G52SEM
The Concise, Waffle-Less Guide

Version 1.0
Craig Knott
January 22, 2013

N.B This guide is relevant for the January 2013 Examination period, if any information in this guide is misleading or incorrect I hold no responsibility.

1 Foreword

This guide is a concise description and explanations of the topics to be included in the January 2013 G52SEM Examination. All of this information is taken from Lecture slides and external resources linked to by the aforementioned, but attempts to reduce the absurd amount of "waffle" that is generally included along side the important content. It is advised that you first read the lecture slides and external material, and use this guide as a personal testing document, to ensure you have understood and can explain all of the key information.

2 What will be on the exam?

The exam will consist of two parts, the case study and four questions. Each question is worth 33 marks and is heavily based around the case study. Only 3 of the 4 questions require answering, so choose wisely. The case study will be a short scenario that will be the basis of the questions. It is vital to understand the case study, as the questions will often revolve around how you would personal implement and design certain aspects of the system described. The exam itself is 1 (One) Hour and 40 (Forty) Minutes long, that is, 30 (Thirty) Minutes for each question, with 10 (Ten) Minutes at the beginning to read the case study.

Tasks that you may be asked to perform in the exam are as follows (note that this is not a definitive list)

- Writing of a Vision and Scope Document (Or parts of it)
- Use Cases (In the style of Alistair Cockburn)
- Screen mock-ups for specified Use Cases
- Domain Designs
- Principles of Domain Designs
- Use of Design Patterns (MVC for instance)
- Software Project Methodologies (Waterfall, Agile, etc.)
- Definition and description of the *Prince*2 software
- Team Issues
- Agile Practices
- Test Driven Design
- Planning and Estimating
- Development Environments and Software Tools

3 Vision and Scope Documents

A good vision and scope document will help a project avoid some of the costliest problems that a project can face. By writing the document and circulating it among everyone involved in the project, the project manager can ensure that each of the stakeholders and engineers share a common understanding of the needs being addressed and that the software must address themselves.

Project Background
Summary of the problem to be solved

Stakeholders and Users
List of the needs of each stakeholder and user

Risks
List of potential risks (external factors that could impact the project)

Assumptions
List of assumptions made, that are required for the projects success

Vision
A single compelling reason and justification for spending time, money and resources on the project

List of features
A list of the features to be included in the project

Scope of phased release (optional)
The release plan of the project, what will be completed by what deadlines

Features that will not be developed
Features that could, but will not, be developed, and why they will not be.

4 Cockburn Use Cases and Screen Mock ups

Use cases are expansions of user stories, that help to formalise the informal story into a pseudo-specification.

Primary Goal
What is trying to be achieved

Primary Actor(s)
Who is trying to achieve it?

Scope
The system we are discussing

Level
The level of the system this takes place at

Preconditions
Values that are present prior to the beginning of the MSS

Minimal Guarantee
Effects that will be always take place, regardless of success of MSS

Success Guarantee
Effects that will only take place if the MSS is successful

Main Success Scenario
A list of 3 to 9 main activities that take place for the completing of the goal

Extensions and Actions
For each possible failure that could occur in the MSS, an extension should be raise (what could go wrong) and an action decided (how to fix it)

4.1 Screen Mock ups

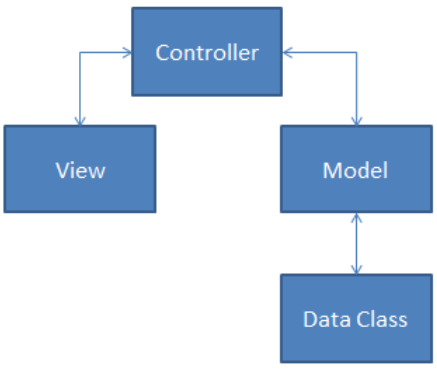
You may be asked to mock up what a specific use case may look like. This is to combat the lack of detail and explanation of the GUI that is present in use cases. Make sure to label all the elements and justify their inclusion.

5 Domain Model Designs

A Domain Model Design shows the set of objects[1] which are instances of classes[2].
Remember the key features of Object Oriented Design

1. Encapsulation
2. Inheritance
3. Polymorphism

Think of Domain Model Designs as Entity Relationship Diagrams. An example from MVC is pictured below



[1] A virtual element that implements behaviour and holds information class
[2] A template for creating objects, behaviour defined in methods, the sets of methods is called an interface, and data is declared in member variables

Listed below are some principles that are generally adhered to when designing software, their initials spell out *SOLID* for easy remembrance.

