feature analysis, bottom up (quantitative evaluation), and visualisation. Thus the model provides both rigour and clarity to both practitioners and academics alike.

CHAPTER 4
SOFTWARE MEASUREMENT

"To measure is to know. If you cannot measure it, you cannot improve it." (William Thomson also known as Lord Kelvin) (1824 - 1907).

4.0 Introduction

This chapter presents a discussion on measurement in general and software measurements and metrics in particular. It identifies controllable and uncontrollable factors and the challenges of understanding, measuring and controlling the quality of both the product and the process

4. 1 What is measurement?

Fenton and Pfleeger (1997) provide a definition of measurement: *"Measurement is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to characterise them according to clearly defined rules. The numeral assignment is called the measure."*

This definition provides a rigorous basis for determining when a proposed measure characterises an attribute and provides rules for determining what statistical analysis are relevant and meaningful. Hence, in order to understand the definition of measurement in the software context, we need to identify the relevant entities and attributes which we are interested in characterising numerically. Measurement theory provides the rigorous framework for determining when a proposed measure characterises an attribute and provides rules for determining what statistical analysis is relevant and meaningful (Briand and Wüst, 2001).

Entities of interest include objects, (e.g. code, specification, person) or processes (e.g. analysis, error identification, testing). Distinct attributes might be length of code, duration, costs. Representation is usually in numbers (or other mathematical objects e.g. vectors and ratios). Finally in order to provide objectivity we need to assign numbers (symbols) according to explicit rules which ensure that the assignment is not random.

Measures and quantitative information in general appeal to practitioners and researchers alike. Simple counts, ratios, comparisons and estimations constitute the backbone of many decisions in science, engineering, organisations and life in general. The Carnegie-Melon Software Engineering Institute’s Capability maturity model CMM (and later CMMI) level 4 level requires that measures of software process and product quality be collected so that process effectiveness can be determined quantitatively (CMMI Product Team, 2002). A process database and adequate resources are needed to continually plan, implement, and track process improvements.

At CMM/CMMI level 5 the optimising level, quantitative feedback data from the process allows continuous process improvement. Data gathering has been partially automated. Management has changed its emphasis from product maintenance to process analysis and improvement (Agrawal and Chari, 2007)

Defect cause analysis and defect prevention are the most important activities added at this level. Very few organisations keep metrics on either systems or the software development process. However, the Software Engineering Institute's Capability Maturity Model (CMM) estimates returns of four- or five-to-one for successful metrics programs.

In addition to understanding what is measurement we need to decide what we are measuring, and how we are measuring in order to maximising success and minimising failure of a metrics programme.

4.2 Controllable Quality Factors

The challenges faced when we are trying to understand, measure and control the quality of both the product and the process are presented in **Paper VI.** Measurements of both the current and the desired system are necessary. Internal metrics (Fenton, 1991) can be obtained in terms of the product (code). They are counts (such as LOC, Number of Classes, McCabe Complexity) and ratios (such as Number of calls per Module, Average size of module, and Average length of hierarchy), and they are also known as direct metrics. Additionally, these metrics can be generated automatically by using static analysis tools such as CANTATA, Testbed and Logiscope. Attributes such as morphology, architectural structure, depth of class hierarchy, size of module, maximum level of module complexity etc. can be controlled through a management mechanism and specific guidelines to the developers.

Controllable design parameters can be found in the software development process, the software product and the software development environment (Fenton, 1994; Kitchenham, 1996). However, external attributes (Fenton 1997), which are behavioural such as understandability and maintainability are more elusive and more difficult to measure. Metrics for these attributes are both qualitative and quantitative. They are almost always obtained indirectly through the use of *surrogate measures* (Kitchenham, 1996). For example maintainability can be estimated, calculated and thus controlled through measuring the time taken for a specified maintenance task. Results obtained by Georgiadou et al. in a series of controlled experiments provided confidence (through statistical methods) in the ability to effectively use surrogate measures (Georgiadou, 93, 94, 97, 98, and 2001).

4.3 Uncontrollable Factors

Human factors are unpredictable and mostly difficult, often impossible to control. For example, one such factor is performance variability in a human being, such as their experience and communication skills needed within a software development team. The developers' performance has an effect on producing quality software products in a similar way to the effect of machines on the manufacturing of products. It is important to maximise and properly maintain programmers' performance. The possible control factors will be conducting educational sessions within and outside a company where software developers are encouraged to learn the new techniques of their interest or to polish their skills. Recreational events may help developers to get to know each other better and this will be reflected in better communication and teamwork in an office. Ergonomic office design, temperature and humidity in the workplace can also affect the developers' performance.

Experimental evaluations, carried out by Basili et al. (1986), Shepperd (1990), Georgiadou (1999, 2001), attempted to identify design parameters and hence factors, which can be controlled. According to Taguchi cited in (Logothetis and Wynn, 1989) it is desirable to choose the set of design parameters, which are less affected by such factors. For example, developers' experience can be controlled to a certain extent by years in the profession and looking at the past projects they had been involved in. However every individual is unique. Their individual capability, patterns of learning and cognition are likely to be different from those of others of similar experience. The health of the developers may also have an effect on their performance at work.

In addition to differentiating between controllable and uncontrollable factors **paper VI** reported on a collaborative multidisciplinary study (in industry) which revealed that the holistic nature of such an approach provides software developers with the use of software measurement as the instrument for understanding, estimating and controlling the quality of specified factors. It must be born in mind that different stakeholder (discussed in **Papers IV and V)** place different emphasis on software attributes by different stakeholders. Usability and reliability are primary concerns to the user. Usability is enhanced through greater understandability, which in turn is enhanced through design correctness and consistency and through training, on-line help and support, all of which reduce productivity with the possible exception of Component Based Development which makes extensive reuse of code and increasingly reuse of designs too.

Reliability is of interest to all groups of stakeholders (in this case users, developers, and sponsors). It can be achieved through correct design, testing, walkthroughs, reviews and inspections. Enhanced reliability increases productivity and therefore decreases costs. Also enhanced functionality increases costs (in the short term) and may cause losses to the sponsor.

It is important for the software developer to understand the objectives and requirements for software product or the process improvement, and to specify the product/process response characteristics that reflect these objectives. The formulation of the problem as well as the production of a list of controlled parameters and noise variables can be achieved through brainstorming and formulated using techniques such as the Ishikawa (cause and effect) or fishbone diagram.

Measurable and, hence, controllable objectives should be chosen such as the number of bugs found during formal inspections, which are conducted during the software life cycle under the specified methodology a company adopts.

In order to understand and control the process we need measurements of both the current and the desired / new system. Internal metrics can be obtained in terms of the product (code) and they are counts (such as LOC, Number Classes, and McCabe Complexity) and ratios (such as Number of calls per Module and Average length of hierarchy).

Attributes such as the morphology, architectural structure, depth of class hierarchy, size of module, maximum level of module complexity etc. can be controlled through a management mechanism and specific guidelines to the developers. *Controllable* design parameters can be found in the software development process, the software product and the software development environment.

However, external attributes such as understandability and maintainability. These characteristics are behavioural, thus more elusive and more difficult to measure. Measurements of these attributes are almost always obtained indirectly through the use of surrogate measures. For example maintainability can be estimated, calculated and controlled through measuring the time taken for a specified maintenance task.

Also human factors are unpredictable and mostly difficult, often impossible to measure and hence to control. One such example is the performance variability in a human being, such as his/her experience and communication skills needed in a software development team. The developers' performance has an effect on producing quality software products in a similar way to the effect of machines on the manufacturing of products. It is important to maximise and properly maintain programmers' performance, however difficult it is due to the uniqueness of every developer. In this paper some **high level guidelines** for developing cultural awareness and for practical measures such as training of staff were provided.

Product quality characteristics specified in various standards and quality models refer to the behaviour or perceived behaviour of software such as Usability, Maintainability, and Reliability. Software Quality is undoubtedly desirable by all stakeholders.

However, the emphasis and detail of their respective requirements differ. Hence, some attributes are considered to be more important than others depending on the stakeholder. In "*The complete alphabet of quality software systems*"(Siakas, et al., 1997) the same stakeholder groupings (user, developer, sponsor) were used indicating only whether a characteristic is of high interest, low interest or of no interest to each group of stakeholders.

In **paper VII** the viewoints and the degree of interest in certain characteristics users, developers and sponsors were considered. Sponsors and managers and other decision makers were considered, because they are the representatives of the financing organisation. Users are considered to be the persons who, in different ways, use the final software product. Users can either be internal in the organisation that develops the software or external customers who use the software. Developers were considered as the persons that are not users or sponsors. Developers can be involved in the production of artifacts at different life-cycle stages. They are primarily interested in functionality and reliability.

A sponsor may be more interested in the overall quality rather than in a specific quality characteristic. The sponsor, in order to optimise quality within a limited time- and cost-frame may need to balance the quality improvement with management criteria for schedule delays and cost over-runs.

There is universal agreement that Functionality and Reliability are common concerns to all stakeholders. However, it can safely be assumed that a software systems developer strives to produce reliable and maintainable systems with maximum functionality. The user desires a system which is reliable, understandable, usable, easy to learn and easy to use and with the necessary functionality. The sponsor is extremely interested in maximising productivity i.e. he/she requires a reliable system with the necessary functionality, produced within acceptable time limits and at the lowest cost possible.

Enhanced Reliability (usually achieved through testing, walkthroughs, reviews and inspections) will most probably reduce productivity and will therefore increase costs. Both of these cause losses to the sponsor. Enhanced functionality increases costs (in the short term) and causes losses to the sponsor.

An **extension to the ISO 9126** (2001) model was proposed through the introduction of ***Extensibility*** and ***Security*** as primary characteristics, hence the acronym PERFUMES which stands for Portability, Efficiency, Reliability, Functionality, Usability, Maintainability, ***Extensibility, Security***, the first six characteristics at this primary level are those of the ISO 9126.

4.4 Conclusion to Chapter 4

Product quality characteristics specified in various standards and quality models refer to the behaviour or perceived behaviour of software such as Usability, Maintainability, and Reliability. Software Quality is undoubtedly desirable by all stakeholders. However, the emphasis and detail of their respective requirements differ. Hence, some attributes are considered to be more important than others depending on the stakeholder.

Chapter 4 refers to a collaborative multidisciplinary study (in industry) which revealed the strengths that the holistic nature of such an approach provides software developers with the use of software measurement as the instrument for understanding, estimating and controlling the quality of specified factors. Measurability and hence controllability of factors need to be established in order to decide what should be measured.

CHAPTER 5

A SUITE OF RANDOMISED CONTROLLED EXPERIMENTS

5.0 Introduction

This chapter describes the main contributions in terms of product. The derivation of the composite metric rho (ρ) which is an indicator for answering the question on whether deficient or malfunctioning systems should be scrapped or re-engineered. It also presents the design and execution of several controlled experiments.

5.1 Controlled Experiments

Having formulated the research hypotheses it was necessary to test them. According to Galliers ***(1992) Formal laboratory controlled experiments*** are primarily used for testing a hypothesis to establish the confidence with which you may predict the implications of a particular theory. They depend heavily on very careful experimental design (using replication, randomisation, blocking) and on the application of statistical techniques to analyse the results. Laboratory experimentation involves the creation of an (artificial) environment, in order to isolate and control potentially confounding variables.

This is the reason such experiments especially in software engineering are mostly run by academics within a university environment where a fairly large number of students can carry out the experimental tasks.

A total of 8 controlled experiments were designed and conducted over a period of 7 years. The reason is that new cohort of novice programmers needed to be recruited as experimental subjects i.e. the people who were to carry out the experimental tasks. The experimental design required the use of 4 laboratories with identical hardware and software. In addition to technical support, two members of staff were required for invigilation in each laboratory.

One experiment was abandoned due to weather conditions (severe snowfall!). It was subsequently re-scheduled due to staff absence, and a third experiment was totally invalidated due to a total breakdown/closure of one of the laboratories. Details can be seen under Logistics or Murphy's Law in Paper XII, and in outline form in Section 5.7).

Prior to any execution of each experiment a brief trial session was conducted to test the laboratory equipment, and to familiarise the students as well as the invigilators with the process.

5.2 Testing the research hypotheses

In this investigation the product characteristics selected to study were maintainability and understandability. The research hypotheses were formulated (Section **1.8.3)** as follows:

H1.1 The maintainability of software depends on its complexity.
(i.e. Low module complexity results in high maintainability).

