Office Use Only

MONASH University

Semester Two 2017
Examination Period

Faculty of Information Technology

EXAM CODES: **FIT2101**

TITLE OF PAPER: **SOFTWARE ENGINEERING PROCESS AND MANAGEMENT**
SAMPLE EXAM MARKING SCHEME

EXAM DURATION: 2 hours writing time

READING TIME: 10 minutes

THIS PAPER IS FOR STUDENTS STUDYING AT: (tick where applicable)

☐ Berwick	☑ Clayton	☑ Malaysia	☐ Off Campus Learning	☐ Open Learning
☐ Caulfield	☐ Gippsland	☐ Peninsula	☐ Monash Extension	☐ Sth Africa
☐ Parkville	☐ Other (specify)			

During an exam, you must not have in your possession any item/material that has not been authorised for your exam. This includes books, notes, paper, electronic device/s, mobile phone, smart watch/device, calculator, pencil case, or writing on any part of your body. Any authorised items are listed below. Items/materials on your desk, chair, in your clothing or otherwise on your person will be deemed to be in your possession.

No examination materials are to be removed from the room. This includes retaining, copying, memorising or noting down content of exam material for personal use or to share with any other person by any means following your exam.
Failure to comply with the above instructions, or attempting to cheat or cheating in an exam is a discipline offence under Part 7 of the Monash University (Council) Regulations.

AUTHORISED MATERIALS

OPEN BOOK	**☑ YES**	**☐ NO**
CALCULATORS	**☐ YES**	**☑ NO**
SPECIFICALLY PERMITTED ITEMS	**☐ YES**	**☑ NO**

if yes, items permitted are:

Candidates must complete this section if required to write answers within this paper

STUDENT ID: __ __ __ __ __ __ __ __ DESK NUMBER: __ __ __ __ __

This page intentionally left blank

This exam is 12 pages long and contains 7 questions. It is worth 62 marks in total.

Please write your answers in the script book provided. If you run out of paper, you may request another script book. You may answer the questions in any order, but please make sure you number each question clearly. To maximize your mark, we suggest that you attempt the questions you find easiest first.

Each question that requires a written answer has a suggested length. This suggestion is only there to help you understand how much detail we're expecting – there is no penalty for writing more or less than the suggested length provided your answers are clear and complete.

Your markers request that you not use red pen.

Question 1 (2 + 3 + 10 = 15 marks)

This question is about **risks**.

a) In your own words, what is a *technical risk*?

A risk that arises from technical aspects of the project (this is enough for 2/2; 1 if correct but vague, or if incorrect but on the right track; 0 if this exact phrasing – it's taken from the lecture slides)

b) Name a technique for mitigating technical risk, and explain how it works in one or two paragraphs.

Name a technique: 1 mark
Explanation: 2 marks

The one we covered in lectures is spiking. It reduces risk by helping teams understand new features, practices, libraries etc. – essentially, somebody on the team writes a minimum end-to-end proof-of-concept that uses the new technology, then documents it at a level that will help the rest of the team understand how it works. Other techniques are fine if they are relevant and the student has understood/described them adequately.

c) Here is part of a risk register belonging to an inexperienced Agile team. List the problems that this risk register has, and suggest ways that the team might fix them.

Risk	Likelihood	Impact	Mitigation
Data centre flooded, all data is lost	Medium	‡	Take weekly backups. Restore from backups if data is lost.
Team member gets sick, leaves project	Medium	Δ	Renegotiate schedule with client
Team member wins lottery, leaves project	0.0001	‡	Renegotiate schedule with client
Data centre burns down, all data is lost	Low	Δ	Take weekly backups. Restore from backups if data is lost.
Program doesn't work	Medium	Δ	Avoid writing buggy code. Fix program if it stops working.
Zombie apocalypse, civilization breaks down	0		Renegotiate schedule with client
New developer on project doesn't know Java	Already happened	Δ	Supply new developer with Java books and reevaluate performance periodically

What are the good and bad points of this risk register? What could the team do to improve it? Write no more than one page.

Good points: most of the mitigation strategies are good and likely to work. All the components are there: description, likelihood and impact, and mitigation.