5.1 S - Single Responsibility Principle (SRP)

Every class should have a single responsibility, and this should be entirely encapsulated by the class.

5.2 O - Open Closed Principle (OCP)

Software entities should be open for extension, but closed for modification

5.3 L - Liskov's Substitution Principle (LSP)

If S is a subtype of T, then objects of type T may be replaced with objects of type S without altering any of the desirable properties of that program

5.4 I - Dependency *I*nversion Principle (DIP)

High level modules should not depend on low level modules, they should both depend on abstractions
Abstractions should not depend on details, details should depend on abstractions

5.5 D - Demeter's Law

"You may only talk to your friends. You cannot talk to the friends of your friends".

6 Design Patterns

6.1 Layering

Layering is the practice of applying additional communications with each layer, and that each layer may only speak to the layer directly above or below it

6.2 Model View Controller (MVC)

A Graphical User Interface design pattern that specifies that the visual elements and data storage may never take to one another, and may only interact through the controller

6.3 Object Relational Mapping (ORM)

A technique for converting data between incompatible type systems.

7 Prince2

Prince2 stands for PRojects in Controlled Environments. It was a project management software that began development in the 90's. The advantages of Prince2 are that

- It is widely recognised and understood
- Provides common vocabulary for all project participants, promoting communication
- Explicitly recognises project responsibilities allowing the understanding of other team roles
- It teaches useful skills, like planning, risk assessment, monitoring and change management

7.1 Is Prince2 agile?

No, it is a process-driven management method, which contrasts with the reactive/adaptive methods like Scrum and extreme programming. Theoretically it could work with Scrum, as tasks are divided into sections, but the iterations are generally far too long.

8 Team Working

The five main dysfunctions of team working are

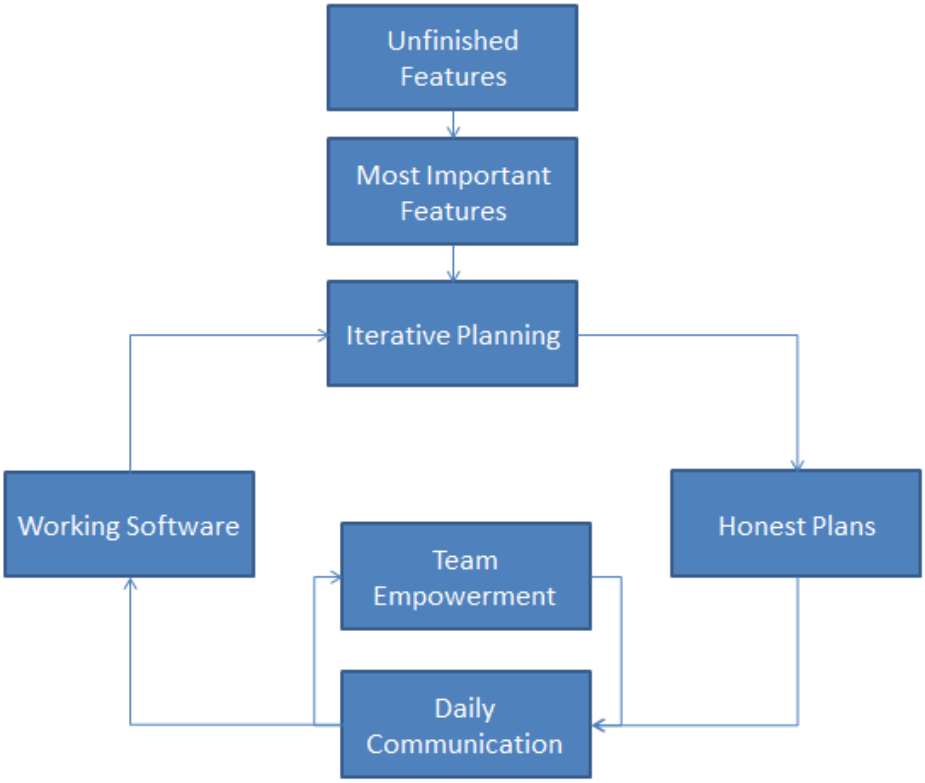
1. Absence of Trust
   Fear of being vulnerable prevents building trust
2. Fear of Conflict
   Desire to preserve artificial harmony stifles productivity
3. Lack of Commitment
   Lack of clarity prevents team members from sticking to decisions
4. Avoidance of Accountability
   The need to avoid interpersonal discomfort prevents team members holding each other accountable
5. Inattention to Global Success
   Pursuit of individual goals erodes collective success