H1.2 The maintainability of software depends on its structuredness
(i.e. Highly structured programs are highliy maintenable).

H2.1 The understandasbility of software depends on the use of colour in the HCI.

H2.2 The understandability of software depends on the use of non-symbolic naming of variables.

Paper VIII reports on the role of re-engineering in the management of software quality. The randomized, controlled experiment tested hypothesis H1.1

H1.1 The maintainability of software depends on its complexity.
(i.e. Low module complexity results in high maintainability)

Modules were judged a good candidate for modularisation if:

their granularity was greater than 50;

their McCabe Complexity was greater than 10;

they contained common code.

The hypothesis was born out. Details can be seen in ***Paper VIII.***

5.3 The derivation of the re-engineering factor ρ (rho)

Paper IX presents the rationale and derivation of the re-engineering factor ρ (rho). The specification of the criteria for both complexity and structuredness are provided. Among the many metrics proposed the four metrics selected to represent the profile of each program were Granularity, McCabe Complexity, Information Flow, and Number of Local Variables.

The derivation of the criteria for software restructuring was specified as Granularity, Information Flow, Local Variables, and McCabe Complexity. Once target (desirable) values of these four attributes are selected, the actual measures are obtained by static analysis and plotted in an anticlockwise sequence. Comparing the star plot of a program to the target plot (inner quadrangle) the differences cannot only be compared visually but also measured.

The re-engineering factor rho (ρ **)** can be calculated using the values of the following formula

Re-engineering Factor (ρ) = (Actual Area - Target Area) / Actual Area

Considering an example where the target (desirable) values for the four module characteristics of interest were specified as follows:

Number of Local Variables = 5, Granularity = 50,

McCabe Complexity = 7, Number of Local Variables = 6.

Programs under consideration for re-engineering are statically analysed and their profiles are superimposed onto the target module profile as shown in the star plot (Figure 5.1) where the inner rectangle depicts the desirable profile whilst the outer rectangle depicts the actual profile.

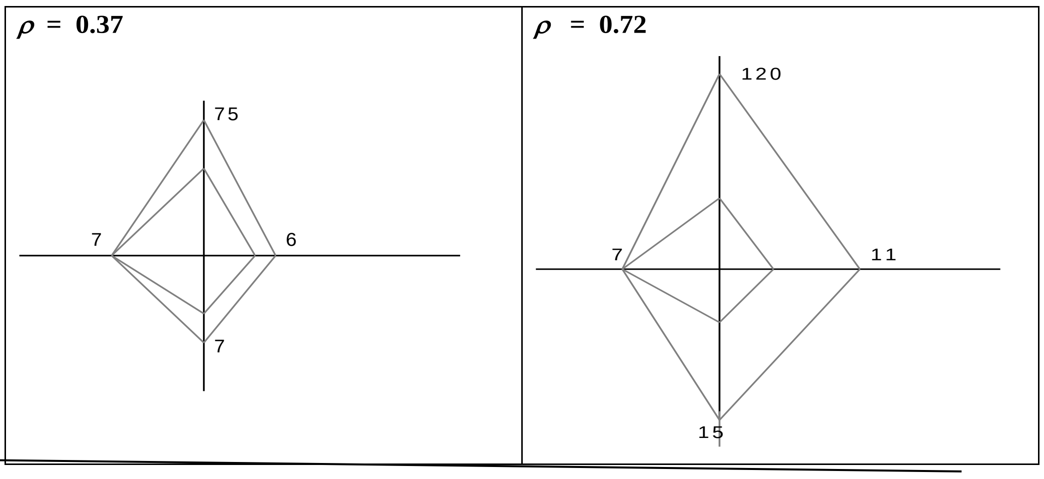

Figure 5.1 – Star plots for two programs under consideration for restructuring

It can be seen that the program on the right deviates substantially from the target (desirable). Thus the program on the right is a stronger candidate for restructuring than the program on the left. The tolerance level on the size of the deviation is a management decision as to whether it will be preferable to re-engineer a program or to discard and develop from scratch. The higher the value of ρ, the more deviation from the desirable profile, in which case the more likely to opt for restructuring. **The re-engineering factor rho (ρ) is a new composite metric.**

5.5 Maintainability of software depends on its structuredness

ISO/IEC/ IEEE 24765 (ISO 2010) - Vocabulary of systems and software engineering) gives three definitions (and the source of the definitions) for maintainability (Definition 3.1668):

- the ease with which a software system or component can be modified to change or add capabilities, correct faults or defects, improve performance or other attributes, or adapt to a changed environment;
- the ease with which a hardware system or component can be retained in, or restored to, a state in which it can perform its required functions ;
- the capability of the software product to be modified. ISO/IEC 14764:2006 (IEEE Std 14764-2006), Software Engineering — Software Life Cycle Processes.

This standard also provides the following definitions for maintenance:

- the average effort required to locate and fix a software failure
- the speed and ease with which a program can be corrected or changed.

(IEEE Std 982.1-2005 IEEE Standard Dictionary of Measures of the Software Aspects of Dependability).

The definition for maintainability used in this research is close to part 1 of the above definition 1:

Maintainability is the ease with which a software system (program) or component can be maintained (i.e. modified to change or add capabilities, correct errors, faults and/or defects, to improve performance or other behaviour, or adapt to a changed environment).

Sine maintainability is not directly measurable, the surrogate measure ***Maintenance*** (as defined by ISO/IEC/ IEEE 24765 (ISO 2010)) was used.

The tasks were to identify and correct embedded logic errors in two versions (one non-structured and one structured) of a program. The surrogate measure was the time taken to carry out the experimental tasks.

Paper X reports the design, execution and results of a randomised controlled experiment estimating the effects of predetermined changes in program structure on the maintainability of different program versions seeded with equivalent logic errors. The experiment tested the research sub-hypothesis

H1.2 The maintainability of software depends directly on its structuredness.

Prior to the execution of the experiment, programs were statically analysed to obtain measurements of internal sub-attributes of the fundamental attribute of structuredness. A first version of a program was modularised according to established rules (specified in **Paper X**) giving a new version of the program with a larger number of modules but with a smaller individual module complexity, and smaller average module complexity.

Structure is an internal attribute. It is multidimensional and as such it cannot be measured directly. The external term structuredness is difficult to define and to measure.

The sub-attributes of structuredness selected were: McCabe Cyclomatic Complexity, Information Flow, Number of Identifiers (Local Variables) and Granularity. A star plot provides a visual comparison of the module profiles. Details can be seen in **Papers VIII and IX.**

In order to ensure validity, interpretability and accuracy of the results factors such as different programming experience of the experimental subjects were factored out. The subjects were students who were novices in C programming although they had earlier passed a programming unit in Modula 2.

The experimental design (shown in Table 5.1) was a cross-over design involving two versions of each program.

Table 5.1 The Cross Over Design

Group	P1	P2
A	V1	V1
B	V1	V2
C	V2	V1
D	V2	V2

Where	P1V1 is	Program 1 version 1 Unstructured
	P1V2 is	Program 1 version 2 Structured
	P2V1 is	Program 2 version 1 Unstructured
	P2V2 is	Program 2 version 2 Structured.

The hypothesis was born out with a p-value equal to 0.008. The unstructured version was more difficult to understand and therefore more difficult to maintain.

Additionally from this first experiment several lessons were learned relating to erratic attendance or reluctance of subjects to conform to rules, such as stop working on program 1 and start working on program 2. Several subjects kept swapping from program to program which meant that we should have had a mechanism of stopping work on the first program at a specified time before embarking on the next one. This same cross-over experimental design was used in all subsequent experiments.

The results of this work can be used to provide an indicator for re-engineering whereby a given program can be restructured in such a way that quality improvement can be quantified or at least estimated.

Hypothesis ***H1.2 The maintainability of software depends on its structuredness*** *was born out.*

5.6 Measuring the Understandability of a Graphical Query Language

In addition to manipulating code the final experiment, reported in **Paper XI**, dealt with the measurement of understandability of the new Graphical Object Query Language named GOQL developed by my co-author Euclid Keramopoulos. The requirements of Keramopoulos were to test various aspects that are likely to impact on the understandability and usability of the interface for his Graphical Query language.

The author:

provided the rationale for selecting the formal experiment evaluation method;
formulated the hypothesis;
designed the experiment;
found the experimental subjects;
ensured laboratory support by colleagues;
helped analyse the experimental results.

The co-author produced the experimental materials and co-analysed the results.

"The use of colour and non-symbolic representation enhance the understandability of the graphical user interface of GOQL."

Again the design was cross over as shown in Table 5.2.

Table 5.2 – Cross Over Design for testing understandability

Group (of experimental subjects)	**Colour (in variables)**	**Symbolic Representation of Variables**
A	Y	N
B	**N**	**N**
C	Y	Y
D	N	Y

Both hypotheses (H2.1 and H2.2) were born out evidenced by the positive correlation between the independent variables (colour and symbolic representation of variables) and the response variable (understandability). The participation of the subjects in providing feedback was captured and analysed. Many of their suggestions fed back into the design and improvement of GOQL.

In addition to proving the research hypotheses, many lessons were learned regarding the logistics. The trial run of the experiment helped avoid problems with equipment and absences of experimental subjects.

5.7 A Framework for the Design and Execution of Controlled Experiments

Following the design and execution of 7 formal, randomised controlled experiments the knowledge and experience gained culminated in the development of a framework for the design and execution of this type of experiment. This framework was presented in **Paper XII**. The experimental process involves a number of Phases and deliverables at each phase. Before the actual execution of an experiment, the experiment needs to be designed according to scientific principles and considerable preparation must take place. Figure 5.2 depicts the proposed framework which consists of three Phases and seven stages.

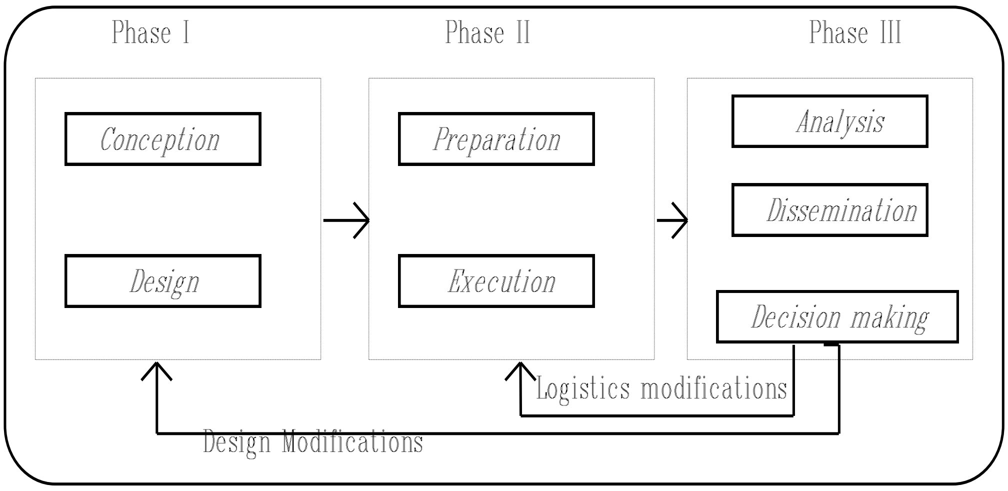

Figure 5.2 The experimental process and opportunities for Improvement

Experiments involve a large number of people including the experimental subjects and the scientific, academic and technical staff, the supervisors/invigilators and others. Experiments need laboratories, equipment and other resources. Experimental materials and tasks as well as methods for capturing data must be produced. Any of the many internal and external factors constituting and affecting an experiment are liable to cause failure and to invalidate an experiment as indeed happened in one of the experiments carried out during this research.

An experiment yields results which need to be analysed and interpreted. The process can be repeated, improved, refined and generalised. Logistics can cause delays or even failures. The proposed framework allows for feedback at the end of each experiment, so that necessary modifications could be carried out. Modifications are of two types namely experimental design, and logistics. Experiments are normally replicated. This iterative improvement builds a body of knowledge which in turn helps improve both the design and the logistics.

5.8 Summary of Chapter 5

This chapter showed the derivation of the composite metric and outlined three of the experiments to test the research hypotheses. The relationship between internal and external metrics was studied and tested experimentally. Beliefs that have been held by the community were tested.