Bad points:

- Likelihood scale is inconsistent; make it consistent
- Impact scale is impossible to interpret; include a key or use normal words
- "Zombie apocalypse" impact is missing
- "Zombie apocalypse" mitigation is laughably inadequate
- "Zombie apocalypse" likelihood is zero (should drop it from the register)
- No real point in distinguishing between flood and fire in the destruction of the data centre: replace with "data centre destroyed, all data is lost" and leave mitigation as-is
- "Program doesn't work" is an outcome, not a risk: consider what may happen to cause the program not to work, and address each potential root cause separately (e.g. misunderstood requirements, changing technical environment, lack of technical knowledge in the team, etc.)…
- …and while we're there, "avoid writing buggy code" is bleeding obvious (and what all good developers should be doing anyway). Give concrete strategies that are likely to help. It will be easier to do this when the risk is broken down by cause.
- "New developer" mitigation might not be sufficient – could give the newbie a mentor, but…
- …on the other hand, anything that's "already happened" isn't a risk, it's an issue – mitigations need to go in the PMP and other project governance documentation

2 marks for good points (1 per point to a max of 2).
8 marks per bad points (1 for stating the problem, 1 for suggesting an improvement, to a max of 8)

Deduct 0.5 mark per point that is vague.

Question 2 (1 + 3 + 2 = 6 marks)

*This question is about **requirements**.*

User stories can be evaluated using the INVEST criteria. These criteria say that user stories should be "Independent, Negotiable, Valuable, Estimable, Small, and Testable".

a) In your own words, what does it mean to say that user stories should be *independent*?

They shouldn't depend on any other user stories. (1 mark.)

b) What are the potential negative consequences of having user stories that are *not* independent?

Because user stories are negotiable, they are able to be changed, reinterpreted, reprioritized etc. as customer priorities or needs change. (1 mark) If user stories are dependent on each other, then changing one means changing all its dependencies. (2 marks)

Other answers are possible, e.g. answers that talk about the order in which user stories are implemented. Pay full marks if reasonable and correct; partial marks if at least defensible – but the main idea is that *dependencies make user stories hard to change.*

c) Give an example of one or more user stories that are *not independent*.

It's not really possible to do this with only one user story. ☺

2 marks if stories are genuinely not independent. 1 mark if they're on the right track but don't quite get there.

Question 3 (5 + 5 = 10 marks)

*This question is about **process models**.*

Here is a description of a software process given by a team member:

"First, our Product Owner sits down with the client and end users, and they spend about a month sorting out a list of user stories to make sure that it's complete. After that, the team gets together and designs the system so that we know which modules and functions need to be written and what their inputs and outputs will be. Then we start to iterate.

"Our sprints are two weeks long and each developer is expected to implement at least one method in each sprint. We keep doing sprints until all the functions we'd planned have been implemented, and then we have one final sprint which has been set aside for testing. After that, we have a retrospective. So our process is a pretty standard version of Scrum."

a) Is the developer correct that this process is standard Scrum? Which features of this process are similar to Scrum and which are different? Write about half a page.

This isn't standard Scrum. (1 mark)

Scrum features:

- there's a Product Owner who is responsible for representing the client's understanding of the requirements
- user stories
- iterations/sprints of about the right length
- testing happens
- there is a retrospective

Non-Scrum features:

- big up-front requirements effort
- big up-front design effort
- no scrum planning
- scrum goals measured in methods instead of user-visible features
- no guarantee that iterations result in deployable code/user value
- client representatives not involved in prioritization
- testing left to the end
- one retro per project instead of per sprint

1 mark per valid point to a maximum of 4. Deduct half a mark per point that is vague.

b) Do the variations from Scrum that you identified in the previous question introduce extra risks to the success of the project? Do they reduce any risks? Write about half a page.

Most introduce extra risks, especially late testing and reduced process improvement. Big up-front requirements reduce the likelihood of scope creep, but at the cost of no longer being able to deal with changes in requirements: if the requirements were misunderstood or change as a result of changes in the business or technical environments, the problem won't be discovered until after delivery.

The answer to this question will depend on the changes identified in the previous one. Give 1 mark per reasonable point to a max of 5, but 0 marks to evaluations that don't match the variations identified in part a.

Question 4 (2 + 1 + 2 = 5 marks)

*This question is about **process improvement**.*

a) In your own words, what is the difference between a *retrospective* and a *sprint review*?

A retrospective looks at the process, a sprint review looks at the product. (Give 1 mark if students are vague about which one is which; 0.5 if they're obviously wrong.)

b) In our own words, what is *velocity*?

How much work a team can get done in a single sprint. (This is sufficient for 1 mark. Nothing for answers copied from the lecture slides.)

c) We said in lectures that it was important for retrospectives to focus particularly on differences between actual and expected velocity. Why is this important? Write about half a page.

If your velocity is lower than expected, then either you're slower than you thought you'd be or you're not good at estimating. Either of those issues might reflect a deeper problem that you should address. On the other hand, if it's higher, then you might be overestimating your stories, and you'll either end up picking out extra stories on an ad-hoc basis or wasting time at the end of the sprint because you've run out of work to do.

Other answers are possible; pay if plausible and genuinely potentially problematic. 1 mark per valid point to a max of 2.

