MIEIC – Software Engineering - 2014/15
Bibliographic Research Work

Goals and rules

The goal of this work, to be performed by groups of 4 to 6 students, is to conduct a bibliographic research (with practical experimentation whenever possible) on one of the software engineering topics listed below, and to prepare a 10 minutes presentation (annotated slides) on the subject.

The oral presentations will take place in the week of 15 to 19 December 2014, according to a schedule to be defined. The group member that will present the work will be selected **randomly** at the begin of the oral presentation.

The annotated slides (written in Portuguese or English) should be submitted by email to the corresponding TP-classes teacher (either nuno.flores@fe.up.pt or hugo.sereno@fe.up.pt) until the end of the 14th of December.

The group composition and theme assignment will be decided in the practical classes of the week of 1 to 5 December 2014. It is recommended that in the classes of the week of 1 to 5 December, each work group performs an initial search and elicitation of relevant Web resources related to the theme, refines goals and scope, and distributes work to be performed until the next class.

This work has a weight of 2 points (out of 20) in the **final mark**.

The final exam will include a group of questions related with the themes of the research works assigned. Each student will have to answer to two questions (related with two themes).
In the following, it is presented a list of themes proposed, grouped by categories. Each category is pre-assigned to a turn as indicated in the table below.

3MIEIC01	3MIEIC02	3MIEIC03	3MIEIC04	3MIEIC05	3MIEIC06	Total
3ª, HSF	6ª, NHF	5ª, HSF	4ª, HSF	5ª, HSF	3ª, NHF	
19	18	23	20	22	20	122
Topic 4	Topic 1	Topic 6	Topic 5	Topic 2	Topic 3	

Contents

1 Software requirements and architecture

1.1 Crowd sourcing and collaborative approaches for requirements elicitation, validation and management

The term crowdsourcing was coined by journalist Jeff Howe, who defines it as “outsourcing a task to a large group of people in the form of an open call”. Crowdsourcing has been used increasingly by the software industry to both lower opportunity costs and increase quality of output by utilizing capital from outside the company, in the form of the experience, labor, or creativity of outside programmers worldwide.
The goal of the research work is to analyse the present and future of crowd sourcing in software engineering, particularly(but not limited) for requirements elicitation.
References:

- http://en.wikipedia.org/wiki/Crowdsourcing
- *StakeCloud - Stakeholder Requirements Communication and Resource Identification in the Cloud*, Irina Todoran (RE’2012) (http://crisys.cs.umn.edu/re2012/program.shtml)

1.2 Software security requirements engineering

Security requirements are a prominent kind of non-functional requirements.
Security is defined in the ISO/IEC 25010 standard as “the degree to which a product or system protects information and data so that persons or other products or systems have the degree of data access appropriate to their types and levels of authorization”. The standard also decomposes security into 5 sub-characteristics: confidentiality, integrity, non-repudiation, accountability and authenticity.
An example of a security requirement is: “The system shall give access to the electronic health record of a patient only to health personnel in charge of his/her treatment”.
The goal of this work is to clarify the vocabulary related to security requirements, and to study approach(es) for identifying, analysing, modeling (e.g., misuse case modeling), specifying and validating security requirements for software based systems. At least an example should be given.
References:

- http://www.cert.org/sse/square/
- http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=58744
- http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=04359475
- http://msdn.microsoft.com/en-us/library/ff648644.aspx
- http://en.wikipedia.org/wiki/Misuse_case
- http://en.wikipedia.org/wiki/Threat_model

1.3 Software architectures and platforms for cloud computing

Cloud computing is one of the most disruptive technologies of recent years. From small businesses to large enterprises, cloud technologies are increasingly becoming the platform of choice for developing and deploying applications.
In addition to lowering the total cost of ownership of applications and systems, cloud platforms provide several characteristics which allow software developers to architect applications which have high performance, massive scalability, high availability and reliability.
In order to address the challenges posed by contemporary application scenarios, a software engineer today should know how to build applications using cloud platforms.
The goal of this work is to study platforms for developing and deploying software applications in the cloud, as well as architectural aspects to consider in the design of such applications. At least one practical example should be given.

Referências:

- https://www.google.com/about/jobs/search/#!t=jo&jid=1036002&
- http://www.oracle.com/technetwork/articles/cloudcomp/jimerson-ha-arch-cloud-1669855.html

1.4 Softwares architecture for big data processing

In recent years, “Big Data” has become a new ubiquitous term. Big Data is transforming science, engineering, medicine, healthcare, finance, business, and ultimately society itself.