Carrying out experiments is a very demanding and difficult undertaking. Considerable preparation and arrangements for a successful and reliable process have to be undertaken prior to any experiment. Among the tasks are: the design of the experiment, the preparation of the experimental materials, the involvement of a large number of people namely the experimental subjects (i.e. the people who will carry out the experimental tasks)   but also technical and academic staff.

Logistics and challenges regarding the availability of experimental subjects and laboratories as well as staff for invigilating the experiments were presented in a paper which included the proposal of a framework for guiding and improving the experimental process.

CHAPTER 6
SOFTWARE PROCESS IMPROVEMENT

6.0 Introduction

This chapter presents the work on process improvement and performance. The contributions reported here are: the Visualisation Framework I^5P, the $VALO_5$ and the integration of the two.

6.1 Knowledge Sharing, Process Maturity and Process Improvement

The philosophy and belief of this researcher as well as experiences from the manufacturing industry gave rise to the following assumptions:
Assumption 1: Process Improvement impacts directly on Product Improvement.
Assumption 2: Process Improvement depends on Maturity level and Knowledge Sharing.

Although it is necessary to improve deficient or failing products it is also important (even more important) to improve the process. This realisation is clearly evidenced and practiced in the manufacturing industry. This part of research explored the relationship of Knowledge Management, Knowledge Sharing, Process Improvement, Performance and Valorisation.

Paper XIII argues that Knowledge Management (KM) and Knowledge Sharing (KS) are strongly linked to organisational maturity. The mechanisms that enable this upward movement, and the achievement of measurable improvements in performance (depicted by the volume of the inverted cone at each level) as the organisation climbs from an adhoc/incidental level to institutionalised, higher levels of process maturity, were investigated.

The **I^{5}P** visualisation framework (shown in Figure 6.1) developed here aligns a Knowledge Sharing level to the appropriate maturity level and characterises the process from incidental to innovative. This framework provides the basis, in terms of preparedness and disposition towards knowledge sharing, for estimating and measuring organisational performance. In today's competitive global business environment organisations are increasingly dependent on Information and Communication Technologies (ICTs) and particularly vulnerable to knowledge dilution.

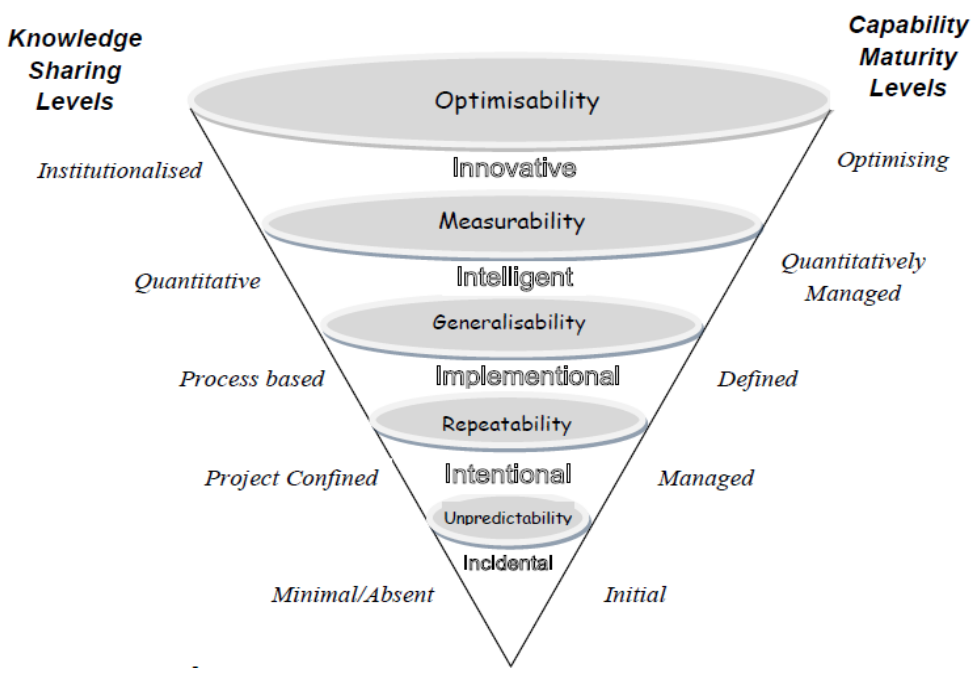

Figure 6.1 - The I^{5}P Visualisation Framework

The framework links knowledge sharing to process maturity, and aims to encapsulate accumulated tacit knowledge in the organisation by preserving it for future needs. The framework will be useful to Information Technology (IT) organisations especially those that are familiar with capability maturity models, such as CMMI (Jalote, 2000).

Knowledge Sharing between project teams, across departments and across the whole organisation depends on the process maturity level (Georgiadou et al., 2015). Trust engenders motivation and improves the process. Hall, T. (2002) carried out an analysis which provides managers with insights into designing appropriate SPI implementation strategies to maximise practitioner support for SPI.

The I^5P framework was developed with the participation of an industry based researcher and practitioner (Mr Bo Balstrup). It was validated by experts in both academia and industry and was supported by and implemented for the European Leonardo Da Vinci Project VALO project (number 2011- l -GR I - LEOOS -06789 in which Middlesex University was a partner, and of which I was initially the local Middlesex Co-ordinator).

6.2 Innovation, Maturity Growth, Quality and Valorisation

As the maturity of process grows, the quality of both processes and products improves. Innovation is encouraged and valorisation of results is facilitated. It is evident that high quality products (including research results) can be more easily disseminated and exploited.

Paper XIV presents the development of the $VALO_5$, which is a novel Valorisation Model (shown in Figure 6.2). Valorisation involves dissemination and exploitation activities. The overall objective of valorisation is to promote a project and its results, and to foster their use by different individuals and organisations, with the attempt of continuously spreading and improving the usage and the content of the results. Decision makers need to be convinced of the value of project outputs, and target groups need to be identified before dissemination. For continued exploitation of a project's results the identification of new environments, new contexts and new target groups are necessary.

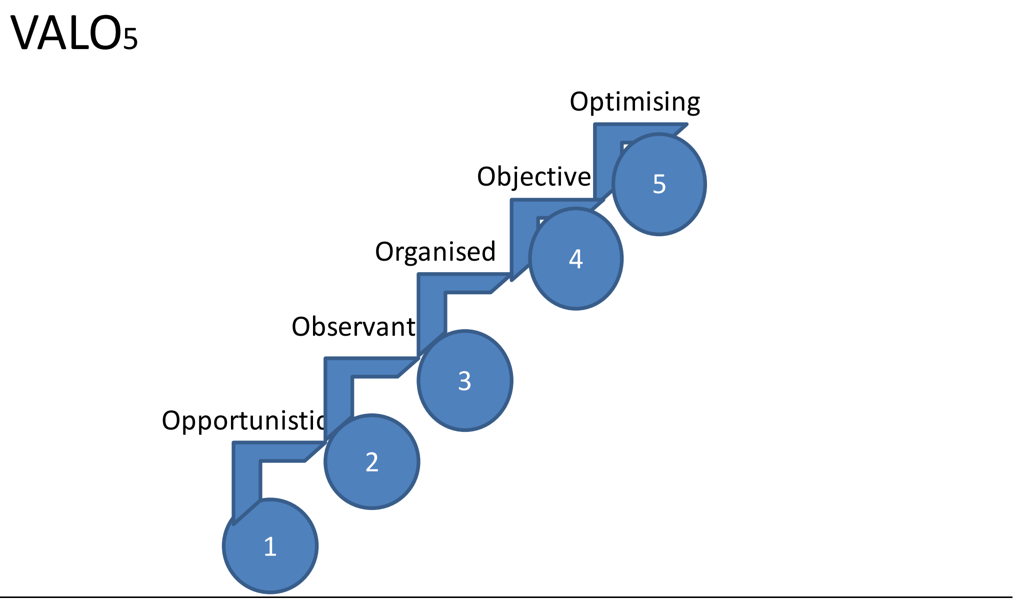

Figure 6.2 – The VALO$_5$ Model

Continuous Process Improvement Principles through on-going review of progress requires planning, implementing, checking/reviewing and correcting/modifying ensuring that errors, omissions or adaptations to accommodate changes .It is based on the principle of the 5-layer maturity model which was also used for the I^{5}P model. At every step Deming's PDCA cycle of continuous improvement is employed to Plan, Do, Check, Act. This work was partially financed by the European Leonardo Da Vinci project number 2011- l -GR I - LEOOS -06789 in which Middlesex University was a partner, and the author was co-investigator).

In Georgiadou and Sheriff (2008) and Georgiadou et al. (2015) the I^{5}P Framework and the VALO$_5$ project were further integrated (Figure 6.3) to show that as the Process Maturity rises, Knowledge sharing also rises, the performance of an organisation moves from the unpredictability level to the optimising level, thus generating the prerequisites for growth in the gained value from projects. Performance and Valorisation increase from Opportunistic (level) through to Optimising (level 5).

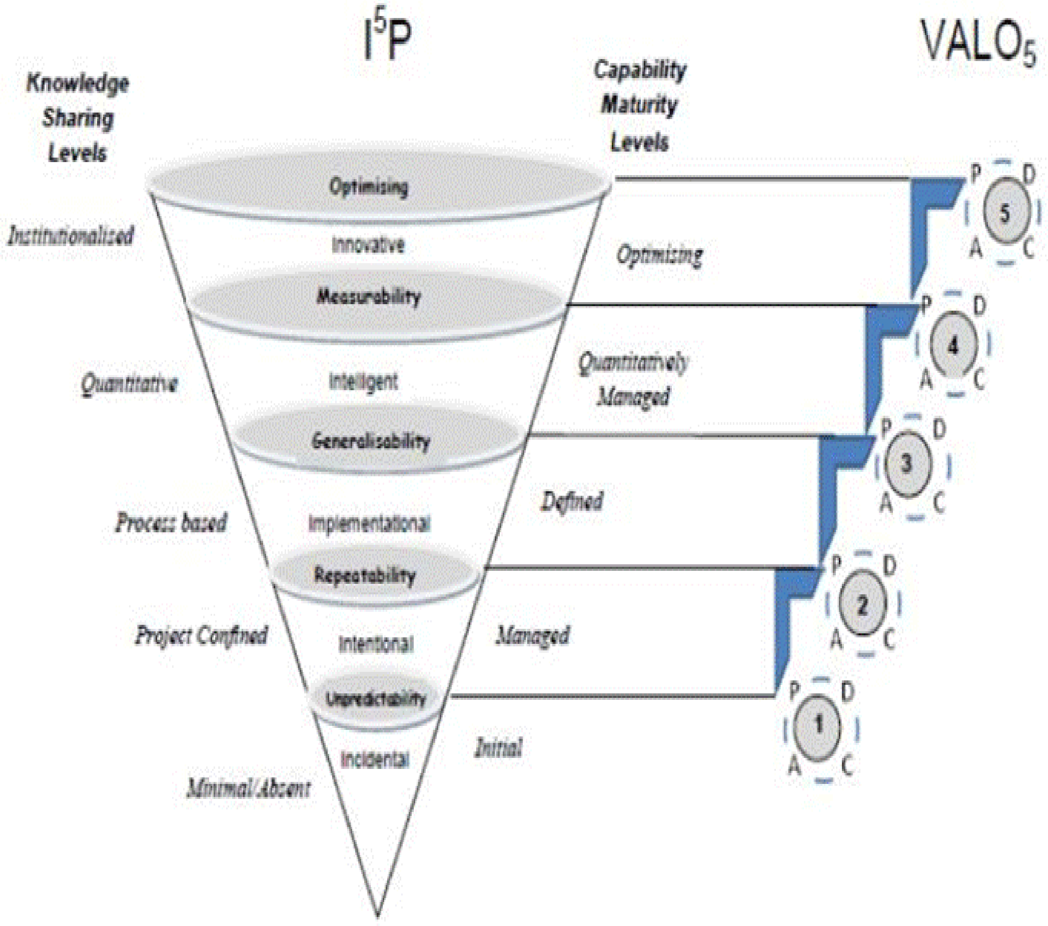

Figure 6.3 - Integrating I^5P & VALO5

6.3 CoFeD: A Visualisation Framework for Comparative Quality Evaluation

Quality evaluations for the purpose of selection are an everyday occurrence informing all decisions spanning the most trivial to the most profound in our individual lives, our professional lives and our scientific endeavours. The challenge of making the most appropriate selection especially from a plethora of available options becomes enormous when the risks of making the wrong choice are imminent and when they have the potential of high and even catastrophic impact. Evaluation for the purpose of selection can be a challenging task particularly when there is a plethora of choices available. Short- listing, comparisons and eventual choice(s) can be aided by visualisation techniques.

