Software Engineering 10th edition Sommerville Solutions Manual

SKU: 9780133943030-SOLUTIONS

2 Software Processes

2.1 Giving reasons for your answer based on the type of system being developed, suggest the most appropriate generic software process model that might be used as a basis for managing the development of the following systems:

- A system to control anti-lock braking in a car
- A virtual reality system to support software maintenance
- A university accounting system that replaces an existing system
- An interactive travel planning system that helps users plan journeys with the lowest environmental impact

---

1. *Anti-lock braking system* This is a safety-critical system so requires a lot of up-front analysis before implementation. It certainly needs a plan-driven approach to development with the requirements carefully analysed. A waterfall model is therefore the most appropriate approach to use, perhaps with formal transformations between the different development stages.
2. *Virtual reality system* This is a system where the requirements will change and there will be an extensive user interface components. Incremental development with, perhaps, some UI prototyping is the most appropriate model. An agile process may be used.
3. *University accounting system* This is a system whose requirements are fairly well-known and which will be used in an environment in conjunction with lots of other systems such as a research grant management system. Therefore, a reuse-based approach is likely to be appropriate for this.
4. *Interactive travel planning system* System with a complex user interface but which must be stable and reliable. An incremental development approach is the most appropriate as the system requirements will change as real user experience with the system is gained.

2.3 Consider the integration and configuration process model shown in Figure 2.3. Explain why it is essential to repeat the requirements engineering activity in the process.

©Ian Sommerville 2014

Software Engineering 10th edition Sommerville Solutions Manual

You need to repeat the requirements engineering activity because it is essential to adapt the system requirements according to the capabilities of the system/components to be reused. These activities are:

1. An initial activity where you understand the function of the system and set out broad requirements for what the system should do. These should be expressed in sufficient detail that you can use them as a basis for deciding of a system/component satisfies some of the requirements and so can be reused.
2. Once systems/components have been selected, you need a more detailed requirements engineering activity to check that the features of the reused software meet the business needs and to identify changes and additions that are required.

2.4 Suggest why it is important to make a distinction between developing the user requirements and developing system requirements in the requirements engineering process.

There is a fundamental difference between the user and the system requirements that mean they should be considered separately.

1. The user requirements are intended to describe the system's functions and features from a user perspective and it is essential that users understand these requirements. They should be expressed in natural language and may not be expressed in great detail, to allow some implementation flexibility. The people involved in the process must be able to understand the user's environment and application domain.
2. The system requirements are much more detailed than the user requirements and are intended to be a precise specification of the system that may be part of a system contract. They may also be used in situations where development is outsourced and the development team need a complete specification of what should be developed. The system requirements are developed after user requirements have been established.

2.6 Explain why change is inevitable in complex systems and give examples (apart from prototyping and incremental delivery) of software process activities that help predict changes and make the software being developed more resilient to change.

Systems must change because as they are installed in an environment the environment adapts to them and this adaptation naturally generates new/different

©Ian Sommerville 2014

Software Engineering 10th edition Sommerville Solutions Manual

system requirements. Furthermore, the system's environment is dynamic and constantly generates new requirements as a consequence of changes to the business, business goals and business policies. Unless the system is adapted to reflect these requirements, its facilities will become out-of-step with the facilities needed to support the business and, hence, it will become less useful.

Examples of process activities that support change are:

1. Recording of requirements rationale so that the reason why a requirement is included is known. This helps with future change.
2. Requirements traceability that shows dependencies between requirements and between the requirements and the design/code of the system.
3. Design modeling where the design model documents the structure of the software.
4. Code refactoring that improves code quality and so makes it more amenable to change.

2.9 Suggest two advantages and two disadvantages of the approach to process maturity that is embodied in the SEI's Capability Maturity Framework.

Advantages of process improvement frameworks

1. The approach provides a means of measuring the state of a process and a structured approach to introducing process improvements.
2. It is useful as a way of building on the experience of others in process improvement.