There are a few key points to remember when constructing a good team

- Remember the process is difficult and time consuming
- Deal with conflict, do not simple avoid it
- Assign tasks to member's strengths
- Focus on team goals
- Keep communication up

9 Extreme Programming

Extreme programming is a lightweight, people friendly approach to software engineering. It focuses on very short development cycles, with an emphasis on testing, honest trade-offs and team work. It stresses customer satisfaction by delivering software you need, when you need it.



9.1 Values of extreme programming

- Simplicity
  Keep designs simple and clean. Build the simplest thing that will be useful, then expand it by adding the simplest feature possible

- Communication
  Constantly converse as a team, and with customers

- Feedback
  Provide early, quick releases for customer feedback

- Respect
  All contributions come from success and respect of each other

- Courage
  Any changing requirements can be confidently dealt with

9.2 Negatives of extreme programming

- Informal change management
- Incremental requirements may lead to many revisions
- No big designs up front
- Hard to find the simplest thing
- Customers can become too attached to the project
- Breaking down the walls of ownership

10 Scrum and Sprint

A Scrum is a new way for teams to work together, where they focus on creating a collection of small developed pieces, that are later merged together.
The key members of the Scrum are

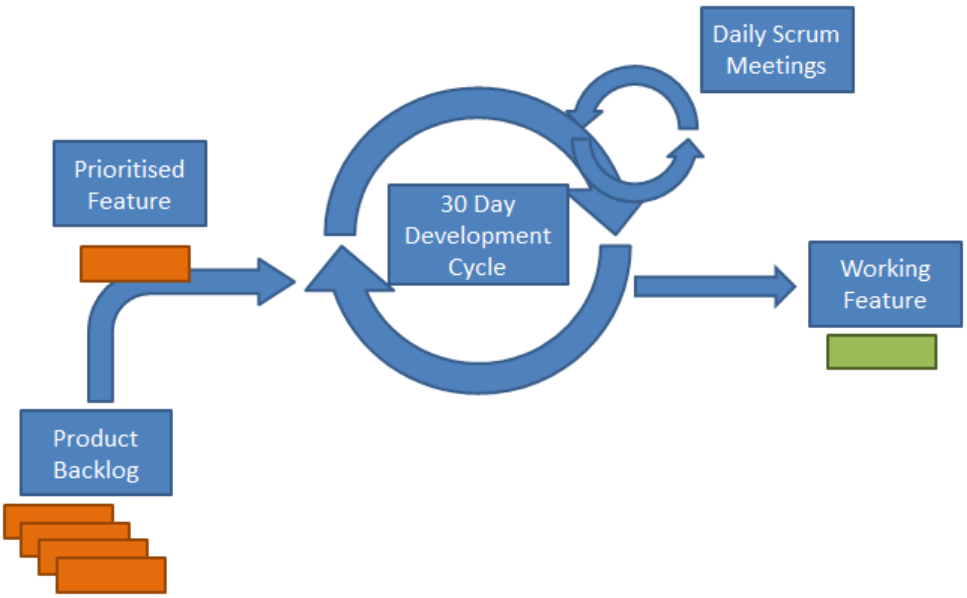
- The Product Owner Decide what needs to be built in each Sprint, dependant on what the development team produced in the last one.
- The Development Teams Build what is needed and display what they have achieved.
- The Scrum Masters Ensure that the Scrum process happens as smoothly as possible

The general objective of the Scrum is to divide the project into a long list of features, called the Product Backlog. These are then prioritised, and each one developed in a Sprint. The object of the Sprint is to complete one of these features in each of the iterations. Daily meetings, called Scrum Review Meetings are held to discuss the progress of the project, what obstacles are present and what should be achieved by the next meetings.