In **Paper XV** Feature Analysis, Tabular and Tree Representations, and Composite Features Diagrams (CFDs) were used for profiling user requirements and for top-down profiling and bottom-up evaluation of items (methods, tools, techniques, processes and so on) under evaluation. The resulting framework **CoFeD** enables efficient visual comparison and initial short-listing. The second phase uses bottom-up quantitative evaluation which aids the elimination of the weakest items and hence the effective selection of the most appropriate item. The versatility of the framework is illustrated by a case study comparison and evaluation of two agile methodologies. Figure 6.4 shows the architecture of CoFeD which revolves around a central hub, the process of continuous Review. This ensures continuous improvement of process and continuous improvement of its outputs (products).

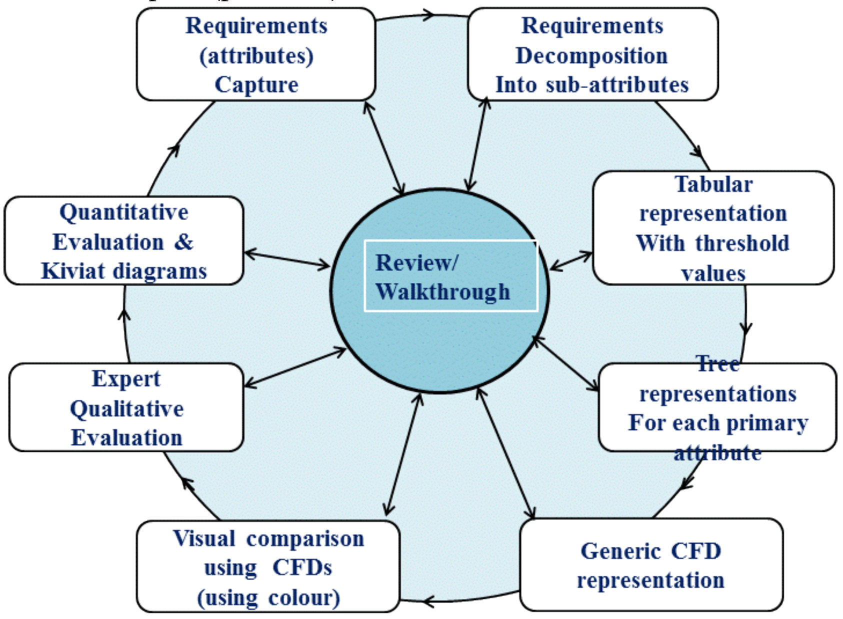
Requirements (attributes) Capture
Requirements Decomposition Into sub-attributes
Tabular representation With threshold values
Tree representations For each primary attribute
Generic CFD representation
Visual comparison using CFDs (using colour)
Expert Qualitative Evaluation
Quantitative Evaluation & Kiviat diagrams
Review/ Walkthrough

Figure 6.4 - The CoFeD Framework Architecture

The techniques used within CoFeD are Feature Analysis, tree representation, Composite Features Diagrams (CFDs), tabulation, and Kiviat Diagrams.

Visualising the profiles of items (methods, tools, people, artifacts and so on) under investigation helps novices, practitioners and experts, to make informed and quick judgements and make decisions. CoFeD and particularly the CFD technique have been used by a notable mobile phone company (which cannot be disclosed because of commercial confidentiality) to plan new versions of their products looking particularly at comparing and improving the playfulness of mobile phones.

6.7 Summary of Chapter 6

This chapter focused on Process Improvement and was based on the assumptions that

1: Process Improvement impacts directly on Product Improvement, and

2: Process Improvement depends on Maturity level and Knowledge Sharing.

The **I^{5}P** visualisation framework provides a mechanism that enables the estimation and quantification of the maturity growth, knowledge sharing and upward movement, and the achievement of measurable improvements in performance (depicted by the volume of the inverted cone at each level) as the organisation climbs from an adhoc/incidental level to institutionalised, higher levels of process maturity, were investigated.
The **VALO$_5$** model (which was applied to a the European Research and Knowledge Transfer project VALO contributes to the debate on gaining value after complication of a project either in industry or academia. Valorisation is itself a process which can grow and improve.
The integration of I^{5}P and VALO$_5$ as well as the Deming cycle Plan, Do, Check, Act (PDCA) emphasise the need for and the benefits of Continuous Improvement.

The **CoFeD** Framework revisits the concept of Product Improvement which also follows a process based on the principle of continuous review and improvement. This constitutes the third pillar of the overall research.

CHAPTER 7

REFLECTIONS AND REFINEMENTS ACROSS THE RESEARCH JOURNEY

7.0 Introduction

This chapter presents reflections on metrics validity through revisiting the composite metric rho (ρ). It also considers the ethical and legal dimensions of systems failures and recurring problem of terminology management/mismanagement and the need for terminology disambiguation for creating clarity.

7.1 Revisiting the re-engineering factor rho *(ρ)* and metrication

One of the early contributions of this research was the development and introduction of the re-engineering factor rho **(ρ)** which enables decision makers to either embark onto re-engineering legacy code or onto developing from scratch**.** Legacy code needs to be continuously updated, corrected, augmented, extended, migrated to new environments, platforms, languages etc.

Since I started this research journey many attempts to address the issue of system failures resulted in new lifecycle models, methodologies, techniques, tools. Yet systems continue to malfunction or fail.

Metrics and metrication of systems is by far the most rigorous method of understanding, controlling, and improving systems. A large number of metrics have been proposed, developed and used over 40 years. On revisiting rho (ρ), a small addition to this vast collection of metrics, I found it necessary to formally present a validation which I had not explicitly presented in earlier years.

Thus in **Paper XVI** "Is the Composite Software Metric ρ (rho) a Good Metric? " which I presented at the 26th International Software Quality Conference, 26 years since I presented Paper I at the very 1st International Software Quality Conference in 1993, I reviewed the major contributions by researchers and practitioners in software metrics.

Measures and quantitative information are of interest to both researchers and practitioners. Measurements are needed for understanding the current situation, for estimating costs and risks and, generally, for aiding decision makers in their operations. It is the backbone of sciences and engineering. Software Engineering is no exception. Software measurement and metrics have been developed and used for planning, estimation and improvement. Metrics can be simple counts, ratios, comparisons and estimations. They form the basis of most decisions in science, engineering, organisations and life in general.

Using Elaine Weyuker's Good Metrics Properties, the author validated the four module metrics namely Granularity (which refers to size), Number of Local Variables (which indicates the degree of cohesion), McCabe Complexity (which gives the alternative paths through the code and hence the number of test data required to exercise the code), and Information Flow (which indicates the degree of coupling). It was concluded that the composite metric ρ (rho), satisfies the Weyuker properties (Weyuker, 1988) since all its four constituent metrics satisfy these properties individually.

7.2 The need for Disambiguation of Terms

In natural languages we use words as synonyms and thus we use them interchangeably, but mostly when we intent to place particular emphasis on what we are trying to communicate. However, scientists and especially researchers give new or even erroneous meanings to existing words or create new words and combinations of words to name concepts, ideas, theories and products. An example from the Information Systems and Software Engineering domains of erroneous use of terms as synonyms from the Information Systems and Software Engineering domains is that of method and methodology. The (mis)use of these terms as synonyms has been perpetuated for over 40 years.

Such phenomena usually appear hen systems, processes or disciplines are new and immature. For example a study by Bozkurt et al. (2015 in their examination of 633 scholarly articles (covering 5 years from 2009 to 2013) reported that at least 12 terms are being used by scholars to describe Distance Education which is due to the unprecedented explosion of technology-based learning. These new terms are similar but not the same. They are, however, often used interchangeably as synonyms.

Software Engineering experienced unprecedented growth of new technologies, methods and theories in recent decades. As a result a large number of terms have been introduced. However, definitions of these terms can vary even across various ISO standards. For this reason, it has become necessary to collect and standardise terminology.

In **Paper XVII** "Navigating the labyrinth of Software 're' words" an examination of terms such as reuse, restructuring, re-engineering, reverse engineering, retro engineering and refactoring where considered. Their meanings and purpose were examined. These 're' words were identified as processes often overlapping and inter-related but with their distinct character and emphasis. It was concluded that these 're' processes aim to bring expected quality improvements but also potential problems arising from their use. The main benefits are reuse and often lower productivity costs than costs incurred from developing from scratch.

For example the expected quality improvements of the **re-engineering process**, which is the primary focus of this research, are: simplicity, understandability, reusability, usability, responsiveness, maintainability, adaptability, reliability, and efficiency. The paper identified and summarised the potential benefits of this study for practitioners but also for academics and scholars.

Paper XVIII (**Georgiadou**, 2018) reports on the current debate on terminology management. According to ISO (2010) Systems and software engineering — Vocabulary, "*The systems and software engineering disciplines are continuing to mature while information technology advances. New terms are being generated and new meanings are being adopted for existing terms. This International Standard was* *prepared to collect and standardise terminology…. It provides definitions that are rigorous, uncomplicated, and understandable by all concerned*.”

During the current decade, the debate on the need for terminology standardisation intensified, evidenced by research presented in Kirsch& Sauberer, (2011), Jacobson et al. (2013), (Clarke et al., (2016), and Sauberer et al., (2017). Researchers have argued that conceptual modeling, vocabularies, taxonomies, and ontologies are mechanisms of understanding and standardization.

According to Clark et al. (2013) *“conceptual modelling provides a mechanism by which a shared understanding between business domain specialists and IT specialists positively enhances the alignment of business and IT goals leading to improved quality of IT Solutions.”*

Ontologies formalise knowledge meaning, and facilitate the search for contents and information. Standardised terminology formalises knowledge meaning and facilitates the search for contents and information. The paper presented a chronology of seminal contributions to the terminology debate.

The systems and software engineering disciplines are continuing to mature while information technology advances at an unprecedented rate. During their journey from the general to the specific and then to the general they often generate new knowledge expressed in novel interpretations of existing terms or they even generate new terms in order to exemplify their research contribution, the originality of their work .

Continuous change and continuous improvements, innovations and changes in technologies bring their own new terms (which may be synonyms to existing terms) to add to the already highly populated vocabulary of Software Engineering. This in turn generates ambiguity.

Practitioners are normally focused on in-house projects and challenges. Their own experiential knowledge is often very useful for the theoreticians especially when they form part of the same team integrating theory and practice (**Paper XVIII**, 2018).

For this purpose I proposed a framework (shown in Figure 7.1) for launching, carrying out and implementing the terminology disambiguation process. It is proposed that the disambiguation of terms is a cyclic sub-process to ensure improvements are achieved through continuous monitoring and coordinating.