References:

- http://blog.sei.cmu.edu/post.cfm/challenges-big-data-294

2 Software design

2.1 Design patterns for multi-core architectures

What are the challenges that arise in developing software for multi-core architectures? What design patterns, pattern languages, programming paradigms and tool support can help in that task?

References:

- http://parallelpatterns.codeplex.com/
- http://www.multicore-systems.org/iwmse2011 (4th International Workshop on Multicore Software Engineering)

2.2 Design patterns for security engineering

Design patterns can be applied to achieve goals in the area of security. All of the classical design patterns have different instantiations to fulfil some information security goal: such as confidentiality, integrity, and availability. Additionally, one can create a new design pattern to specifically achieve some security goal.
References:

- http://www.cert.org/secure-coding/
- http://www.cs.st-andrews.ac.uk/~ifs/Books/SE9/ (namely section 14.1 Design for security)
- http://homes.dico.unimi.it/~monga/sess11.html (The 7th International Workshop on Software Engineering for Secure Systems)
- http://en.wikipedia.org/wiki/Security_Patterns

2.3 Design patterns for cloud computing

Cloud computing comes in several flavours: Infrastructure-as-a-Service (IaaS) such as Amazon Web Services, or AWS; platform-as-a-service (PaaS) including Google App Engine or Azure; software-as-a-service (SaaS) e.g., salesforce.com; frameworks that enable the above like Apache Hadoop or Microsoft's Dryad. In any of these flavours, cloud computing has engendered a disruptive change in the requirements, architecture, implementation and evolution methodologies for software. Investigate what design patterns for cloud computing are of higher relevance for software development.

References:

- http://www.cloudpatterns.org/
- https://sites.google.com/site/icsecloud2011/ (Software Engineering for Cloud Computing Workshop)
- http://blogs.msdn.com/b/billzack/archive/2010/03/29/windows-azure-design-patterns.asp

x (Windows Azure Design Patterns)
- https://www.docker.com/
- https://coreos.com/

2.4 Map-reduce: Design patterns for BigData Computing

MapReduce is a programming model for processing large data sets, and the name of an implementation of the model by Google. MapReduce is typically used to do distributed computing on clusters of computers. The model is inspired by the map and reduce functions commonly used in functional programming, although their purpose in the MapReduce framework is not the same as their original forms. MapReduce libraries have been written in many programming languages. A popular free implementation is Apache Hadoop.
References:

- ttp://en.wikipedia.org/wiki/MapReduce
- ttp://www.infoq.com/news/2012/02/MapReducePatterns
- https://spark.apache.org/

2.5 Reactive Software. Design patterns for large scale software systems.

Redundancy frequently exists in all stages of software development. This duplication of information can increase the size and cost of software artifacts and cause maintenance headaches to keep changes in sync. There are various ways to deal with redundancy, namely by reuse, refactoring, modularization and abstraction. However, there scenarios where redundancy is actually beneficial, as it can improve the quality and/or resiliency of a system. Can you describe some of these scenarios?

What are the challenges of dealing with redundancy? When should it be eliminated from software artifacts and processes?

Some examples/references:

- Reactive Manifesto. http://www.reactivemanifesto.org/
- Reactive Design Patterns. http://www.manning.com/kuhn/
- Principles of Reactive Programming. https://www.coursera.org/course/reactive

3 Software engineering processes and tools

3.1 Web based IDEs - Widely adoption possible?

Analyse existing web base IDEs.

References:

- http://techgearz.com/2011/06/eclipse-orion-project-a-web-based-ide/
- http://shiftedit.net/
- https://www.fpcomplete.com/blog/2012/11/designing-the-haskell-ide
- http://ace.c9.io/

3.2 PEX4fun - Serious gaming environment for teaching?

Pex4Fun (http://www.pexforfun.com/) is a web-based serious gaming environment for teaching computer science. Pex4Fun can be used to teach and learn computer programming at many levels, from high school all the way through graduate courses. With Pex4Fun, a student edits code in any browser -- with Intellisense -- and Pex4Fun executes it and analyzes it in the cloud. Pex4Fun connects teachers, curriculum authors, and students in a unique social experience, tracking and streaming progress updates in real time. In particular, Pex4Fun finds interesting and unexpected input values that help students understand what their code is actually doing. The real fun starts with Coding Duels where students write code to implement a teacher's specification. Pex4Fun finds any discrepancies in behavior between the student's code and the specification.