10.1 Scrum Values

- Transparency
  All aspects of an element should be visible for those responsible for building it
- Inspection
  A frequent task of evaluating progress
- Adaptation
  If the inspection reveals an element that deviates outside of acceptable limits, it must be modified

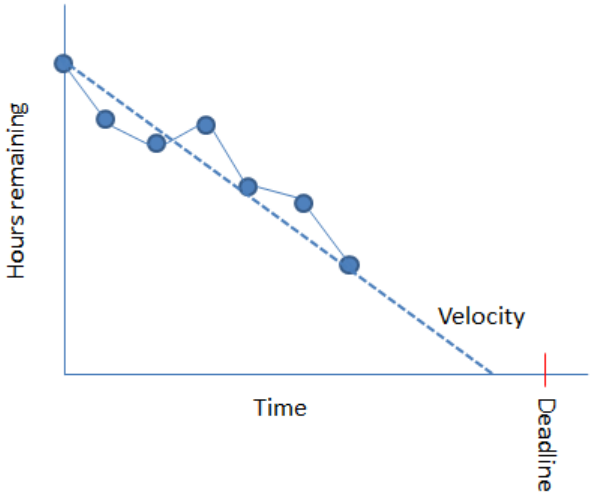
10.2 Scrum Diagram



11 Burn down Charts and Kanban Boards

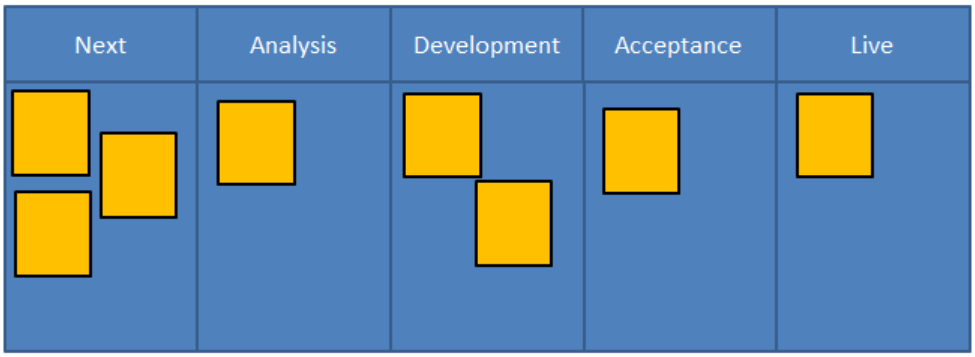
11.1 Burn down Chart

A burn down chart is a measure of how much needs to be completed, against how much time remains. The Y axis displays the number of man hours the total time of the project was estimated to take, and the X axis is time, up to the deadline. Each day, a point is plotted to display how much progress has been made. After a few days, a trend line can be plotted, this is called the project velocity. This can be used to show whether the project is on track or not.



11.2 Kanban Boards

A Kanban board a board with a list of activities that move along the board as progress is made on them. They are good for visualization the work flow of a project, and allow for quick assessment of the progress of a project. The categories for most software engineering projects are, "Next", "Analysis", "Development", "Acceptance" and "Live", showing how the features go through their individual life cycles. Tasks can also be marked in order of importance, so that team members prioritise which tasks they focus on.



12 Planning and Estimating

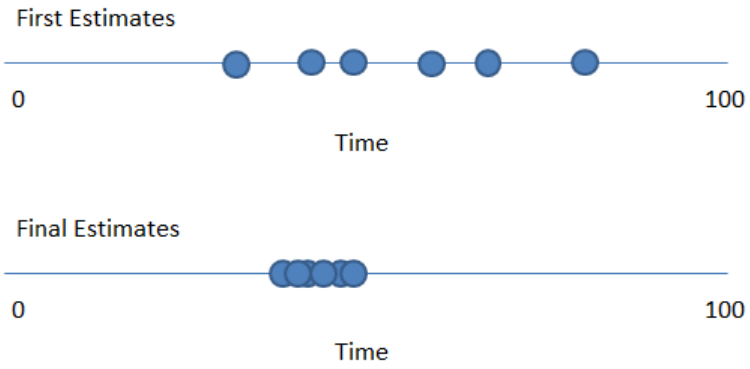
12.1 Time Estimates

12.1.1 Parkinson's Estimate

"Work expands to fill the time available for it's completion"