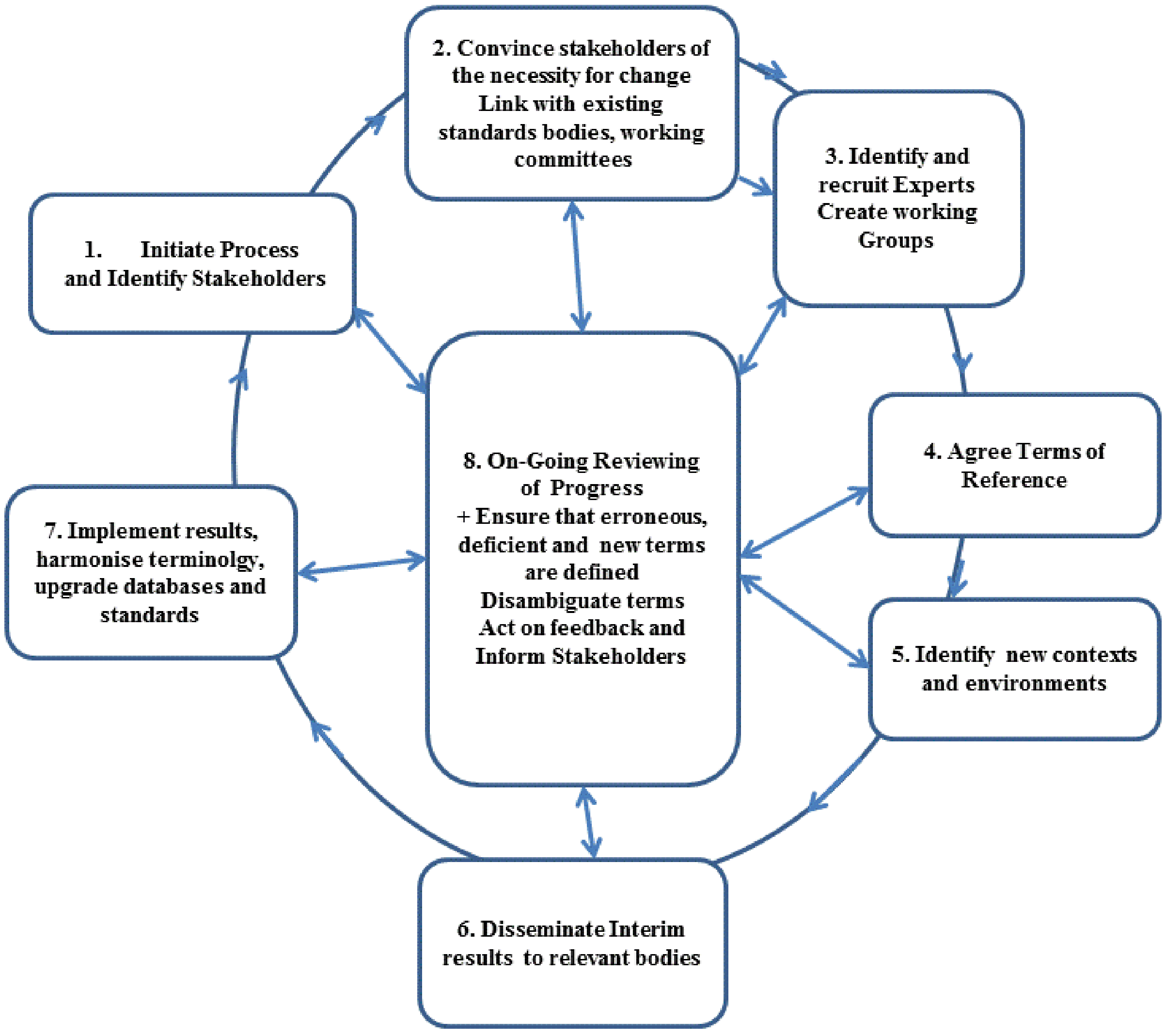
2. Convince stakeholders of the necessity for change Link with existing standards bodies, working committees
3. Identify and recruit Experts Create working Groups
1. Initiate Process and Identify Stakeholders
8. On-Going Reviewing of Progress + Ensure that erroneous, deficient and new terms are defined Disambiguate terms Act on feedback and Inform Stakeholders
4. Agree Terms of Reference
7. Implement results, harmonise terminolgy, upgrade databases and standards
5. Identify new contexts and environments
6. Disseminate Interim results to relevant bodies

Figure 7.1 - A framework for the effective disambiguation of terms [Source: Paper XVIII]

The central node (Number 8) forms the core of the framework. It is the on-going sub-process of reviewing all the work coming from all other stages and all committees, sub-committees, standards bodies and so on. The bidirectional arrows emphasise the need of information change, updating all stakeholders and responding to external changes.

Terms and their relationship to other terms may be rejected or accepted, modified, refined, and incorporated in existing databases. If experts disagree and cannot reconcile their differences in opinion, terms may be rejected or revisited for further elaboration. Without stifling creativity and imagination and the resulting generation of new terms or new meanings to existing terms, it is necessary to employ mechanisms to capture them but more importantly to identify and describe their relationship to existing terms.

7.3 Who should be responsible for bad quality and systems failures?

Software systems and software projects have been regularly failing causing losses which are not only financial but social too. The term 'social loss' implies:

- losses due to poor and varied performance of a product;
- failure to meet the customer's requirements of fitness for use or for prompt delivery;
- harmful side-effects caused by the product.

Thus it is important for educators to raise awareness among the new cohorts of students of their responsibility as scientists and as human beings to produce correct, reliable, functional and maintainable systems.

In **paper XIX** (**Georgiadou**& George, 2006) "Information Systems Failures: Whose responsibility?") , this question was posed concentrating on systems failures from the legal point of view. Although there is nowadays greater awareness of the dangers and possibilities of litigation in case of failure, and some awareness of the social responsibility of engineers, managers, and financiers, there is still low awareness of and commitment to the ethical dimension.

In Rahanu et al. (2018) it was argued that "*a set of defensible moral obligations that must be fulfilled in the development and deployment of systems, protagonists such as: project managers, software engineering teams, systems analysts, clients, etc. can fulfil their ethical duties, thus increasing the likelihood a deployed system that is compliant with principles of health and safety and wellbeing of its users. Ultimately systems development and deployment must be underpinned with ethical consideration.*"

7.4 Quality and Value are in the eye of the stakeholder

In Siakas, et al. (1997) Berki suggested that Quality is in the eye of the stakeholder. In **paper XX** a brief review of existing holistic, inclusive and participative approaches was presented. However, it was noted these approaches focused on 'success' rather than value. These approaches strived to enhance software products and/or software processes through value surrogates such as participation, negotiation, fairness, consensus, democracy, interaction, discussion and empowerment. Yet, software development is far from perfect. In **paper XX** it was proposed that, emphasis should aim at delivering mutually acceptable values through Value Compatible Appreciation (VCA), rather than focusing on surrogates of value which results in some types of compromise. VCA would engender the recognition and appreciation of mutual benefits and constraints.

7.5 Original contributions of this research

This research made several contributions to the debate on and practice of software quality, product quality, process quality and their improvement. It also developed metrics, guidelines, models and frameworks for guiding various processes and activities.

Tables 7.1a, 7.1b, and 7.1c summarise these contributions, the benefits they are intending to achieve, their limitations and indications of future work.

Table 7.1a Research Contributions, Benefits, Limitations and Future Work

Contribution	**Benefits**	**Limitations**	**Future Work**
Re-engineering Factor **a Composite Metric** **rho (ρ)**	Indicator for effective decision making on re-engineering (or not) of legacy code.	Applies only to procedural code at the moment.	Additional composite metrics e.g. develop: ρ_u *for improving usability* ρ_{oo} *for deciding on re-engineering of OO code*
GEQUAMO & GEQUAMO II	Generic Customisable Visualisation Quality Model enabling qualitative and quantitative evaluations for identifying strengths and weakness and for the purpose of selection. GEQUAMO uses both qualitative and quantitative methods for classifying features and their sub-features, and for measuring respectively.	The sub-attributes decomposition may result in overlaps of tree branches (must be controlled through the construction rules). The CFD (Composite Features Diagramming) technique is also a thinking tool for classifying groups of attributes and all their lower levels of attributes/features	Implementation of automated tools for visualising the profiles of items under evaluation and for bottom up measurement and quantification.

Table 7.1b   Research Contributions, Benefits, Limitations and Future Work

Contribution	Benefits	Limitations	Future Work
A Framework for the Design and Execution of Controlled Experiments	Streamlining the process which can be repeated, improved, refined and generalized. Logistics improvement. A body of knowledge can be gathered and processes can be improved.	Limited opportunities and extent to which identified relationships exist in the real world.	Further replication of experiments for additional validation and refinement of the proposed Phases of the Framework.
The I^5P Framework	Depicts the maturity, preparedness and disposition towards knowledge sharing, for estimating and measuring organisational performance. Implemented in and validated by experts from industry and academia.		Longitudinal monitoring of different size companies to obtain further performance measurements.
$VALO_5$	A new model for Innovation, Maturity Growth, Quality and Valorisation.		Longitudinal monitoring of different companies to obtain further data for facilitating the growth and gaining added value for companies.

Table 7.1c Research Contributions, Benefits, Limitations and Future Work

Contribution	Benefits	Limitations	Future Work
CoFeD: A Visualisation Framework for Comparative Quality Evaluation	Architecture of CoFeD revolves around a central hub, the process of continuous **Review** and improvement. Tried and tested representation and visualisation techniques.		Automated tools for profiling and visualisation
A Framework for the Disambiguation of Terms	Standardisation , Clarity	May stifle innovation and creativity	Co-ordinate with standards working groups
Guidelines for Value Compatibility Appreciation Analysis	Avoidance of dissent, conflict and resistance to change		Industrial surveys and case studies

CHAPTER 8 CONCLUSION

This chapter is the overall conclusion which summarises the Contributions to Knowledge and the Significance of the Study.

The ubiquitous presence of Software Systems brings many benefits to individuals, groups, organisations, and society at large. However, as they become increasingly complex a large number of them are regularly failing. Failed or challenged systems are not only costly financially, they are, and have been, harmful in terms of safety and violation of privacy.

The moment a software system goes live it is indeed a legacy system. The rate of change in technologies means that, apart from undergoing corrections to accommodate new or changed requirements, systems must accommodate change in general. They must constantly evolve and adapt. Changes can bring improvements but they are also likely to have a ripple effect which can be turbulent especially in complex and embedded systems.

Even when software systems continue to function, they maybe deficient and thus they require continuous corrective maintenance because requirements:

(i) had been initially misunderstood,
(ii) had been omitted or mis-specified, or
(iii) have changed.

Also, even if existing systems continue to function, as technologies change, they need to be migrated to new platforms and programming languages. Maintenance in this case is adaptive or perfective as is the case of performance improvement.

Preventive maintenance could be carried out, but in reality most of the time the whole software engineering community is ‘firefighting’ as problems crop up regularly. Although there has been a gradual shift from corrective to preventive methods, and the focus shifted from primarily product improvement to process improvement, software systems and also software projects have been failing regularly. Often software projects are abandoned after several years, and after wasting several millions of pounds. A brief look at articles such as "Britain: The health service's IT problem; Computerising the NHS"; The Economist, Oct 19, 2002, Vol.365 (8295), pp.51-52 will attest to the enormity of the problem.

This research spanned a period of over 25 years and encompassed aspects of quality improvement, starting from product improvement and re-engineering of legacy code, moving to process improvement and establishing interrelationships of process and product quality.

Controlled experiments were used to test various hypotheses which were born out. Software characteristics (both internal and thus directly measurable) and external (indirectly measurable) were studied and used to decide whether restructuring legacy code is advisable.

On re-engineering, this thesis contributes the **novel composite metric rho (ρ),** which can act as an indicator for decision makers on whether legacy code should be restructured or abandoned to be replaced by developing a new system from scratch.

A series of controlled experiments were designed and conducted to test the research hypotheses. At the end of Part 1 of the research, a framework for the design of experiments was developed.

The generic quality model GEQUAMO and GEQUAMO II were developed. They are customisable, as each class of stakeholder places different emphasis on characteristics such as usability, maintainability, and reliability. The models combine a top-down qualitative method for feature analysis, decomposition and classification, and bottom up quantitative evaluation and comparison, including visualisation, for the purpose of selection.

Based on the assumption and widespread belief that process improvement impacts positively on product improvement the research continued with a focus on process improvement by looking at process maturity and knowledge sharing and their impact on performance. **The I^{5}P framework** was developed in collaboration with industry and validated by experts from academia and industry. This framework provides a mechanism for estimating performance improvement as the maturity and knowledge-sharing within a team, a project, or an organisation advance from level 1 to level 5.

Further work on value gains from process improvement, the valorisation model **$VALO_5$** was developed under the auspices of a European Research and Knowledge Transfer project. Further integration of I^5P and $VALO_5$ as well as the Deming PDCA continuous improvement cycle (Georgiadou & Sheriff, 2008), and (Georgiadou, et al., 2015) showed that process improvement can bring value to an organisation, and can reconcile differences between different stakeholders.

The **CoFeD Visualisation Framework** for Comparative Quality Evaluation was developed and illustrated by case studies. Finally a study of dominant discourse on terminology management and the need for disambiguation of terms for the achievement of clarity and standardization resulted in a framework whose architecture is built around central hub of continuous review and feedback for continuous monitoring and standardisation without stifling creativity and innovation.

In addition to the above metrics, frameworks and models the thesis contributes to the on-going debate on software quality and quantifiable quality improvement to both product and process.

