Questions Updated, correct to the best of my knowledge! Cheers Marina for pointing out the answer to b pt. 2.

a) Discuss, giving your own opinion, the ideas contained in Edsger Dijkstra's "On the Cruelty of really teaching Computer Science". [Marks 15]

http://en.wikipedia.org/wiki/On_the_Cruelty_of_Really_Teaching_Computer_Science

- The main idea argues that computer programming should be understood as a branch of mathematics, and that the formal provability of a program is a major criterion for correctness.
- However, Since the term "software engineering" was coined, formal verification has almost always been considered too resource-intensive to be feasible.
- In complex applications, the difficulty of correctly specifying what the program should do in the first place is a common source of error.
- The notion that the cost of production of hardware should be a constraint in programming was foreign to Dijkstra and until the end of his life, Dijkstra maintained that the central challenges of computing hadn't been met to his satisfaction, due to an insufficient emphasis on program correctness.

- I think that Computer Science as taught today does not follow all of Dijkstra's advice. Today's teachings generally emphasize techniques for managing complexity and preparing for future changes eg. abstraction, programming by contract, and design patterns. Programming techniques to avoid bugs and conventional software testing methods are taught as basic requirements, and students are exposed to certain mathematical tools, but formal verification methods are not included in the curriculum except perhaps as advanced topics. So in some ways, Dijkstra's ideas have been adhered to; however, the ideas he felt most strongly about have not been.

b) Give a summary of the main ideas in the paper "SLAM and Static Driver Verifier: Technology Transfer of Formal Methods inside Microsoft" by Thomas Ball, Byron Cook, Vladimir Levin, and Sriram K. Rajamani, and, based on the paper, discuss what are, in your opinion, the three most important software engineering issues associated with technology transfer of formal methods? [Marks 15]

Acronym - FERPMC (Cheers Will)

Main Ideas

- Focus on Problems not Technology
    - It is easier to convince a product group to adopt a new solution to a pressing problem that they already have in comparison to convincing them to adopt new technology if the link to the problem that it solves is unclear.
- Exploit Synergies
    - It's a great idea for people to cross the boundaries of their traditional research communities to collaborate with people from other similar communities when trying to solve a problem. Progress in research can be accelerated by following this recipe.
- Reflect and Assess

- In a research project that spans several years, it is important to regularly reassess the progress you are making towards your main goal. However, this is not an easy task, as the difficulty of correctly specifying what the project should do in the first place is a common source of error.

Most important software engineering issues associated with technology transfer of formal methods