Disadvantages of process improvement frameworks

1. Like any measurement system, there is a tendency to introduce improvements to improve the measured rating rather than concentrate on improvements that meet real business goals.
2. The maturity model approach is expensive and bureaucratic to operate. It is not really suitable for organisations that use agile development.

©Ian Sommerville 2014

Software Engineering 10th edition Sommerville Solutions Manual

3 Agile Software Development

3.2 Explain how the principles underlying agile methods lead to the accelerated development and deployment of software.

The principles underlying agile development are:

1. *Individual and interactions over processes and tools*. By taking advantages of individual skills and ability and by ensuring that the development team know what each other are doing, the overheads of formal communication and process assurance are avoided. This means that the team can focus on the development of working software.
2. *Working software over comprehensive documentation*. This contributes to accelerated development because time is not spent developing, checking and managing documentation. Rather, the programmer's time is focused on the development and testing of code.
3. *Customer collaboration over contract negotiation*. Rather than spending time developing, analyzing and negotiating requirements to be included in a system contract, agile developers argue that it is more effective to get feedback from customer's directly during the development about what is required. This allows useful functionality to be developed and delivered earlier than would be possible if contracts were required.
4. *Responding to change over following a plan*. Agile developers argue (rightly) that being responsive to change is more effective than following a plan-based process because change is inevitable whatever process is used. There is significant overhead in changing plans to accommodate change and the inflexibility of a plan means that work may be done that is later discarded.

©Ian Sommerville 2014

Software Engineering 10th edition Sommerville Solutions Manual

3.3 Extreme programming expresses user requirements as stories, with each story written on a card. Discuss the advantages and disadvantages of this approach to requirements description.

Advantages of stories:

1. They represent real situations that commonly arise so the system will support the most common user operations.
2. It is easy for users to understand and critique the stories.
3. They represent increments of functionality – implementing a story delivers some value to the user.

Disadvantages of stories

1. They are liable to be incomplete and their informal nature makes this incompleteness difficult to detect.
2. They focus on functional requirements rather than non-functional requirements.
3. Representing cross-cutting system requirements such as performance and reliability is impossible when stories are used.
4. The relationship between the system architecture and the user stories is unclear so architectural design is difficult.

3.6 Compare and contrast the Scrum approach to project management with conventional plan-based approaches as discussed in Chapter 23. Your comparison should be based on the effectiveness of each approach for planning the allocation of people to projects, estimating the cost of projects, maintaining team cohesion and managing changes in project team membership.

Planning allocation of people to projects

Scrum

Scrum handles people allocation informally. Team members 'bid' for features from the product backlog to implement if they think that their expertise is appropriate. Alternatively, the tasks can be allocated by the Scrum master.

There is no formal mechanism in Scrum for planning for project members with very specific expertise to be temporarily allocated to a team. This need must be identified by the Scrum master and he or she has to discuss how the expertise can be made available.

Plan-based development

©Ian Sommerville 2014

Software Engineering 10th edition Sommerville Solutions Manual

A project plan is used to identify the parts of the system to be delivered and these are specified in the requirements document. The expertise required for each part can then be identified and the allocation of people to projects planned on that basis.

Estimating project costs

Scrum

Project costs are estimated based on the required delivery date for the software and people working in the Scrum team. The functionality of the system is adjusted so that some working system will always be delivered for the original cost estimation. Of course, this may not be adequate for the customer and they have to become involved in rescheduling the delivery of the system.

Plan-based development

Project costs are based on an analysis of the functionality specified in the requirements document as well as the non-functional requirements of the system. They may be adjusted to reflect team size and delivery schedule. It is normal for costs to be underestimated and the final project to cost much more than originally estimated. An average cost for team members is assumed.

Maintaining team cohesion

Scrum

Team member meet daily either face to face or electronically. Extensive informal discussions and communications are encouraged. Team members negotiate work to be done from the project backlog. This all leads to a shared feeling of product ownership and a very cohesive team.