Quality is difficult to define, very difficult to achieve, and even more difficult to measure. In reality there can be no absolute or exact value for quality. The definition below was proposed and used throughout this research:

Software Quality is the totality of product as well as process characteristics, and their interaction and measurement (whether qualitative or quantitative) that satisfy different stakeholder requirements.

In engineering everything is an approximation, a compromise – sometimes small sometimes bigger. The degree of tolerance and threshold values/acceptable limits of deviation depend on many interdependent factors.

Icarus's flight to freedom ended up in disaster because the specifications given by his father Daedalus who was an architect were ignored and the safe limits were violated.

"'My Icarus' he says; 'I warn thee fly
Along the middle track: nor low, nor high;
If low, thy plumes may flag with ocean's spray;
If high, the sun may dart his fiery ray.'"

(Ovid)

Flyvbjerg and Budzier (2011) warned that IT projects are nowadays so big and their influence so wide ranging across many aspects of the organisation, that "*they pose a singular new kind of risk that can sink entire corporations, cities, and even nations."* As software is integrated into bigger products and systems, the concerns can become magnified. The software industry has the highest failure rate among all other engineering disciplines.

An occupation that runs late on more than 75 % of projects and cancels as many as 35 % of larger projects is not a true engineering discipline' (Jones, 2010).

As systems are not only software (Kaposi and Pyle, 1993) a holistic approach is necessary in order to achieve faster and better progress in quality improvement. Generally, quality is desirable by all stakeholders. Yet as "quality is in the eye of the stakeholder" (Siakas et. al, 1997) it is necessary to address the specific. Reconciling different worldviews which Peter Checkland (1981) called 'accommodating' different Weltanschauungen is a challenge that needs continuous attention and effort.

Finally, this research demonstrated that adopting a holistic approach enables the study and the improvement of both the product and the process. In addition it is possible to estimate and measure such improvements. It is certain that changes to the culture of relying on corrective strategies and actions to preventive strategy and actions are imperative.

"The world as we have created it is a process of our thinking. It cannot be changed without changing our thinking."

Albert Einstein (in an Interview with Michael Arminne, June 23rd, 1946)

CHAPTER 9
EPILOGUE – PERSONAL REFLECTIONS

9.0 Introduction

The Epilogue is a personal reflection on my research journey, my quest for knowledge and understanding, the creation of new knowledge, also the role of educators in raising awareness and sense of responsibility in preparing the younger generations of software engineers to understand and embrace ethical principles in their future professional and personal life.

9.1 Quality and Social Responsibility

Over the 25 years of this research many changes took place in the software engineering and information systems fields. Methods, methodologies, lifecycles, techniques, tools and frameworks were proposed, introduced, improved, and abandoned. However, the problems of software systems' deficient quality persist. It is also inevitable to need to deal with legacy issues as software evolves and changes. Thus, the research reported here is relevant to both the discourse and the practical implementations of solutions offered by academics and practitioners**.**

My quest for knowledge and understanding culminated in the creation of new knowledge, and the development of models and frameworks as mechanisms for product and process improvement. The visualisation techniques, models and frameworks, which I developed, aid understanding and decision making for both researchers and practitioners. The application of my research findings in industry, as well as in curriculum development, and European Projects for Research & Knowledge Transfer has been a gratifying endorsement.

Process improvement is intertwined with product improvement. This integration is widely believed to result in improved products and services. In Rahanu et al. (2018) it was emphasised that "*central to each of software process improvement models is the notion of a focused and sustained effort towards building a process infrastructure of* *effective software engineering and management practices. The software process improvement strategy aims for something that is more focused, more repeatable, and more reliable, with regards to the quality of the system developed (conformance to requirements, reliability, usability etc.), the timeliness of delivery and the expected cost. Quality in use also has implications on performance, reliability, and usability. The overall assumption is that a sound and improving process is likely to result in high quality systems i.e. process improvement is likely to result in improved products".*

My role as a researcher and as an educator enabled me to impart knowledge and expose my students to my philosophy of social responsibility, fairness, and morality. I was able to oversee the development of new curricula on Software Product and Process Improvement, particularly at Masters level, to discuss and test my knowledge and discoveries with many thousands of students (classes I taught were often 300-400 strong per annum), and dozens of colleagues in the UK, Europe and further afield. Links with industry kept my feet on the ground as practitioners have a focused mind on real, tangible problems that need to be addressed in the short term than is the case with academic investigations.

The advent of global connectivity and rapid technological change brought with them intense social change, threats against existing distributions of power, and capital. New obligations, new kinds of crime and over-dependency on interconnected technologies and embedded systems, are making it difficult to know whose responsibility it is to ensure the quality of systems in use can be assured. As an educator, in order to instill in my students (current and future software engineers) principles of right or wrong, I often start my classes on Information Systems Quality Management with the question***:*** ***Is 99% quality level good?*** I invariably receive a resounding YES! I give them 5 minutes to discuss among themselves, and then I show them the following table:

3.8 Sigma = 99% Good	6 Sigma = 99.999 % good
20,000 articles of mail lost each hour	7 articles lost per hour
15 mminutes of unsafe drinking water every day	2 minutes unsafe water per year
5,000 incorrect surgical operations per week	2 incorrect procedures per week
2 short or long landings at most major airports each day	1 short or long landing every 5 years
200,000 wrong drug prescriptions each year	68 wrong prescriptions per year

[http://www.snuniversity.nl/downloads/University/artikel-crosby.pdf (accessed 2/07/2018)]

After a few more minutes of discussion, I ask "What if you or a member of your friends or family is the patient that undergoes one of the incorrect surgical procedures, or if you or a loved one is a passenger on one of the short or long aircraft landings?" Their answers to this and to the first question are quite revealing as the students come to realise and reflect on the very real dangers of badly designed, poorly tested, erroneous, failing systems. The students can (and do) reflect on the concept and the importance of building quality into systems. They can also reflect on their own future role and responsibilities in this industry.

In addition, because the students originate from diverse backgrounds, cultures, and countries I also urge them to think of the word or phrase for ***quality and its meaning*** in their own mother tongue – only then they can feel and understand the importance of quality. This was a challenge posed by Shimon Peres to over 2,000 delegates at the 10th International Conference of the Israel Society for Quality, which I attended in November 1994.

The word **Quality** itself was introduced in the English language c.1300; it means "temperament, disposition", and derives from old English *qualite "meaning nature, characteristic", itself deriving from modern French* qualité, from Latin qualitatem "meaning property, nature, state, condition", said to have been coined by Cicero to translate the Greek (which is my own mother tongue) **poiotes** (**ποιότης**) meaning degree of goodness. Simon Peres' challenge made me think that **ποιότης** is also the root of the word poet (which means creator in Greek) and also of the word poetry *"which is perceived by many as the highest form of intellectual creation"* [https://www.etymonline.com/word/quality (accessed 1/07/2018)].

It can be assumed that quality (meaning good quality) is universally desirable. However, our field is still immature. Mature disciplines have clear and unambiguous nomenclature. Standard definitions, and standard measures.

9.2 Software Engineering: Is it an Engineering discipline?

Jackson (1994) asserted that *“software engineering is not a discipline; it is an aspiration, as yet unachieved. Many approaches have been proposed, including reusable components, formal methods, structured methods and architectural studies. These approaches chiefly emphasise the engineering product; the solution rather than the problem it solves.”*

Abran et al. (2003) emphasised that “*the metrology perspective suggests that the field of software measurement has not yet been fully addressed by current research, and that much work remains to be done to support Software Engineering as an engineering discipline based on quantitative data and adequate measurement methods meeting the classic set of criteria for measuring instruments as described by the metrology body of knowledge in large use in the engineering disciplines”.*

Despite the immense progress achieved it seems that even after 50 years since the coining of the term ***Software Engineering*** we are still far from justifying the term ***engineering*** compared to other types of engineering.

The work carried out by a number of researchers, such as Holcombe and Ipate, (1998) demonstrates that formal methods are probably the best way of designing correct systems. Anthony White (2013) developed a control-theoretic model of the requirements process. This model addresses “the questions of deadlines, quality objectives, effort, and size of the requirements team. The objective of the model, applied to the requirements process, is to help a manager make decisions regarding the expansion of the workforce, change of quality in the process and to control how the requirements process achieves its objectives”.

9.3 Continuous search for truth, knowledge and improvement

I, like most researchers started my investigation in a top-down manner, a wide ranging ambition spanning a large knowledge area often involving universal concepts, ideas, themes, principles, and ideals that are found and can be proven within, between, and across various subject areas and disciplines.

As researchers we gradually gain insights and understanding of the enormity and complexity of our originally ambitious scope and purpose, we narrow down our investigation focusing on smaller and more specific problems.

Cerca Trova (seek and ye shall find) [https://www.florenceinferno.com/cerca-trova-or-catrovacer/"(last accessed 1/07/2018)]. may be a mysterious inscription but inspiration, creativity, perseverance, an open and questioning mind are necessary abilities or qualities that a researcher must have to engage in effective problem solving and to find the underlying truth in any situation.

"Cerca trova" (seek and ye shall find) is *a mysterious inscription that is located at the top of* ***Vasari's fresco "The Battle of Marciano"*** *positioned in the Hall of the Five Hundred in* ***Palazzo Vecchio, Florence, Italy.***

CERCA TROVA

The words mean *"Seek and you shall find."* They appear on a flag in a battle scene (part of the painting can be seen here), painted as a fresco by Giorgio Vasari on one of the walls of the Salone dei Cinquecento (council chamber) of the Palazzo Vecchio.

Some people think that Vasari's painting may hide an earlier work by Leonardo da Vinci, called "The Battle of Anghiari."

My research journey was very long I did not wish it (as Kavafis a Greek poet from Alexandria, Egypt advised in 1911). However, life took me in different directions: there were illnesses, bereavements, temporary pauses, long stoppages, backtrackings, extraordinary work-loads, going in circular paths, restarting… Throughout this journey I gathered much knowledge, and I created some new knowledge too. I gained understanding of Software Systems Failures, and understood my own limitations. I am wiser. My life-long learning journey for seeking and sharing knowledge, for achieving deeper understanding, creating new knowledge and finding some solutions to seemingly unsolvable problems continues to perhaps reach the elusive destination of 'Ithaka'.

Ithaka by Constantine Kavafy

As you set out for Ithaka
hope the voyage is a long one,
full of adventure, full of discovery.
Laistrygonians and Cyclops,
angry Poseidon—don't be afraid of them:
you'll never find things like that on your way
as long as you keep your thoughts raised high,
as long as a rare excitement
stirs your spirit and your body.
Laistrygonians and Cyclops,
wild Poseidon—you won't encounter them
unless you bring them along inside your soul,
unless your soul sets them up in front of you.

Hope the voyage is a long one.
May there be many a summer morning when,
with what pleasure, what joy,
you come into harbours seen for the first time;
may you stop at Phoenician trading stations
to buy fine things,
mother of pearl and coral, amber and ebony,
sensual perfume of every kind—
as many sensual perfumes as you can;
and may you visit many Egyptian cities
to gather stores of knowledge from their scholars.

Keep Ithaka always in your mind.
Arriving there is what you are destined for.
But do not hurry the journey at all.
Better if it lasts for years,
so you are old by the time you reach the island,
wealthy with all you have gained on the way,
not expecting Ithaka to make you rich.

Ithaka gave you the marvelous journey.
Without her you would not have set out.
She has nothing left to give you now.

And if you find her poor, Ithaka won't have fooled you.
Wise as you will have become, so full of experience,
you will have understood by then what these Ithakas mean.

(C.P. Cavafy, Collected Poems. Translated by Edmund Keeley and Philip Sherrard. Edited by George Savidis. Revised Edition. Princeton University Press, 1992)

REFERENCES

NOTE: Many additional references to the ones below appear at the end of Papers I to XX.

Abran, A, Sellami, A, Suryn, W. (2003)Metrology, measurement and metrics in software engineering, Proceedings. 5th International Workshop on Enterprise Networking and Computing in Healthcare Industry (IEEE Cat. No.03EX717).

Agrawal M., Chari, K. (2007) Software Effort, Quality, and Cycle Time: A Study of CMM Level 5 Projects, IEEE Transaction on Software Engineering, Vol. 33, No. 3.

Avison, D., Fitzgerald, G. (1995), "Information Systems Development: Methodologies, Techniques and Tools", McGraw-Hill, 2nd edition.

