Title:	Software Process Engineering APPROVED
Long Title:	Software Process Engineering
Module Code:	COMP9055
Credits:	5
NFQ Level:	Expert
Field of Study:	Computer Software
Valid From:	Semester 1 - 2017/18 (September 2017)
Module Delivered in	1 programme(s)
Module Coordinator:	TIM HORGAN
Module Author:	PAUL WALSH
Module Description:	Traditionally, software architects were responsible for the design and technological decisions of the software development process. Over the past decade, the traditional role of the software architect has changed. Agile development methodologies has meant the adoption of tools and practices that help avoid or decouple up front systems or architectural design. Agile teams focus on smaller local decisions, restructuring the system through refactoring maintaining the development pace. In this module, students will be introduced to the main software development methodologies and development operations (DevOps) processes designed to support the development of software and software based systems. As part of this module, students will examine the challenges that architects, developers and testers face in this environment and strategies and best practices to minimise risk and maximise the quality of the software produced.

Learning Outcomes	
On successful completion of this module the learner will be able to:	
LO1	Discuss how software process engineering concepts and techniques can help an organisation achieve its strategic objectives.
LO2	Critically assess how the traditional role of a software architect has been impacted by software development process such as agile and technological changes.
LO3	Build and refine a product backlog that effectively records and manages technical debt as part of an agile development methodology.
LO4	Evaluate and apply best practices to the refactoring of code as part of a Test Driven Development (TDD) approach to the development of software.
LO5	Create a Development and Operations (DevOps) pipeline that considers test automation as part of its Continuous Integration (CI).

Pre-requisite learning
Module Recommendations *This is prior learning (or a practical skill) that is strongly recommended before enrolment in this module. You may enrol in this module if you have not acquired the recommended learning but you will have considerable difficulty in passing (i.e. achieving the learning outcomes of) the module. While the prior learning is expressed as named CIT module(s) it also allows for learning (in another module or modules) which is equivalent to the learning specified in the named module(s).*
No recommendations listed
Incompatible Modules *These are modules which have learning outcomes that are too similar to the learning outcomes of this module. You may not earn additional credit for the same learning and therefore you may not enrol in this module if you have successfully completed any modules in the incompatible list.*
No incompatible modules listed
Co-requisite Modules
No Co-requisite modules listed
Requirements *This is prior learning (or a practical skill) that is mandatory before enrolment in this module is allowed. You may not enrol on this module if you have not acquired the learning specified in this section.*
No requirements listed
Co-requisites
No Co Requisites listed

Module Content & Assessment

Indicative Content
Software Development Processes Activities - requirements, architecture, implementation, testing & maintenance. Process models - Waterfall, prototyping, iterative, rapid, structured, object-oriented, agile. Advantage and disadvantages of different process models.
Agile Principles of agile methodology and the agile manifesto. Main methodologies – Extreme Programming (XP), Scrum, Feature Driven Development (FDD), Lean Software Development, Kanban, Scrum. Practices in Agile – modelling, Test-Driven Development, Rapid Application Development (RAD), Pair Programming. Development-Operations (DevOps) – tool chain. Automated deployment. Source code management, versioning.
Project Management Traditional project management approaches. Agile Project planning, Agile project life cycle.- Self organising teams. Project Management functions. Scrum - theory, tram, scrum master, sprint and planning, product backlog.
Development Operations What is DevOps? Deployment Architecture- design for continuous delivery and integration, build automation, test management; release management, cross platform support, extensibility, source control management, versioning, dependency management. Impact of cloud architecture and microservices in DevOps. DevOps monitoring - important metrics. DevOps Security - STRIDE model, identity management and security issues with VMs and cloud architectures.
Architect role in Agile and DevOps Role of software architect in Agile - requirements and definition of product backlog, teamwork, dealing with design issues, balancing the requirements and work, interaction with developers, product owner and testers. Responsibility of architect in DevOps - automated testing, deployment pipelines, continuous integration, subject matter expert in best practices in DevOps, review, auditing, technical documentation.
Software Cost Estimation Cost estimation tools, model based estimation techniques. Expertise based approaches - when it is useful, limitations etc Agile sizing approaches - numeric sizing, t-shirt sizes, the Fibonacci sequence, dog breeds. Factors that affect effort - story size, complexity, velocity, development cost etc.
Technical Debt What is technical debt? Classification of technical debt. Dealing with technical debt in legacy projects. Philosophies in dealing with technical debt. Technical debt quadrant – reckless, prudent, inadvertent. Strategies and tactics for avoiding and removing technical debt. Software quality versus business reality. Measuring debt. Communicating debt consequences.
Code Refactoring Definition. Origins. Common pitfalls. Benefits of code refactoring. Relationship to Test Driven Development (TDD). Code duplication. Identification where to apply which refactoring. Guaranteeing that refactoring preserves software behaviour. Assessing impact of refactoring on quality. Refactoring techniques. Formalisms techniques - program slicing, formal improvement theory, software metrics etc. How code refactoring fits into software engineering process. Regression testing.
Automated testing Benefits and limitations of automated testing. Role of tester and developer. System tests, unit tests. Challenges in automated tests for testers and developers. Test design and analysis. Test execution and review. Tools i.e. Selenium and other testing frameworks.