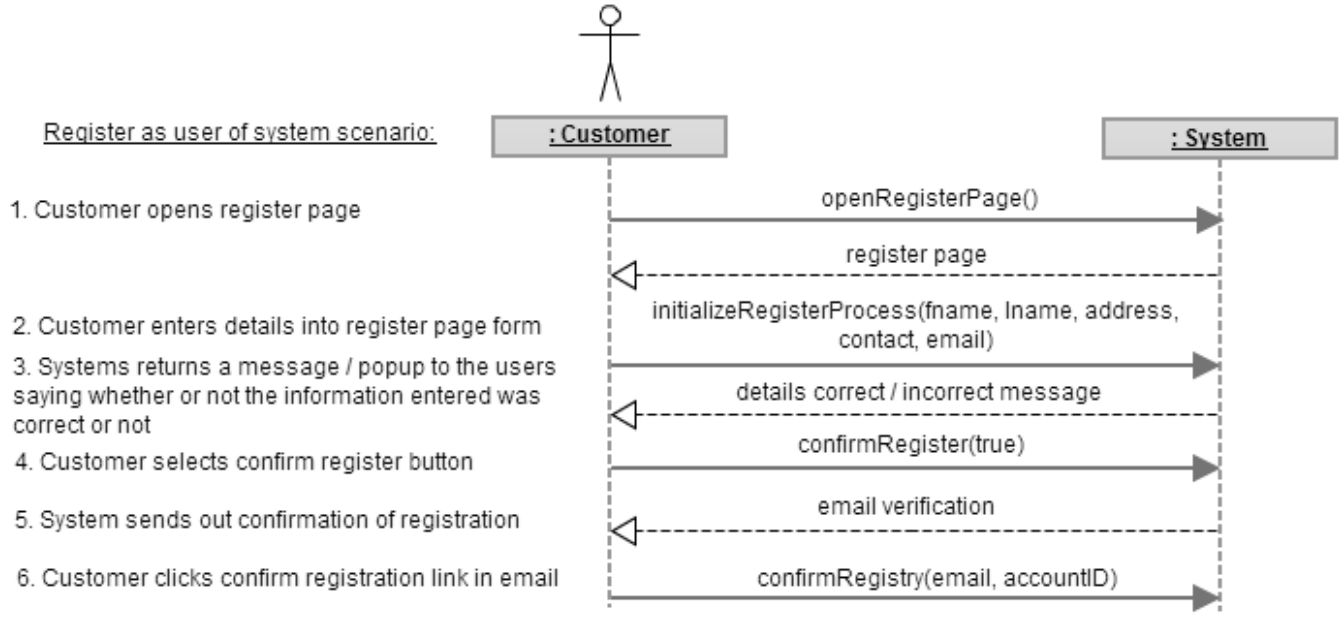
- Plan Carefully
  - To get maximum leverage in any research project, one has to plan in order to be successful. It is crucial not to underestimate the value of such ground work. However this is not always easy, as some aspects of the project cannot always be accurately forecasted.
- Maintain Continuity and Ownership
  - Interns and visitors can write code but then they leave, someone has to maintain continuity of the project. This is quite a difficult task, as once these people leave, someone else, who may have had no prior involvement in the project, may have to pick up where they left off.
- Communication
  - It is important that all members working on the project communicate effectively, regarding all aspects of the project they are undertaking. However this is not always the case, for example, the new project manager of SDV 1.1 thought the project was ready for development much in advance than the actual development date, and this was due to a lack of or poor communication.

c) A process, such as the Rational Unified Process or Larman's Agile UP can be used to assist in achieving a software implementation that satisfies the requirements of the system. A formal method, such as program verification based on Floyd-Hoare Logic, can also be used to assist in ensuring that a software implementation satisfies the requirements of the system. Discuss the use of these two techniques, including in the discussion any differences and similarities, and illustrating the discussion with particular application examples. [Marks 20]

- Both the Rational Unified Process and Larman's Agile UP are features of the Unified Modelling Language. For more than a decade now, the two communities of UML and formal methods have been working together to produce a simultaneously practical (via UML) and rigorous (via formal methods) approach to software engineering.
- UML is the de facto standard for modelling various aspects of software systems in both industry and academia, despite the inconvenience that its current specification is complex and its syntax imprecise.
- The fact that the UML semantics is too informal have led many researchers to formalize it with all kinds of existing formal languages in order to measure the correctness of computer programs.

//Rational Unified Process (Uses component-based architectures)
UML

- The Unified Modeling Language includes a set of graphic notation techniques to create visual models of object-oriented software-intensive systems. The Unified Modeling Language is used to specify, visualize, modify, construct and document the artifacts of an object-oriented software-intensive system under development.
- For example, System Sequence Diagrams are regularly used within UML. Here the system sequence diagram is used primarily to show the interactions between objects in the sequential order that those interactions occur, and also to show possible inter-system events.



Floyd-Hoare Logic

- Floyd–Hoare logic is a formal system with a set of logical rules for reasoning rigorously about the correctness of computer programs.
- The central feature of Hoare logic is the Hoare triple. A triple describes how the execution of a piece of code changes the state of the computation. A Hoare triple is of the form {P} C {Q} where P and Q are assertions and C is a command. P is named the precondition and Q the postcondition: when the precondition is met, the command establishes the postcondition.
- For example;

$$\{x+1=43\}\, y := x+1 \,\{y=43\} \qquad \text{(Assignment Axiom)}$$

$$(x+1=43 \Leftrightarrow x=42)$$

$$\{x=42\} \qquad y := x+1 \,\{y=43 \wedge x=42\} \qquad \text{(Consequence Rule)}$$