Avison, D., Fitzgerald, G. (2003) “Information Systems Development: Methodologies, Techniques and Tools", Graw-Hill, 3rd edition.

Basili, V., R. Selby, W. & Hutchins D. H. (1986), Experimentation in Software Engineering. IEEE Trans. on Software Engineering, SE-12. p. 733-743.

Beck, K. (1999) Extreme Programming Explained: Embrace Change. Addison-Wesley.

Beynon-Davies, P., (1999) , "Human error and information systems failure: the case of the London ambulance service computer-aided despatch system project," in *Interacting with Computers*, vol. 11, no. 6, pp. 699-720, June 1999.

Berki, E., **Georgiadou, E**., Sadler, C., Siakas K. V., (1997): A Methodology is as Strong as the User Participation, International Symposium on Software Engineering in Universities - ISSEU 97, Rovaniemi, 7-9 March, pp.36-51.

Boehm, B. W., Brown, J.R., Kaspar, J.R., Lipow, M., MacCleod (1978) Charavteristics of Software Quality, Amsterdam, North Holland.

Bozkurt, A., Akgun-Ozbek, E., Yilmazel, S., Erdogdu, E., Ucar, H. Guler, E., Sezgin, S., Karadeniz, A., Sen-Ersoy, N., Goksel-Canbek, N.,Dincer, G.D., Ari, S. and Aydin, C.H. (2015), Trends in Distance Education Research: A Content Analysis of Journals 2009-2013, The international review of research in open and distributed learning, Vol 16, No 1 (2015)

Briand, L. C.: Wüst, J. (2001), *Integrating scenario-based and measurement-based software product assessment.* The Journal of Systems and Software, 59(2001), pp. 3-22.

Busha, C. A. & Harter, S. P. (1980). Research methods in librarianship: Techniques and interpretations. New York: Academic Press.

Charette, R. N., (2005) "Why software fails [software failure," in IEEE Spectrum, vol. 42, no. 9, pp. 42-49, Sept. 2005. doi: 10.1109/MSPEC.2005.1502528

CHAOS Report (1994): The Standish Group International, Available on-lineat http:// www.standishgroup.com/sample_research/chaos_1994_1.php, 1994(accessed 1/07/2018).

CHAOS Report (2015): The Standish Group International, http://www.standishgroup.com (accessed 1/07/2018).

Checkland P. (1981), Systems Thinking, Systems Practice, John Wiley and Sons Ltd .

Clark, T., Frank, U., Kulkami, V., Barn, B., Turk, D. (2013) Domain specific languages for the model driven organization, GlobalDSL 2013 Proceedings of the First Workshop on the Globalization of Domain Specific Languages Pages 22-27.

Clarke, P., Calafa, M., A. and Ekert, D. and Ekstrom, J. and Gornostaja, T. and Johansen, Jorn and Mas, A. and Messnarz, Richard and Najera Villar, Blanca and O'Connor, Alexander and O'Connor, Rory and Reiner, M. and Sauberer, G. and Schmitz, K-D. and Yilmaz, Murat (2016) Refactoring software development process terminology through the use of ontology. In: 23nd European Conference on Systems, Software and Services Process Improvement (EuroSPI 2016), 14-16 Sept 2016, Graz, Austria. ISBN 978-3-319-44817-6.

CMMI Product Team (2002) Capability Maturity Model, Integration (CMMISM), Version 1.1 CMMISM for Systems Engineering, Software Engineering, Integrated Product and Process Development, and Supplier Sourcing (CMMI-SE/SW/IPPD/SS, V1.1) Continuous Representation, CMU/SEI-2002-TR-011, ESC-TR-2002-011, Improving processes for better products.

Crosby, P. (1979) "Quality is Free", McGraw-Hill.

Dalcher, D. (2005). "Software Processes: Lessons and Reflections." Software Process Improvement and Practice 10(2): 99-100.

Dalcher, D. (2014) Rethinking Success in Software Projects: Looking Beyond the Failure Factors, in Software Project Management in a Changing World, Ruhe, G., Wohlin, C. (Eds), 2014, XX, 477 p.81 Springer.

Dalcher, D. (2017), Is it Time to Rethink Project Success? Keynote, Software Quality Management International Confernce XXV, in proceedings Achieving Software Quality, in Development and in Use, P Marchbank, M Ross, G Staples(eds) , Southampton Solent University, 2017.

del Águila, I.M., Palma, J. , Túnez, S. (2014) Milestones in Software Engineering and Knowledge Engineering History: A Comparative Review, ScientificWorldJournal. 2014; 2014: 692510. (10.1155/2014/692510).

Deming, W.E., (1986).Out of the Crisis. MIT Press. Cambridge, MA.

Dennis S., Gouran, Randy Y. Hirokawa & Amy E. Martz(1986)A critical analysis of factors related to decisional processes involved in the challenger disaster,Central States Speech Journal,37:3,118-135,DOI: 10.1080/10510978609368212.

Dromey, R.G. (1995), A Model for Software Product Quality, IEEE Transactions on Software Engineering, February, 1995, pp. 146-162. Elsevier, The Journal of Systems and Software 47 (1999) 149 – 157.

Eveleens, J. L., Verhoef, C. (2010) "The rise and fall of the Chaos report figures," IEEE Software, vol. 27, pp. 30-36, 2010.

Fenton, N. E., and S. L. Pfleeger, (1977), Software Metrics: A Rigorous Approach, 2nd ed., Boston: International Thomson Computer Press. Fenton, N.E. (1994), Software Measurement: A Necessary Scientific Basis, IEEE Transactions on Software Engineering, Vol. 20, No.3, 199-206.

Fenton, N.E., Neil, M. (1999), Software metrics: successes, failures and new directions, The Journal of Systems and Software 47 (1999) pp. 149-157.

Galliers, R. (Eds) (1992) Information Systems Research, Issues, Methods and Practical Guidelines, Blackwell Scientific Publications.

Flyvbjerg, B., Budzier, A. (2011), Why your IT project may be riskier than you think. Harv Bus Rev 89(9):83–85

Georgiadou E., Karakitsos, G.,Sadler, C., Stasinopoulos D. (1993) An experimental examination of the role of re-engineering in the management of software quality, Software Quality Management II Vol., Computational Mechanics Publications, 1993.

Georgiadou, E., Karakitsos G., Sadler C., (1994) "Improving the program quality by using the re-engineering factor metric ρ", The 10th. International Conference of the Israel Society for Quality, November 1994.

Georgiadou E., Sadler C.,(1995) "Achieving quality improvement through understanding and evaluating Information Systems Development Methodologies", 3rd International Conference on Software Quality Management, SQM'95, Seville, Spain, Apr. 1995.

Georgiadou, E., (2003), **"**Software Process and Product Improvement: A Historical Perspective", International Journal of Cybernetics, Volume 1, No1, Jan 2003 pp172-197.

Georgiadou, E., Sheriff, M. (2008) Reconciling stakeholder conflicts by analysing apparently contradicting notions of value in SE projects, In: Software Quality Management International Conference, 2008, Belfast, Northern Ireland.

Georgiadou, E., Siakas, K, V., Balstrup, B., (2010), The I5P Visualisation Framework for Performance Estimation through the Alignment of Process Maturity and Knowledge

Sharing International Journal of Human Capital and Information Technology Professionals, 2 (2), pp. 37- 47. ISSN 1947-3478.

Georgiadou, E., Siakas, K. V., Abeysinghe, G., Sheriff, M. (2015). Enhancing Project Value through evaluating organisational maturity and knowledge sharing capability, in R. Lock, R. Dawson, E. Georgiadou, M. Ross, G. Staples (eds), Outlook on Quality, the BCS Quality Specialist Group's Annual International 23rd Software Quality Management (SQM) conference, 30 March, British Computer Society, Loughborough, UK, pp. 47-62.

Marc-Alexis Côté, Witold Suryn, **Elli Georgiadou, (2007)** : "*In search for a widely applicable and accepted software quality model for software quality engineering"*, . Software Quality Journal 15(4): 401-416 (2007).

Georgiadou, E. (2003), GEQUAMO: a generic, multilayered, customisable, software quality model. Software Quality Journal, 11 (4). pp. 313-323.

Georgiadou, E. (2008) "GEQUAMO II Verification, Validation and Improvement of a Generic, Multilayered, Customisable, Software Quality Model ", Software Quality Management, 2008, Ulster University, March 2008.

Georgiadou, E. , Siakas K.V. , Berki E., (2003): Quality Improvement through the Identification of Controllable and Uncontrollable Factors in Software Development, 11th EuroSPI 2003 (European Software Process Improvement Conference), Graz, Austria, 10-12.12.2003.

Siakas K.V., **Georgiadou, E.** (2005): PERFUMES: A Scent of Product Quality Characteristics, The 13th Software Quality Management International Conference, SQM 2005, March 2005, Glouchestershire, UK.

Georgiadou E., Karakitsos G., Sadler, C., Stasinopoulos, D. (1993) *"An experimental examination of the role of re-engineering in the management of software quality*", 1st Software Quality Management International Conference II Vol., Computational Mechanics Publications, 1993.

Georgiadou, E., Karakitsos G., Sadler C.*, (1994) "Improving the program quality by using the re-engineering factor metric ρ",* The 10th. International Conference of the Israel Society for Quality, Jerusalem, November 1994.

Georgiadou, E., Karakitsos, G., Sadler C., Stasinopoulos D, Jones, R. (1994) *Program maintainability is a function of structuredness,* 2nd International Software Quality Management, Computational Mechanics Publications, Edinburg, Scotland, August 1994.

Georgiadou, E., Karakitsos, G., Sadler C., Stasinopoulos D, Jones, R. (1994) "*Program maintainability is a function of structuredness",* 2nd Software Quality

Management International Confreence, Computational Mechanics Publications, Edinburg, Scotland, August 1994.

Georgiadou, E., Keramopoulos, E. (2001) "*Measuring the Understandability of a Graphical Query Language through a Controlled Experiment"*, 9th International Conference on Software Quality Management, SQM 2001, April 2001, University of Loughborough, UK.

Georgiadou, E., (2007) "*A framework for the design and execution of controlled experiments in Software Engineering", S*oftware Quality Management International Conference, Special 50th Anniversary of the BCS, Tampere, Finland, 2007.

Georgiadou, E., Siakas, K. "*VALO$_5$ (2013) – Innovation, Maturity Growth, Quality and Valorisation"*, Systems, Software and Services Process Improvement Systems, Software and Services Process Improvement Communications in Computer and Information Science, Springer, Volume 364, 2013, pp 294-299.

Georgiadou, E., White, A., Comley, R. (2017) " *CoFeD: A Visualisation Framework for Comparative Quality Evaluation"*, in Achieving Software Quality in Development and in Use, P Marchbank, M Ross, G Staples (eds), 25th Software Quality International Conference, 2017.

Georgiadou, **E.,** (2018), "*Is the Composite Software Metric **ρ (rho)** a Good Metric?" in* Computing and Quality, 26th Software Quality International Conference, 2018.

Georgiadou, E. (2009**)** **"***Navigating the labyrinth of software Re-words"***,** 17th Software Quality Management International Conference, Southampton, UK, April 2009.

Georgiadou, E. (2018) *"Reflections on the need for Disambiguation of Terminology for software Process Improvement",* EuroSPI 2018, in Systems, Software and Services Process Improvement, Volume 896, Communications in Computer and Information , Springer, 2018 (in print).

Georgiadou, E. & George, C. (2006) *"Information Systems Failures: Whose responsibility?"* Proceedings of the 11th INternational Conference on Software Process Improvement - Research into Education and Training, (INSPIRE 2006), April, Southampton, UK, ISBN 1-902505-77-8.

Gilb, T. (1981) Evolutionary development, SigSoft, ACM SIGSOFT, Software Engineering Notes, Volume 6 Issue 2, pp 17-17.

Gillies, A.C. (1992), "Software Quality: Theory and Measurement", International Thompson Computer Press.

Grady, R., Caswell, D. (1987). Software Metrics: Establishing a Company-wide Program. Prentice Hall.

Hall, T., Jagielska, D., Baddoo, N. (2007) Motivating developer performance to improve project outcomes in a high maturity organization, Software Quality Journal, Vol 15, Issue 4, pp. 365-381.

Hirscheim, R.A., (1982), Information Systems Epistemology: A Historical Perspective, IFIPWG82 [https://ifipwg82.org/sites/ifipwg82.org/files/Hirschheim_0.pdf (accessed 15/07/2018)].