Plan-based development

Team cohesion is the responsibility of the project manager and he or she has to take explicit actions to encourage this. The general approach relies on formal meetings that are relatively infrequent and this does not lead to the development of a cohesive team.

Managing changes in project team membership

Scrum

This is a topic that is rarely discussed in Scrum but is a fundamental problem because so much information is informal and reliant on people remembering what has been agreed. When someone leaves, it can be very difficult to bring a replacement team member up to speed, especially if very little project documentation is available.

Plan-based development

©Ian Sommerville 2014

Software Engineering 10th edition Sommerville Solutions Manual

The project management plan is based around expertise rather than individuals and project documents should be available. Therefore, if a team member leaves, then a new team member with comparable expertise can read what has been done and, after understanding this, should be able to serve as a replacement.

3.8 Why is it necessary to introduce some methods and documentation from plan-based approaches when scaling agile methods to larger projects that are developed by distributed development teams.

---

1. *Project planning* is often essential when developing software with larger teams to (a) ensure that the right people are available when they are needed to be involved in the development process and (b) ensure that the delivery schedules of different parts of the system developed by different teams are aligned. This means that if Part A depends on Part B, the schedule should ensure that Part B is developed before Part A.
2. *Requirements analysis and documentation* is important to decide how to distribute the work across teams and to ensure that each team has some understanding of what other teams are doing.
3. *Design documentation* especially interface specifications are important so that teams can develop independently without having access to software that is under development.
4. *Risk management* may be required to ensure that all of the teams understand the risks faced and can organize their work to minimize these risks. Risk management may also be useful to cope with different delivery schedules used by different teams.

3.10 It has been suggested that one of the problems of having a user closely involved with a software development team is that they 'go native'. That is, they adopt the outlook of the development team and lose sight of the needs of their user colleagues. Suggest three ways how you might avoid this problem and discuss the advantages and disadvantages of each approach.

---

1. *Involve multiple users in the development team.* Advantages are you get multiple perspectives on the problem, better coverage of user tasks and hence requirements and less likelihood of having an atypical user. Disadvantages are cost, difficulties of getting user engagement and possible user conflicts.

©Ian Sommerville 2014

Software Engineering 10th edition Sommerville Solutions Manual

2. *Change the user who is involved with the team*. Advantages are, again, multiple perspectives. Disadvantages are each user takes time to be productive and possible conflicting requirements from different users.

3. *Validate user suggestions with other user representatives*. Advantages are independent check on suggestions; disadvantage is that this slows down the development process as it takes time to do the checks.

©Ian Sommerville 2014

Software Engineering 10th edition Sommerville Solutions Manual

4 Requirements Engineering

4.2 Discover ambiguities or omissions in the following statement of requirements for part of a ticket-issuing system:

An automated ticket machine sells rail tickets. Users select their destination and input a credit card and a personal identification number. The rail ticket is issued and their credit card account charged. When the user presses the start button, a menu display of potential destinations is activated, along with a message to the user to select a destination and the type of ticket required. Once a destination has been selected, the ticket price is displayed and customers are asked to input their credit card. Its validity is checked and the user is then asked to input their personal identifier (PIN). When the credit transaction has been validated, the ticket is issued.

Ambiguities and omissions include:

1. Can a customer buy several tickets for the same destination together or must they be bought one at a time?
2. Can customers cancel a request if a mistake has been made?
3. How should the system respond if an invalid card is input?
4. What happens if customers try to put their card in before selecting a destination (as they would in ATM machines)?
5. Must the user press the start button again if they wish to buy another ticket to a different destination?
6. Should the system only sell tickets between the station where the machine is situated and direct connections or should it include all possible destinations?

©Ian Sommerville 2014

Software Engineering 10th edition Sommerville Solutions Manual

4.4 Write a set of non-functional requirements for the ticket-issuing system, setting out its expected reliability and response time.