12.1.2 Wideband Delphi

- Define clearly possible tasks
- Estimate time for each task
- Estimators reveal their reasoning without revealing their estimates
- Repeat 2-3 until estimates converge

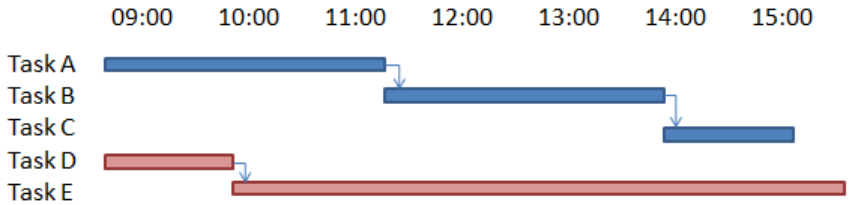


12.2 Planning

Plans are generally a directed tree of tasks, linked to their prerequisites. Any loops means you have made an error.

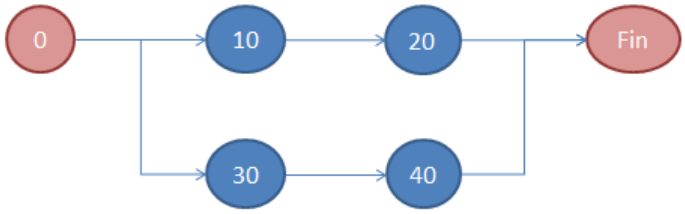
12.2.1 Gantt Charts

Named after Henry Gantt. A bar chart that illustrates the amount of time that each of the tasks of the project will take. It displays how they are linked together through prerequisites.



12.2.2 PERT Charts

Projection Evaluation Review Technique charts, are a collection of numbered nodes, where two consecutive events are linked by an arrow that is labelled with the task required to traverse these nodes, and the time weight of it.

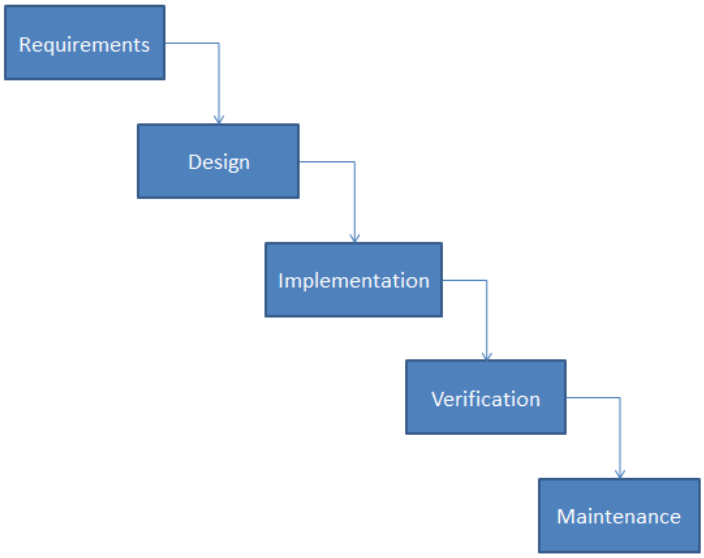


12.3 Critical Path Analysis

This represents the minimum time required to complete the project. Any extension on the critical path will increase the overall life time of the project.

13 Waterfall Method

The waterfall model is the old method of doing things in software engineering. It was the de facto method, but new agile approaches are becoming more and more popular. The idea of the waterfall model is perfect each stage before moving onto the next, and hope that you will not need to back track. The problem with this is that it is too inflexible and the cost of change increases as the project progresses. It does have some strengths, such as good planning, solid designs, and good documentation so that others can take over from you. However these do not overcome the rigidity and error resonation that can occur.



The stages present in the waterfall method

14 Agile Methods

Agile methods are modern, iterative approaches to software development, that are replacing the older models due to their reactive and adaptive methodologies.

14.1 Agile Manifesto

The agile manifesto states

Value	Over
Individuals and Interactions	Process and tools
Working Software	Comprehensive documentation
Customer Collaboration	Contract negotiation
Responding to change	Following a plan

14.2 When should you use Agile approaches?

Generally you should use agile in all new projects. But it is best suited for groups of 25 members or less, who generally have a high level of experience. It is also good for projects where the requirements are set to change a lot, as agile methods are fit to deal with change a lot better. "Adaptation, instead of prediction"

Some different agile methodologies are discussed below.