Assessment Breakdown	%
Course Work	100.00%

Course Work

Assessment Type	*Assessment Description*	*Outcome addressed*	*% of total*	*Assessment Date*
Written Report	An example assessment would be to evaluate how software development methodologies and DevOps has impacted the development of software and the role of the architect as part of the process.	1,2	30.0	Week 8
Project	In this project the student would be expected to build and refine a product backlog that accounts for technical debt. They would also be expected to build a pipeline that supports the continuous integration of code and supports automated testing. A written report may also be required as part of the submitted project.	1,2,3,4,5	70.0	Sem End

No End of Module Formal Examination

Reassessment Requirement

Coursework Only
This module is reassessed solely on the basis of re-submitted coursework. There is no repeat written examination.

The institute reserves the right to alter the nature and timings of assessment

Module Workload

Workload: Full Time				
Workload Type	*Workload Description*	*Hours*	*Frequency*	*Average Weekly Learner Workload*
Lecture	Lecture underpinning learning outcomes.	2.0	Every Week	2.00
Lab	Lab supporting content delivered in class.	2.0	Every Week	2.00
Lecturer-Supervised Learning (Contact)	Independent study.	3.0	Every Week	3.00
			Total Hours	7.00
			Total Weekly Learner Workload	7.00
			Total Weekly Contact Hours	7.00

Workload: Part Time				
Workload Type	*Workload Description*	*Hours*	*Frequency*	*Average Weekly Learner Workload*
Lecture	Lecture underpinning learning outcomes.	2.0	Every Week	2.00
Lab	Lab supporting content delivered in class.	2.0	Every Week	2.00
Independent & Directed Learning (Non-contact)	Independent study.	3.0	Every Week	3.00
			Total Hours	7.00
			Total Weekly Learner Workload	7.00
			Total Weekly Contact Hours	4.00

Module Resources

Recommended Book Resources

- Len Bass 2015, ***DevOps: A Software Architect's Perspective***, 1 Ed., Addison-Wesley Professiona [ISBN: 9780134049847]

Supplementary Book Resources

- Kim, Gene, Behr, Kevin, & Spafford, George 2013, ***The phoenix project: A novel about IT, DevOps, and helping your business win***, IT Revolution Press [ISBN: 9780988262591]
- Gerritt Beine 2017, ***Technical Debts: Economizing Agile Software Architecture***, Walter de Gruyter [ISBN: 9783110462999]
- Martin Fowler, Kent Beck, John Brant, William Opdyke, Don Roberts, Erich Gamma 1999, ***Refactoring: Improving the Design of Existing Code***, Addison-Wesley Professional [ISBN: 9780201485677]
- Robert C. Martin 2008, ***Clean Code: A Handbook of Agile Software Craftsmanship***, Prentice Hall [ISBN: 9780132350884]
- Len Bass, Ingo Weber, Liming Zhu 2015, ***DevOps: A Software Architect's Perspective***, SEI Series in Software Engineering [ISBN: 9780134049847]

Recommended Article/Paper Resources

- P. Kruchten, R. L. Nord and I. Ozkaya 2012, ***Technical Debt: From Metaphor to Theory and Practice***, IEEE Software, 29(6) http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6336722&isnumber =6336715
- B. Curtis, J. Sappidi and A. Szynkarski 2012, ***Estimating the Principal of an Application's Technical Debt***, IEEE Software, 29(6)
- E. Lim, N. Taksande and C. Seaman 2012, ***A Balancing Act: What Software Practitioners Have to Say about Technical Debt***, IEEE Software, 29(6)
- T. Mens and T. Tourwe 2004, ***A survey of software refactoring***, IEEE Transactions on Software Engineering, 30(2)
- 2014 ***Introducing DevOps to the Traditional Enterprise***, InfoQ, June http://miroslawdabrowski.com/downloads/DevOps/Introducing%20DevOps%20to%20the%20 Traditional%20Enterprise.pdf
- M. Callanan and A. Spillane 2016, ***DevOps: Making It Easy to Do the Right Thing***, IEEE Software, 33(3) http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7436644&isnumber =7458753

Other Resources

- Website: ***What role do architects have in agile development?*** http://www.ben-morris.com/what-role-do-a rchitects-have-in-agile-development/
- Website: ***The Role of the Agile Architect*** https://agilescout.com/the-role-of-the-a gile-architect/
- Website: ***The Architecture Owner Role: How Architects Fit in on Agile Teams*** http://www.agilemodeling.com/essays/arch itectureOwner.htm#sthash.aKHWJL9Z.dpuf
- Website: ***Technical Debt*** http://www.construx.com/10x_Software_Dev elopment/Technical_Debt/
- Website: ***Technical Debt: Strategies & Tactics for Avoiding & Removing it*** http://blogs.ripple-rock.com/SteveGarnet t/2013/03/05/TechnicalDebtStrategiesTact icsForAvoidingRemovingIt.aspx
- Website: ***TechnicalDebtQuadrant*** https://martinfowler.com/bliki/Technical DebtQuadrant.html
- Website: Agile Alliance***Project Management and Technical Debt*** https://www.agilealliance.org/project-ma nagement-and-technical-debt/
- Website: Agile Alliance***Refactoring*** https://www.agilealliance.org/glossary/r efactoring/
- Website: ***Agile, Scrum and the nonsense about architects*** https://blog.codecentric.de/en/2013/11/a gile-scrum-nonsense-architects/

Module Delivered in

Programme Code	Programme	Semester	Delivery
CR_KSDEV_9	Master of Science in Software Architecture & Design	1	Mandatory