3.3 Jenkins - Continuous Integration Software

Continuous integration, one of the foundational aspects of Agile software development methodologies, is defined by Martin Fowler to be "a fully automated and reproducible build, including testing, that runs many times a day. This allows each developer to integrate daily, thus reducing integration problems."
Jenkins is an open source continuous integration tool written in Java. The project was forked from Hudson after a dispute with Oracle. Jenkins provides continuous integration services for software development. It is a server-based system running in a servlet container such as Apache Tomcat. It supports SCM tools including CVS, Subversion, Git, Mercurial, Perforce, Clearcase and RTC, and can execute Apache Ant and Apache Maven based projects as well as arbitrary shell scripts and Windows batch commands. The primary developer of Jenkins is Kohsuke Kawaguchi. Released under the MIT License, Jenkins is free software. Builds can be started by various means, including being triggered by commit in a version control system,

scheduling via a cron-like mechanism, building when other builds have completed, and by requesting a specific build URL. Investigate features related to Jenkins, discussing advantages and limitations.

References:

- http://jenkins-ci.org/
- http://martinfowler.com/articles/continuousIntegration.html

3.4 Application lifecycle management with IBM Rational Team Concert

Application Lifecycle Management (ALM) is a continuous process of managing the life of an application through governance, development and maintenance. ALM is the marriage of business management to software engineering made possible by tools that facilitate and integrate requirements management, architecture, coding, testing, tracking, and release management. IBM Rational Team Concert is a software development team collaboration tool developed by the Rational brand of IBM and was first released in 2008. The software is available in both client versions and a Web version. It provides software development teams with a collaborative environment in which they can manage all aspects of their work, such as plans, tasks, revision control, build management, and reports. IBM Rational Team Concert is built on the Jazz Platform, an extensible technology platform that helps teams integrate tasks across the software life cycle.

References:

- http://en.wikipedia.org/wiki/Application_lifecycle_management
- http://www-01.ibm.com/software/rational/products/rtc/

3.5 Application lifecycle management with Microsoft TFS

Application Lifecycle Management (ALM) is a continuous process of managing the life of an application through governance, development and maintenance. ALM is the marriage of business management to software engineering made possible by tools that facilitate and integrate requirements management, architecture, coding, testing, tracking, and release management. “Visual Studio Team Foundation Server 2012 (TFS) is the collaboration platform at the core of Microsoft's application lifecycle management (ALM) solution. TFS supports agile development practices, multiple IDEs and platforms locally or in the cloud and gives you the tools you need to effectively manage software development projects throughout the IT lifecycle.”

References:

- http://en.wikipedia.org/wiki/Application_lifecycle_management
- http://msdn.microsoft.com/en-us/vstudio/ff637362.aspx

4 Software construction and programming paradigms

4.1 Post-Functional Programming: The best of both worlds?

Scala is a general purpose programming language designed to express common programming patterns in a concise, elegant, and type-safe way. It smoothly integrates features of both object-oriented and functional languages, enabling programmers to ready their programs for the new multi-core, multi-machine, cloud-based era of computing. Its powerful, higher-kinded, partial inferred type-system not only reduces code size (usually by a factors of 2 - 3 when compared to an equivalent Java application), but it also ensures higher degrees of correctness and code re-use. On the other hand, the mathematical treatment typical of functional principles plagues the newcomer with arcane terms such as monads, monoids, and functors. Will the powerfulness of Scala be both its strength as well as its doom? How will it stand against the ivory tower of Haskell?

References:

- http://www.scala-lang.org/
- http://typeclassopedia.bitbucket.org/

4.2 Java 8

A major upgrade to Java was released on 18 March 2014 and included some features that were planned for Java 7 but later deferred. This include language-level support for lambda expressions (officially, lambda expressions; unofficially, closures) under Project Lambda, and default methods (virtual extension methods) which make multiple inheritance possible in Java; Project Nashorn, a JavaScript runtime which allows developers to embed JavaScript code within applications; Annotation on Java Types; Unsigned Integer Arithmetic; among others. Discuss why these functionalities were highly-requested from the community, their advantages to the construction of software, and if (and why) they are on par with other advanced high-level languages such as Swift, Rust and Scala.

