Back to Software Engineering

- Large, complex, multi-version software needs a proven development process in order to be successful
- It is not sufficient – but is necessary

114

114

Successful Software Applications

- What represents success? Well engineered software is:
  - *Correct* in that it both provides the behaviour the client requires, and also implements that behaviour according to an agreed on specification
  - *Reliable* in that the software performs its required functions under specified conditions for a specified period of time
  - *Efficient* in that it performs its functions within a specified time and within specified resources
  - *Maintainable* over its lifetime in that it is robust with respect to changes that have to be made – the level of confidence that it meets all its quality attributes is maintained after changes are made to the system
  - Intuitively *usable* with an appropriate user interface

115

115

3 Vital Components in SE (The 3 Ps)

- People
  - Must be "qualified" i.e. competent to do their work
  - Each person has a specified role (perhaps more than one)
- Process
  - There are various development processes with different pros and cons
    - Waterfall, spiral, agile
- Product
  - The tangible implementation (code, firmware, etc)
  - Support documents (requirements, design, analysis, reviews, testing)

116

116

People

- Team members have specific tasks and must have the training to perform those tasks
- Social skills often are as important as technical skills – easy to understand when we know that software engineering is performed by groups
- Cohesive groups are a pre-requisite for success – and good communications within and between groups is essential
  - We have excellent tools to support some of this

117

117

Process

- An effective process must have well defined steps with explicit milestones and clear success criteria for each task
- The process provides guidance on what to do next
- It helps prevent essential steps from being skipped
- It facilitates planning and scheduling
- A well-planned "rational" process helps us avoid
  - duplication of effort
  - work that is not directly useful for the task at hand

118

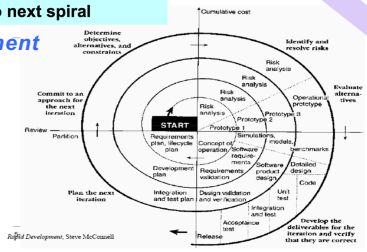
118

Example Processes

Software development life-cycle is often referred to as *SDLC*

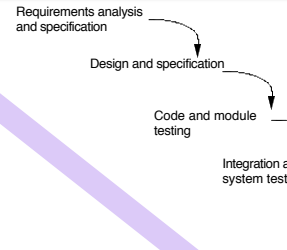
- **Determine objectives and constraints**
- **Identify and resolve risks**
- **Evaluate options to resolve risks**
- **Develop and verify deliverables**
- **Plan next spiral**
- **Commit (or not) to next spiral**

Spiral Development
from
Washington University
Computer Science

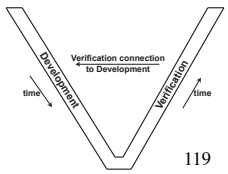


Waterfall Model

From "Fundamentals of Software Engineering", Ghezzi, Jazayeri & Mandrioli



V-Model



119

119

Product

- Documentation
  - Planning documents (including standards), especially configuration management and revision control
  - Development documents such as hazard analyses, requirements, design, code listings, test reports
  - Installation documents
  - User guides and operations documents
  - Maintenance documents
- Executable code – or alternatives such as circuits for *Field Programmable Gate Arrays (FPGAs)*

120

120

Standards

- There are many international standards that govern the development of software-dependent systems
  - The three main international standards bodies in this regard are: *IEEE*, *ISO* and *IEC*
- Current standards have a number of positive attributes
  - They provide guidance for those who are new to the process
    - And may be effective teaching tools for all users
  - They aid forward progress by suggesting "next steps"
  - They were designed to serve as a foundation for achieving and maintaining quality products (unfortunately most standards are process-focused, and following a good process is not guaranteed to produce a high quality product)

121

121

Standards (continued)

- Standards provide a means to capture international consensus on what constitutes professional practice in their respective areas
- Many process oriented standards recommend ways to avoid systematic errors
- Some years ago a number of "functional safety standards" were introduced
  - IEC 61508
    - ISO 26262 - automotive
    - IEC 62304 – medical
  - DO-178C – civil aviation

122

122

Summary: Recap of Software Engineering

- Engineering approach to software development
- Rigorous rather than "hacking"
- It is much more than good ways to code
  - The product includes code, but also all supporting documentation
- Requires well-prepared people and processes to produce an adequate product
  - Qualified people
  - Applying a planned and systematic process
  - Resulting in a well-engineered system
  - With adequate objective evidence that all quality attributes have been achieved

123

123

Engineering

- The profession of Engineering was started in order to further technological innovation and deployment – while *protecting the public from harm*
- It is natural, therefore, to consider our responsibility when developing software-intensive systems that have the capability of causing harm

124

124

Safety

- Safety is basically freedom from harm
- It is a global property of the system – you can take 2 components that are provably safe on their own, but when you put them together the resulting system is not safe
- Developers of safety-critical software must ensure that
  - system safety requirements are complied with
  - the software does not introduce behaviours that could contribute to the system causing harm
  - the software will maintain any equipment and processes the system controls in a “safe state”, even when hardware fails
  - must not allow inappropriate system inputs to result in the system causing harm

125

125

Safety-Critical Software

- *Safety-critical systems* are those systems in which a failure may lead to a catastrophic result – loss of life, serious injury, etc.
- *Safety-critical software* is software used in a safety-critical system (definitions coming a little later)
- What is different about safety-critical software compared with general software?
  - Safety (and security) as a primary goal
  - *ALARA* – Risk is *As Low As Reasonably Achievable*
  - Development with licensing in mind
  - Hazard analysis
  - Hard real-time behaviour
  - Enhanced rigour
  - Thorough documentation
  - Stakeholders

126

126

Safety-Critical Software: Safety and Security

- *Safety**: "the expectation that a system does not, under defined conditions, lead to a state in which human life, health, property, or the environment is endangered"
- *Security**: "the protection of system items from accidental or malicious access, use, modification, destruction, or disclosure"

* ISO/IEC/IEEE 24765, Systems and software engineering – Vocabulary, 2010-12-15

127

127

Examples of Safety-Critical Systems

Nuclear Power Station

Pacemaker

Plane

Modern Car

Train

Insulin Pump

128

128

Regulation

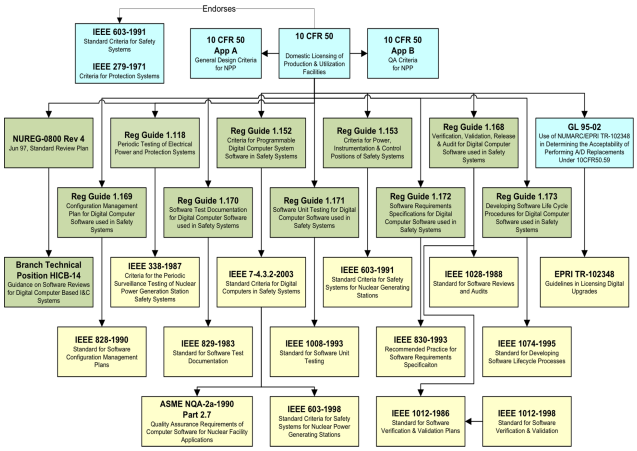
- With regard to software, some application domains are regulated
  - Nuclear power
  - Civil aviation
  - Medical devices
  - Trains
- Some are not
  - Automobiles!

129

129

Example – US Nuclear Regulation

Regulations and Guidance for DI&C



130

130