Possible non-functional requirements for the ticket issuing system include:

1. Between 0600 and 2300 in any one day, the total system down time should not exceed 5 minutes.
2. Between 0600 and 2300 in any one day, the recovery time after a system failure should not exceed 2 minutes.
3. Between 2300 and 0600 in any one day, the total system down time should not exceed 20 minutes.

All these are availability requirements – note that these vary according to the time of day. Failures when most people are traveling are less acceptable than failures when there are few customers.

4. After the customer presses a button on the machine, the display should be updated within 0.5 seconds.
5. The ticket issuing time after credit card validation has been received should not exceed 10 seconds.
6. When validating credit cards, the display should provide a status message for customers indicating that activity is taking place.

   This tells the customer that the potentially time consuming activity of validation is still in progress and that the system has not simply failed.
7. The maximum acceptable failure rate for ticket issue requests is 1: 10000.

Note that this is really ROCOF. I have not specified the acceptable number of incorrect tickets as this depends on whether or not the system includes trace facilities that allow customer requests to be logged. If so, a relatively high failure rate is acceptable as customers can complain and get refunds. If not, only a very low failure rate is acceptable.

Obviously, these requirements are arbitrary and there are many other possible answers. You simply have to examine their credibility.

©Ian Sommerville 2014

Software Engineering 10th edition Sommerville Solutions Manual

4.6 Suggest how an engineer responsible for drawing up a system requirements specification might keep track of the relationships between functional and non-functional requirements.

Keeping track of the relationships between functional and non-functional requirements is difficult because non-functional requirements are sometimes system level requirements rather than requirements which are specific to a single function or group of functions.

One approach that can be used is to explicitly identify system-level non-functional requirements that are associated with a functional requirement and list them separately. All system requirements that are relevant for each functional requirement should be listed. They can be related by including them in a table as shown below.

Functional requirement	Related non-functional system requirements	Non-functional requirements
The system shall provide an operation which allows operators to open the release valve to vent steam into the atmosphere.	Safety requirement: No release of steam shall be permitted if maintenance work is being carried out on any steam generation plant.	Timing requirement: The valve must open completely within 2 seconds of the operator initiating the action.

Notice that in this example, the system non-functional requirement would normally take precedence over the timing requirement, which applied to the specific operation.

Obviously, any sensible answer that provides a way of linking functional and non-functional requirements is acceptable here.

4.7 Using your knowledge of how an ATM is used, develop a set of use cases that could serve as a basis for understanding the requirements for an ATM system.

There are a variety of different types of ATM so, obviously, there is not a definitive set of use cases that could be produced. However, I would expect to see use cases covering the principal functions such as withdraw cash, display balance, print statement, change PIN and deposit cash. The use case description should describe the actors involved, the inputs and outputs, normal operation and exceptions.

Withdraw cash:

©Ian Sommerville 2014

Software Engineering 10th edition Sommerville Solutions Manual

Actors: Customer, ATM, Accounting system
Inputs: Customer's card, PIN, Bank Account details
Outputs: Customer's card, Receipt, Bank account details
Normal operation: The customer inputs his/her card into the machine. He/she s promoted for a PIN which is entered on the keypad. If correct, he/she is presented with a menu of options. The Withdraw cash option is selected. The customer is promoted with a request for the amount of cash required and inputs the amount. If there are sufficient funds in his/her account, the cash is dispensed, a receipt if printed and the account balance is updated. Before the cash is dispensed, the card is returned to the customer who is prompted by the machine to take their card.
Exception: Invalid card. Card is retained by machine; Customer advised to seek advice.
Incorrect PIN. Customer is request to rekey PIN. If incorrect after 3 attempts, card is retained by machine and customer advised to seek advice.
Insufficient balance Transaction terminated. Card returned to customer.