References:

- http://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html
- https://blogs.oracle.com/thejavatutorials/entry/jdk_8_documentation_developer_preview

4.3 Polytipic and Origami Programming

Shapeless is a type class and dependent type based generic programming library for Scala. It had its origins in several talks by Miles Sabin (@milessabin), given over the course of 2011, on implementing scrap your boilerplate and higher rank polymorphism in Scala. Since then it has evolved from being a resolutely experimental project into library which, while still testing the limits of what's possible in Scala, is being used widely in production systems wherever there are arities to be abstracted over and boilerplate to be scrapped.

There are advantages in expressing programs as instances of common patterns, rather than from first principles — the same advantages as for any kind of abstraction. Essentially, one can discover general properties of the abstraction once and for all, and infer those properties of the specific instances for free. These properties may be theorems, design idioms, implementations, optimisations, and so on. Some of such patterns/abstractions are known as folds and unfolds. In a precise technical sense, folds and unfolds are the natural patterns of computation over recursive datatypes; unfolds generate data structures and folds consume them. Functional programmers are very familiar with the foldr function on lists, and its directional dual foldl; they are gradually coming to terms with the generalisation to folds on other datatypes.

Discuss what kind of abstraction does Polytipic and Origami programming stands on, and why would such abstractions help on the construction of correct software. You can choose to explore this topic with either Scala or Haskell.

References:

- https://www.cs.ox.ac.uk/jeremy.gibbons/publications/origami.pdf
- https://github.com/milessabin/shapeless
- http://typeclassopedia.bitbucket.org/
- https://www.haskell.org/haskellwiki/Typeclassopedia

4.4 Go

Go, also commonly referred to as **golang**, is a programming language initially developed at Google in 2007. It is a statically-typed language with syntax loosely derived from that of C, adding garbage collection, type safety, some dynamic-typing capabilities, additional built-in types such as variable-length arrays and key-value maps, and a large standard library.

References:

- http://en.wikipedia.org/wiki/Go_%28programming_language%29
- https://golang.org/

4.5 Swift

Swift is an innovative new programming language for Cocoa and Cocoa Touch. Writing code is interactive and fun, the syntax is concise yet expressive, and apps run lightning-fast. Swift is ready for your next iOS and OS X project — or for addition into your current app — because Swift code works side-by-side with Objective-C.

Compare the language with other well-known high-level languages (like Java, Scala and even Objective-C). Also point out its applicability and development tools.

References:

- http://en.wikipedia.org/wiki/Swift
- https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/

4.6 Clojure

Clojure is a dialect of the Lisp programming language created by Rich Hickey. Clojure is a general-purpose programming language with an emphasis on functional programming. It runs on the Java Virtual Machine, Common Language Runtime, and JavaScript engines. Like other Lisps, Clojure treats code as data and has a macro system.

Compare the language with other well-known functional languages (like Erlang, Haskell and Scala) and more common Object-Oriented Languages (Java, C#). Also point out its applicability and development tools.

References:

- http://en.wikipedia.org/wiki/Clojure

4.7 Rust

Rust is a general purpose, multi-paradigm, compiled programming language developed by Mozilla Research. It is designed to be a "safe, concurrent, practical language", supporting pure-functional, concurrent-actor, imperative-procedural, and object-oriented styles.

References:

- http://en.wikipedia.org/wiki/Rust_%28programming_language%29

5 Software verification and validation

5.1 Mobile Apps Testing - Testing with Frank

Frank allows you to write structured text test/acceptance tests/requirements (using Cucumber) and have them execute against your iOS application. What are the specific challenges of testing mobile applications with Frank?

References:

- http://www.testingwithfrank.com/
- http://cukes.info/
- http://en.wikipedia.org/wiki/Mobile_application_testing
- http://googletesting.blogspot.pt/2013/08/how-google-team-tests-mobile-apps.html

5.2 Web application test automation with Selenium

Selenium is a portable software testing framework for web applications. Selenium provides a record/playback tool for authoring tests without learning a test scripting language (Selenium IDE). It also provides a test domain-specific language (Selenese) to write tests in a number of popular programming languages, including C#, Java, Groovy, Perl, PHP, Python and Ruby. The tests can then be run against most modern web browsers. Selenium deploys on Windows, Linux, and Macintosh platforms.
What are the challenges of web application testing and test automation? How does Selenium addresses those challenges? What are its main features? Can you provide a simple example? How does it compare with other tools?