14.3 Spiral Model

The spiral model is an iterative approach to software development. It divides the project into smaller agile segments, and each segment involves repeating the software engineering process over it many times. It was one of the first of its kind. It is a good method as it combines an

iterative development with a systematic approach, which allows for incremental releases of work and has built in risk assessment.

14.4 Agile Unified Process (AUP)

The Agile Unified Process encompasses seven main principles, and these define a simple, easy to understand approach to developing business application software using agile techniques and concepts.

These principles are

Model	Understand the organisation and problem domain to identify the solution domain to identify the solution
Implementation	Transform the model into code, with unit tests
Test	Test for bugs and imperfections
Deployment	Plan for system delivery, and make sure the system is usable by end users
Configuration Management	Manage access to past releases
Project Management	Direct activities that are to take place, including risks, people, and coordination with external parties
Environment	Supporting the rest of the effort by ensuring that the correct tools and guidance are available

14.5 Pair Programming

Pair programming, or *broprogramming*, is the discipline of having two people write the code together with one mouse, keyboard and computer. The purpose is that two people will be able to better spot typographical errors, question whether code will work or if it could be improved, question what may break, and spot may could be simplified. The benefits of pair programming are generally that the two parties will share knowledge, the code will contain less bugs, and that two heads are generally better than one.

15 Bad Software Engineering Methodologies

In this day an age, some software engineering methodologies have become outdated. One of these is the water fall method, as discussed above, another two are discussed below, these are "Big Specification Up Front" and "Coding without designs".

15.1 Big Specification Up Front

The idea of this approach is to create the most detailed specification at the beginning of the project. The problem with this approach is that at the beginning, you only have a limited knowledge, we don't know how the project will develop, and the specification will become so large that starting the task will seem too monumental.

15.2 Coding without Designs

"Monkey at a type writer". If you don't have a clear model of how the system will work - it won't.

16 Source Control

There are three main source control system that may be brought up, *Git*, *Mercurial*, and *Subversion* (*SVN*). Source control is used to manage past releases, keep a back up of work, and keep a record of changes.

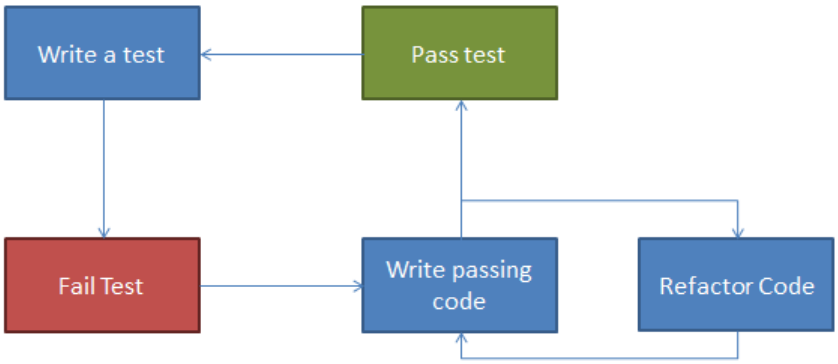
GIT	Fast, generally easier to branch and merge, good interface, and you can download local code base
Mercurial	Tidier than GIT, good choice for open source projects, used like *SVN*
SVN	Easy to display changes, simple and predictable, supports locking of files, expects request small commits

As we used SVN in our group projects, some useful commands to remember are

svn update	Refresh your working copy of the code base
svn ci -m	Check-in changes, with a comment
svn co	Check-out a working copy of the code base
svn add	Add a file to the code base
svn delete	Delete a file from the code base

17 Test Driven Design and Unit Testing

Test Driven Development (coupled with unit testing) is a new way to program that focuses on, instead of writing the code and then writing tests, the tests are written first, and the code is then written to pass these tests. The advantages of test driven design are that it provides fearless refactoring, emergent architecture, less time spent debugging, lower defect rates and confidences in the operation of the program. It generally follows the following model.



18 Principles of software engineering

- Make it as simple as possible
- Built it simply, then improve upon it
- Decide what it should do before how it should do so
- Make it work before making it pretty or efficient