Display balance:
Actors: Customer, ATM, Accounting system
Inputs: Customer's card, PIN, Bank Account details
Outputs: Customer's card
Normal operation: The customer authenticates using card and PIN as in Withdraw cash and selects the Display Balance option. The current balance of their account is displayed on the screen. The card is returned to the customer.
Exception: Invalid card. As in Withdraw cash
Incorrect PIN. As in Withdraw cash

Print statement:
Actors: Customer, ATM, Accounting system
Inputs: Customer's card, PIN, Bank Account details
Outputs: Customer's card, Printed statement
Normal operation: The customer authenticates using card and PIN as in Withdraw cash and selects the Print statement option. The last five transactions on their account is printed. The card is returned to the customer.
Exception: Invalid card. As in Withdraw cash
Incorrect PIN. As in Withdraw cash

Change PIN:
Actors: Customer, ATM
Inputs: Customer's card, PIN
Outputs: Customer's card

©Ian Sommerville 2014

Software Engineering 10th edition Sommerville Solutions Manual

Normal operation: The customer authenticates as in Withdraw cash and selects the Change PIN option. He/she is prompted twice to input the new PIN. The PINS input should be the same. The customer's PIN is encrypted and stored on the card. Card returned to customer.
Exception: Invalid card. As in Withdraw cash.
Incorrect PIN. As in Withdraw cash.
PINS do not match. The customer is invited to repeat the process to reset his/her PIN.

Deposit cash:

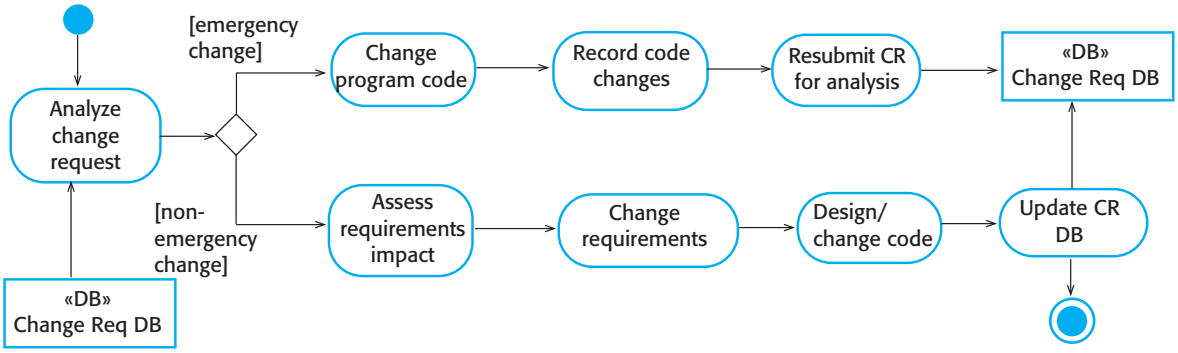
Actors: Customer, ATM, Accounting system
Inputs: Customer's card, PIN, Bank Account details, Cash to be deposited
Outputs: Customer's card, Receipt
Normal operation: The customer authenticates as in Withdraw cash and selects the Deposit option. The customer is promoted with a request for the amount of cash to be deposited and inputs the amount. He or she is then issued with a deposit envelope in which they should put the cash then return it to the machine. The customer's account balance is updated with the amount deposited but this is marked as uncleared funds and is not cleared until checked. A receipt is issued and the customer's card is returned.
Exception: Invalid card. As in Withdraw cash.
Incorrect PIN. As in Withdraw cash.
No cash deposited within 1 minute of envelope being issued. Transaction terminated. Card returned to customer.

4.9 When emergency changes have to be made to systems, the system software may have to be modified before changes to the requirements have been approved. Suggest a model of a process for making these modifications that will ensure that the requirements document and the system implementation do not become inconsistent.

The following diagram shows a change process that may be used to maintain consistency between the requirements document and the system. The process should assign a priority to changes so that emergency changes are made but these changes should then be given priority when it comes to making modifications to the system requirements. The changed code should be an input to the final change process but it may be the case that a better way of making the change can be found when more time is available for analysis.