References:

- http://seleniumhq.org/
- http://en.wikipedia.org/wiki/Selenium_(software)

5.3 Property-based testing

ScalaCheck is a library written in Scala and used for automated property-based testing of Scala or Java programs. ScalaCheck was originally inspired by the Haskell library QuickCheck, but has also ventured into its own.

The goal of this work is to experiment with QuickCheck/ScalaCheck and get acquainted with its potentialities. Discuss how is this philosophy different from classic unit-testing.

References

- http://www.scalacheck.org/

5.4 Behaviour-driven development with Cucumber

“Behaviour-Driven Development (BDD) is an evolution in the thinking behind Test-Driven Development and Acceptance Test Driven Planning.” What is BDD? How does it compare/relate with TDD? The goal of this work is to experiment with Cucumber and get acquainted with its potentialities.

References:

- http://en.wikipedia.org/wiki/Behavior_Driven_Development
- http://cukes.info/
- http://jbehave.org/
- http://behaviour-driven.org/

6 Software evolution

6.1 Program understanding: tools and techniques

Program comprehension ("program understanding", "source code comprehension") is a domain of computer science concerned with the ways software engineers maintain existing source code. The cognitive and other processes involved are identified and studied. The results are used to develop tools and training. Software maintenance tasks have five categories: adaptive maintenance, corrective maintenance, perfective maintenance, code reuse, and code leverage. This work should discuss current state-of-the-art tools and techniques for program comprehension of software systems (apart from those used in class - e.g., http://cs.brown.edu/~spr/codebubbles/index.html).

6.2 Program slicing

What is program slicing? How can it be used to help program understanding and maintenance?

References:

- http://en.wikipedia.org/wiki/Program_slicing

6.3 Self-* systems: how far are we from the fact of having self-aware applications?

Autonomic Computing refers to the self-managing characteristics of distributed computing resources, adapting to unpredictable changes while hiding intrinsic complexity to operators and users. . Started by IBM in 2001, this initiative ultimately aims to develop computer systems capable of self-management, to overcome the rapidly growing complexity of computing systems management, and to reduce the barrier that complexity poses to further growth.
In a self-managing autonomic system, the human operator takes on a new role: instead of controlling the system directly, he/she defines general policies and rules that guide the self-management process. For this process, IBM defined the following four functional areas:

- Self-configuration: Automatic configuration of components;
- Self-healing: Automatic discovery, and correction of faults;
- Self-optimization: Automatic monitoring and control of resources to ensure the optimal functioning with respect to the defined requirements;
- Self-protection: Proactive identification and protection from arbitrary attacks.

References:

- http://en.wikipedia.org/wiki/Autonomic_computing
- http://www.cs.helsinki.fi/u/niklande/opetus/SemK07/Self-healing-intro.pdf

6.4 Software development and maintainability metrics

A software metric is a measure of some ***property*** of a piece of software or its specifications. Since quantitative measurements are essential in all sciences, there is a continuous effort by computer science practitioners and theoreticians to bring similar approaches to software development. The goal is obtaining objective, reproducible and quantifiable measurements, which may have numerous valuable applications in schedule and budget planning, cost estimation, quality assurance testing, software debugging, software performance optimization, and optimal personnel task assignments.

In this work you should discuss common software metrics (size, cost, quality, productivity, etc.) and how can they be used to improve (e.g., in terms of automatically measure code quality and maintainability) software development.

Interesting recent news include:

- http://www.sig.eu/en/News_and_publications/Publications/1366/__US-Health-Care-Exchanges-most-efficient-software-development-project-on-the-planet__.html
- http://dailyinfographic.com/wp-content/uploads/2013/10/1276_lines_of_code2.png
- http://www.sig.eu/en/Research/690/__Maintainability_Model__.html
- http://xradar.sourceforge.net/
- http://agile.diee.unica.it/wetsom2011/index.html

6.5 Automatic Programming

Automatic Programming identifies a type of computer programming in which some mechanism generates a computer program to allow human programmers to write the code at a higher abstraction level.

References:

- http://en.wikipedia.org/wiki/Automatic_programming