Question 5 (6 marks)

*This question is about **software metrics**.*

Imagine a large company that is organized into several Scrum teams. A senior executive suggests paying a bonus to the team that averages the highest velocity each year. What do you think of this proposal, and why? Write about half a page.

Likely points:

This proposal introduces a perverse incentive, because teams might do what the executive presumably wants (i.e. work harder) but might also be tempted to overestimate their stories.

Story estimation is done by the team and isn't an exact science, so you can't directly compare one team's story estimations to another's. Since velocity is story points per iteration, that means that velocities aren't comparable either. Velocities are only comparable *within* a team.

This proposal is likely to damage relationships between teams – if any teams in the company are collaborating, they now have an incentive to sabotage one another.

This proposal could harm morale if it makes the workplace feel more hostile.

6 marks: identifies at least two relevant points, explains them well, clearly articulated conclusion follows logically from discussion of points.

4 marks: two points discussed well but conclusion absent or trivial; or only one point with matching conclusion.

2 marks: a relevant point is mentioned.

If torn between two marks, give the average (e.g. if you can't decide between a 4 or a 6, it's a 5.)

Question 6 (5 + 5 = 10 marks)

*This question is about **Agile software development practices**.*

a) One of the principles behind the Agile Manifesto is to "*deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale*". What advantages can a team gain by delivering working (but unfinished) software to the client frequently, as opposed to delivering only the finished product when development is complete? Are there any disadvantages to frequent delivery? Write about half a page.

Advantages include:

- quick feedback on misinterpreted requirements
- faster turnaround when fixing
- no need to document issues in great detail as the team can improve the system immediately
- changes required are likely to be small and tractable rather than large and expensive

Disadvantages include:

- requires client to be available frequently to give feedback
- can take up a significant amount of developers' time

Other answers are possible. 1 mark per valid point to a maximum of 5 (0.5 if vague).

b) The Agile Manifesto says that Agile practitioners value "*working software over comprehensive documentation*". What effect does this have on the practices that Agile developers adopt, compared to the practices of more traditional software developers? Write about two paragraphs.

- Agile developers document requirements in much less detail because client representatives are present and can explain the requirements in person
- Design is documented as and when needed rather than producing a complete design
- Documentation in general is kept small enough to be easy to keep up to date as requirements change
- Planning is mostly done on a sprint-to-sprint basis, which avoids the need to redo the schedule when requirements change
- Agile developers are better able to deal with requirements change because less documentation will need to change alongside the code

Other answers possible. 1 mark per valid point to a maximum of 5 (0.5 if vague).

Question 7 (10 marks)

*This question is about **choosing a process model**.*

Riparia is a small software company that specializes in creating software to help government agencies manage rivers and waterways. Their products allow river managers to record levels of pollutants, fish populations, introduced pest species, weeds, and other indicators of the health of the ecosystem.

Riparia is a very small company, comprising three experienced developers with at least seven years' experience programming in this field. It has been using ad-hoc methods to write software for local governments. However, they have recently won a large contract to provide software to the Parks and Wildlife Department of the national government. This has allowed them to hire two junior programmers and a third senior programmer, who is an excellent programmer but lacks experience with ecological monitoring.

The new software will mostly be used by rangers and fisheries officers, who spend most of their time in remote areas and often cannot be contacted by phone. One of the finance officers in the Parks and Wildlife department has volunteered to act as a contact person and pass on any questions that the team has for these end users. She says that she will usually be able to get a response to her email within three days.

Riparia now has to decide how to create this new software. One of the junior developers suggests adopting an Agile process model such as Scrum, the other suggests that a heavyweight process model like Spiral might be more suitable, and one of the experienced staff members suggests sticking to their existing ad-hoc approach.

What kind of process do you think Riparia should adopt, and why? Your answer should consider the potential risks and benefits of each of the alternatives: Agile, heavyweight, and ad-hoc. Write about one page.

Agile:

+ some experienced developers who understand the field
+ finance person can act as Product Owner
- experienced devs new to Agile
- new staff lack experience in the domain, may have trouble estimating
- hard to get hold of representative end users
- finance person doesn't sound like a "representative end user"

Ad-hoc:

+ current systems developed with it
+ existing staff used to it
- doesn't scale well (and new system has larger scope)
- team has doubled in size; more structured management will be needed

Heavyweight:

+ good fit for junior staff (requires less maturity)
+ likely to be easier to arrange requirements elicitation as a one-off

3 marks for considering *both* pros *and* cons of *each of* the three process models.

1 mark for choosing and clearly articulating a process model that is supported by the analysis.

I would probably go for a heavyweight model here. Lack of client input is a deal-breaker for Scrum and a finance person is unlikely to be fully aware of the day-to-day work of a park ranger or fisheries officer, and ad-hoc is unlikely to scale to this larger project and team. It's possible to argue for Scrum, too.