©Ian Sommerville 2014

Software Engineering 10th edition Sommerville Solutions Manual



Software Engineering 10th edition Sommerville Solutions Manual

5 System Modeling

5.2 How might you use a model of a system that already exists? Explain why it is not always necessary for such a system model to be complete and correct. Would the same be true if you were developing a model of a new system?

You might create and use a model of a system that already exists for the following reasons:

1. To understand and document the architecture and operation of the existing system.
2. To act as the focus of discussion about possible changes to that system.
3. To inform the re-implementation of the system.

You do not need a complete model unless the intention is to completely document the operation of the existing system. The aim of the model in such cases is usually to help you work on parts of the system so only these need to be modelled. Furthermore, if the model is used as a discussion focus, you are unlikely to ne interested in details and so can ignore parts of the system in the model.

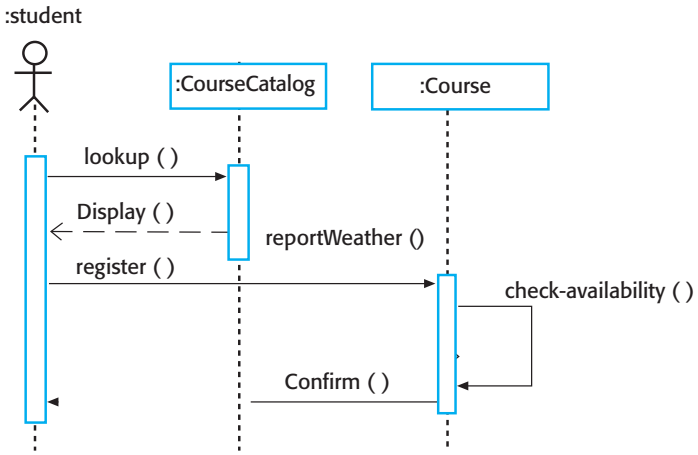
This is true, in general, for models of new systems unless a model-based approach to development is taking place in which case a complete model is required. The other circumstances where you may need a complete model is when there is a contractual requirement for such a model to be produced as part of the system documentation.

5.5 Develop a sequence diagram showing the interactions involved when a student registers for a course in a university. Courses may have limited enrolment, so the registration process must include checks that places are available. Assume that the student accesses an electronic course catalog to find out about available courses.

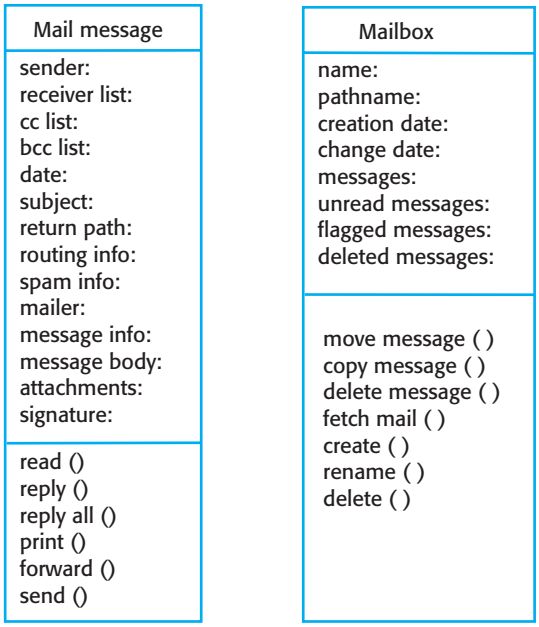
A relatively simple diagram is all that is needed here. It is best not to be too fussy about things like UML arrow styles as hardly anyone can remember the differences between them.

©Ian Sommerville 2014

Software Engineering 10th edition Sommerville Solutions Manual



5.6 Look carefully at how messages and mailboxes are represented in the email system that you use. Model the object classes that might be used in the system implementation to represent a mailbox and an e-mail message.