Holcombe, M., Ipate, F. (1998) Correct Systems: Building a Business Process Solution. Springer-Verlag, London
Howell, K.E., (2013), An Introduction to the Philosophy of Methodology, Sage Publications, London 2013.

Hyatt, L.E. and Rosenberg, L.H., (1996), A software quality model and metrics for identifying project risks and assessing software quality, 1996.

IEEE Standard 1061 (1998) IEEE Standard for a Software Quality Metrics Methodology, Software Engineering Standards Committee of the IEEE Computer Society.

ISO (2010) ISO/IEC/IEEE 24765:2010 (©ISO/IEC 2010 & © IEEE 2010), Published by ISO in 2011, Switzerland.

ISO 9001 (1994), 2nd Edition: Quality Systems: model for quality assurance in design/development, production, installation, ans servicing, International Organization for Standardization, Geneva.

ISO (2001) Technical Committee ISO/IEC 9126-1:2001 Software engineering -- Product quality -- Part 1: Quality model.

ISO (2014): ISO/IEC 25000: 2014, Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) – Guide to SQuaRE.

Jacobson I., Pan-Wei Ng, McMahon P., Spence I., Lidman S. (2013). The Essence of Software Engineering—Applying the SEMAT Kernel. Addison-Wesley.

Jackson, M. (1994) Problems, methods and specialization, Software Engineering Journal, Volume 9, Issue 6, November 1994, p. 249 – 256, Online ISSN 2053-910X.

Jalote, P., (2000) CMM in Practice: Processes for Executing Software Projects at Infosys. Addison Wesley Longman, 2000.

Jayaratna, N (1994), "Understanding and Evaluating Methodologies", NIMSAD: ASystemic Approach, McGraw-Hill.

Jones, C. (2010) cited by Dalcher (2014) Software engineering best practices:lessons from successful projects in the top companies. McGraw Hill, New York.

Juran, J. M., Blanton, Godfrey, A. (1999), Juran's Quality Handbook , McGraw-Hill.

Kaposi, A., Pyle, I. (1993) Systems are not only software, Software Engineering Journal, Volume 8, Issue 1, January 1993, p. 31 – 39.

Kirsch, B. I., Sauberer, G. (2011), Terminological Precision - A Key Factor in Product Usability and Safety, in Design, User Experience and Usability, Theory, Methods, Tools and Practice, Orlando, Florida, USA, DUXU 2011.

Kitchenham, B. (1996), Software Metrics, Measurement for Software Process Improvement, NCC, Blackwell.

Kitchenham, B., and Pfleeger, S., (1996), software quality — The elusive target, IEEE Software, January (1996) pp. 12- 21.

Kulkarni, V. (2016), Model Driven Development of Business Applications – a Practitioner's Perspective, IEEE/ACM 38th IEEE International Conference on Software Engineering Companion (ICSE-C).

Law, D., and Naeem, T., (1992), `DESMET: Determining and Evaluation methodology for Software MEthods and Tools', *Proceedings of BCS Conference on CASE - Current Practice, Future Prospects*, Cambridge, England, March 1992.

Lee, A. (1991). Integrating positivist and interpretivist approaches to organizational research. *Organ. Sci.* **2** 342–365.

Lehman, M.M. (1998) Software's Future: Managing Evolution, IEEE Software, Vol 15, No 1, 1998, 40-44.

Lehman, M.M., (1980), Programs, life cycles, and laws of software evolution. Proc. IEEE 68 (9), 1060–1076.

Lehman, M.M., Ramil, J.F., Wernick, P.D., Perry, D.E., Turski, W.M., (1997), Metrics and laws of software evolution - the nineties view. In: Proceedings of the 4th International Symposium on Software Metrics. IEEE Computer Society, Washington, DC, USA, p. 20.

Liebowitz, J., (2015), IT Project Failures: What Management Can Learn, IT Professional, Nov.-Dec. 2015, Vol. 17(6), pp.8-9.

Logothetis, N. and Wynn, H.P. (1989) Quality through Design: Experimental Design, Off-Line Quality Control and Taguchi's Contributions. Clarendon Press, Oxford.

McCall, J. A. &, Richards, P. K. & Walters, G. F. (1977). Factors in Software Quality, US Rome Air Development Center Reports, US Department of Commerce, USA.

Miguel, J.P., Mauricio, D. , Rodriguez, G. (2014) A Review of Software Quality Models for the Evaluation of Software Products, International Journal of Software Engineering & Applications (IJSEA), Vol.5, No.6, November 2014 , DOI : 10.5121/ijsea.2014.5603 31.

Naumann, J.D., Jekins, M., (1982), Prototyping: The New Paradigm for Systems

Development, *MIS Quarterly,* Vol. 6, No. 3 (Sep., 1982), pp. 29-44.

Orlikowski, W.J., Baroudi, J.J. (1991) Studying information technology in organizations: research approaches and assumptions. Information Systems Research 2(1), 1–28.

Pfleeger-Lawrence, S. (1998), Software Engineering: Theory and Practice, Prentice Hall.

Polymenakou, A, Serafeimidis V. (1995), Unlocking the Secrets of Information Systems Failures: the Key Role of Evaluation", 5th Panhellenic Informatics Conference, Proceedings Vol II, Athens, Greece.

Pressman, R., S; Maxim, Bruce, R. (2015) Software engineering: a practitioner's approach, McGraw-Hill Education, New York (8th ed., International student).

Rahanu, H., Georgiadou, E., Siakas, K.V., Ross, M. (2018) Imperative Ethical Behaviours in Making Systems Development and Deployment Compliant with Health & Safety and Wellbeing EuroSPI 2018, in Systems, Software and Services Process Improvement, Volume 896, Communications in Computer and Information , Springer, 2018 (in print).

Remenyi, D., Williams, B. (1996), The nature of research: qualitative or quantitative, narrative or paradigmatic? Information Systems Journal.
[https://doi.org/10.1111/j.1365-2575.1996.tb00009.x (accsssed 15/07/2018)].

Rokeach, M. (1973), The nature of human values, Free Press, Collier-Manmillan, Lomdon.

Sarkar, S. (1996) (Ed.), The emergence of logical empiricism: from 1900 to the Vienna circle, Garland Publishing. New York.

Sauberer, G., Villar, N., Blanca and Drebler, Jens and Schmitz, K-D., Clarke, P, O'Connor, R. (2017) Do we speak the same language?: terminology strategies for (software) engineering environments based on the elcat model - innovative terminology e-learning for the automotive industry. In: 24th European Conference on Systems, Software and Services Process Improvement (EuroSPI 2017), 6-8 Sept 2017, Ostrava, Czech Republic. ISBN 978-3-319-64218-5.

Shepperd, M.J. (1990), 'An empirical study of design measurement', Softw. Eng. J., 5(1), pp3-10, 1990.

Shewhart W. A., (1986) Statistical method from the viewpoint of quality control, Mineola, NY: Dover Publications, 1986.

Siakas, K. V., Berki, E., Georgiadou, E., Sadler, C. (1997): The Complete Alphabet of Quality Software Systems: Conflicts and Compromises, 7th World Congress on Total Quality & Qualex 97, New Delhi, India, 17-19 February.

Sommerville, I. (2016) Software Engineering (10th edition), Pearson Education Limited , Boston, 2016.

Taguchi, G. (1985), "Quality Engineering in Japan", Bulletin of the Japan Society of Precision Engineering, Vol 19 No (4), pp. 237-242, (1985).

Taguchi, G. (1986), “Introduction to Quality Engineering - Designing Quality into Products and Processes”, Asian Productivity Organization, Tokyo, (1986).

Vandierendonck, H., Mens, T. (2011), Averting the Next Software Crisis, Computer, 04/2011, Volume 44, Issue 4.

Weyuker, E. (1988) “Evaluating Software Complexity Measures,” IEEE Trans. SoftwareEng., vol. 14, no. 9, pp. 1357-1365

White, A.S, (2006), ‘External Disturbance control for software project management’, Int. J Project Management, 24, 127-135.

White, S.A. (2013) A control model of the software requirements process, Kybernetes, Volume: 42 Issue: 3, 2013.

Whitmire, S.A. (1997): Object-Oriented Design Measurement, John Wiley & Sons Inc., 1997.

Winston, P. (2008), New, improved airport: Same old lost luggage; Business Insurance, June 30, 2008, Vol. 42(26), p.6.

APPENDICES

Appendix A

Glossary of terms pertinent to this research

Term	Definition	Source
Maintainability	*is the ease with which a software system (program) or component can be maintained (i.e. modified to change or add capabilities, correct errors, faults and/or defects, to improve performance or other behaviour, or adapt to a changed environment).*	*Proposed by the author*
Measurement	is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules	Finkelstein, 1984
Methodology	is a system of practices, techniques, procedures, and rules used by those who work in a discipline	ISO/IEC/IEEE 24765:2010
Process	is a set of activities that begin with the identification of a need and conclude with the retirement of a product that satisfies the need; or more completely, as a set of activities, methods, practices, and transformations that people use to develop and maintain software and its associated products (e.g. project plans, design documents, code, test cases, user manuals).	Whitmire, 1997

Term	Definition	Source
Process Improvemnet	are actions taken to change an organisation's processes so that they more effectively and/or efficiently meet the organization's business goals.	**ISO/IEC/IEEE 24765:2010**
Reengineering	the examination and alteration of software to reconstitute it in a new form, including the subsequent implementation of the new form.	**ISO/IEC/IEEE 24765:2010**
Resources	are people, tools, materials, methods, time, money, training (or generally knowledge and skill) and products from other projects	Whitmire, 1997
SoftwareMetric	*is a measurable property which is an indicator of one or more of the quality attributes.*	*Proposed by the author*
Software products or artifacts	are the products (deliverables/outputs of the software process. These products may be plans, functional specifications, process models, procedure manuals, coding, test data, test results and so on	Whitmire, 1997
Software project	is the relationship between instances of a problem to be solved, internal and external goals and standards, processes, methods and techniques, constraints and finally a product (one or more deliverables)	Whitmire, 1997
Software Quality	*is the totality of product as well as process characteristics, and their interaction and measurement (whether qualitative or quantitative) that satisfy different stakeholder requirements.*	*Proposed by the author*

Term	Definition	Source
Structuredness	is a factor connected to a low probability of errors	Oulsnam, 1982
Un-structuredness	is a factor connected to a high higher probability of errors	Oulsnam, 1982
Valorisation	is the process of value creation from knowledge, by making it applicable and available for economic or societal utilisation, and by translating it in the form of new business, products, services, or processes. It includes dissemination and exploitation of results.	VALO project
Value	is an enduring belief that a specific mode of conduct or end-state is personally or socially preferable to an opposite or converse mode of conduct or end-state of existence.	Rokeach (1973)
Usability	is the capability of the software product to be understood, learned and liked by the user, when used under specific conditions.	Georgiadou & Sheriff (2008)

Appendix B

Access Guide for Submitted Papers

Paper I	eprints.mdx.ac.uk
Paper II	https://link.springer.com/article/10.1023/A:1023833428613
Paper III	https://link.springer.com/article/10.1007/s11219-007-9029-0
Paper IV	https://link.springer.com/article/10.1023/A:1025817312035
Paper V	eprints.mdx.ac.uk
Paper VI	https://pdfs.semanticscholar.org/7415/766cdae5f9233708fe0e87f530cd31f58f61.pdf
Paper VII	eprints.mdx.ac.uk
Paper VIII	eprints.mdx.ac.uk
Paper IX	eprints.mdx.ac.uk
Paper X	eprints.mdx.ac.uk
Paper XI	eprints.mdx.ac.uk
Paper XII	eprints.mdx.ac.uk
Paper XIII	https://pdfs.semanticscholar.org/5080/8b77c65e0c5960c3f89eb56b195bc469495f.pdf
Paper XIV	https://link.springer.com/chapter/10.1007/978-3-642-39179-8_26
Paper XV	eprints.mdx.ac.uk
Paper XVI	eprints.mdx.ac.uk
Paper XVII	eprints.mdx.ac.uk
Paper XVIII	https://link.springer.com/chapter/10.1007/978-3-319-97925-0_49
Paper XIX	eprints.mdx.ac.uk
Paper XX	eprints.mdx.ac.uk