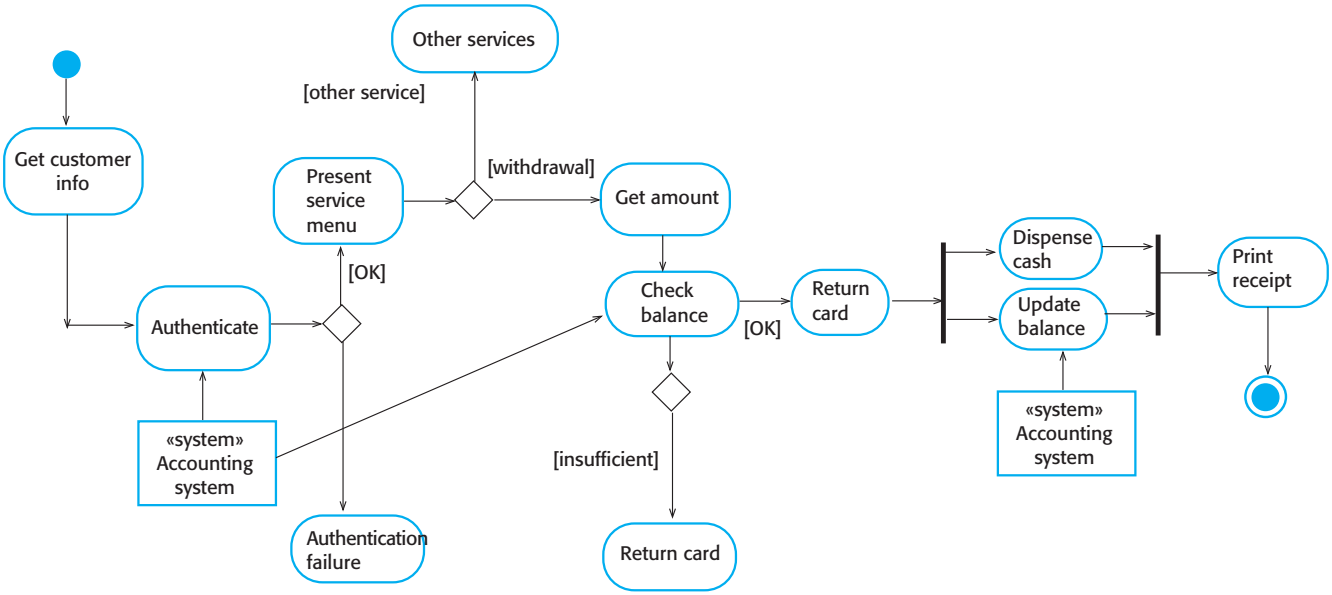


©Ian Sommerville 2014

Software Engineering 10th edition Sommerville Solutions Manual

5.7 Based on your experience with a bank ATM, draw an activity diagram that models the data processing involved when a customer withdraws cash from the machine.

Notice that I have not developed the activities representing other services or failed authentication.



5.10 You are a software engineering manager and your team proposes that model-driven engineering should be used to develop a new system. What factors should you take into account when deciding whether or not to introduce this new approach to software development?

The factors that you have to consider when making this decision include:

1. The expertise of the team in using UML and MDA. (Is expertise already available or will extensive training be required.)
2. The costs and functionality of the tools available to support MDA. (Are tools available in house or will they have to be purchased. Are they good enough for the type of software being developed)

©Ian Sommerville 2014

Software Engineering 10th edition Sommerville Solutions Manual

3. The likely lifetime of the software that you are developing. (MDA is most suitable for long-lifetime systems)
4. Requirements for high performance or throughput (MDA relies on code generation that creates code which may be less efficient than hand written code)
5. The long term benefits of using MDA (are there real cost savings from this approach)
6. The enthusiasm of the software developers. (are all team members committed to this new approach)

©Ian Sommerville 2014

Software Engineering 10th edition Sommerville Solutions Manual