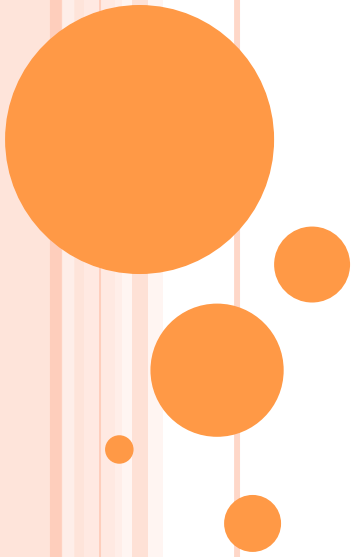
SOFTWARE SECURITY



Put together by –

Meenakshi Mani

Tanvi Shah

What is Software Security

- Its all about building secure software !
- The process of designing, building, and testing software for security
- Taking the pro-active approach : building security INTO the software as opposed to securing it after building it.

Just to clarify ..

Software Security ≠ Security Software !!

WHY GO IN FOR SOFTWARE SECURITY(1)

- Good, secure software is the need of the day.
- Reduction in the expenses incurred in “Fixing bugs” in a software.
- Market value – if your software isn’t secure, it is not going to stay in the market

WHY GO IN FOR SOFTWARE SECURITY(2)

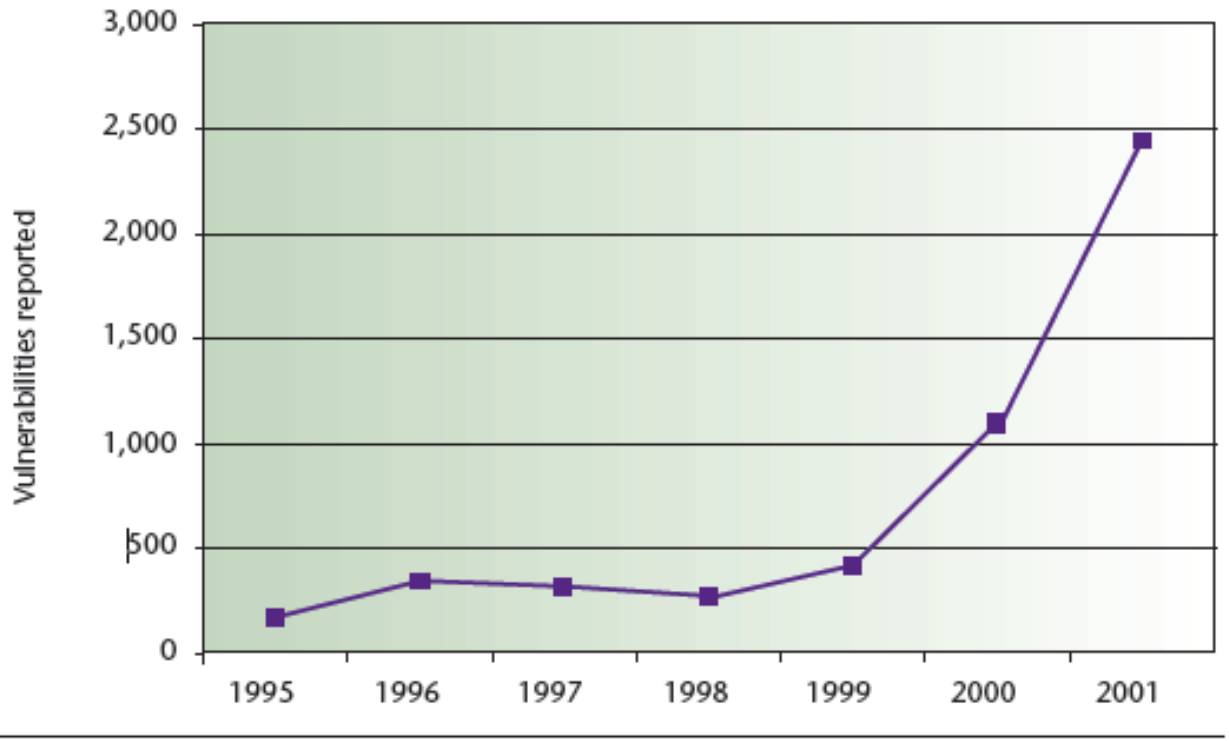


Figure 1. Security-related software vulnerabilities reported to CERT/CC over several years. This figure shows that the problem is growing, but, we don't know the exact reasons for this growth (for example, more defective code, better hacking, more code, and so on).

Reasons for insecure Software

- **Reliance on networked devices**
  - *Growing internet connectivity makes it easier for hackers*
- **Easily extensible systems**
  - *Extensions increase scope for software vulnerabilities*
- **Increasing complexity of the software needed to be built**
  - *Windows XP had 40 million lines of code!!*

WHAT CAN WE DO ?

- Be pro-active in building security into the software from ground-up
- Important to include security into every phase of the Software Development life cycle.

WHY ?

- Software security is a system-wide issue that involves both building in security mechanisms and designing the system to be robust.
- You can't spray paint security features onto a design and expect it to become secure.
- Most approaches in practice today involve securing the software AFTER its been built.
- Not the best approach , and certainly not effective enough as has been proved (we still have issues with our software being meddled with by hackers don't we!)

BUILDING SECURITY IN(1)

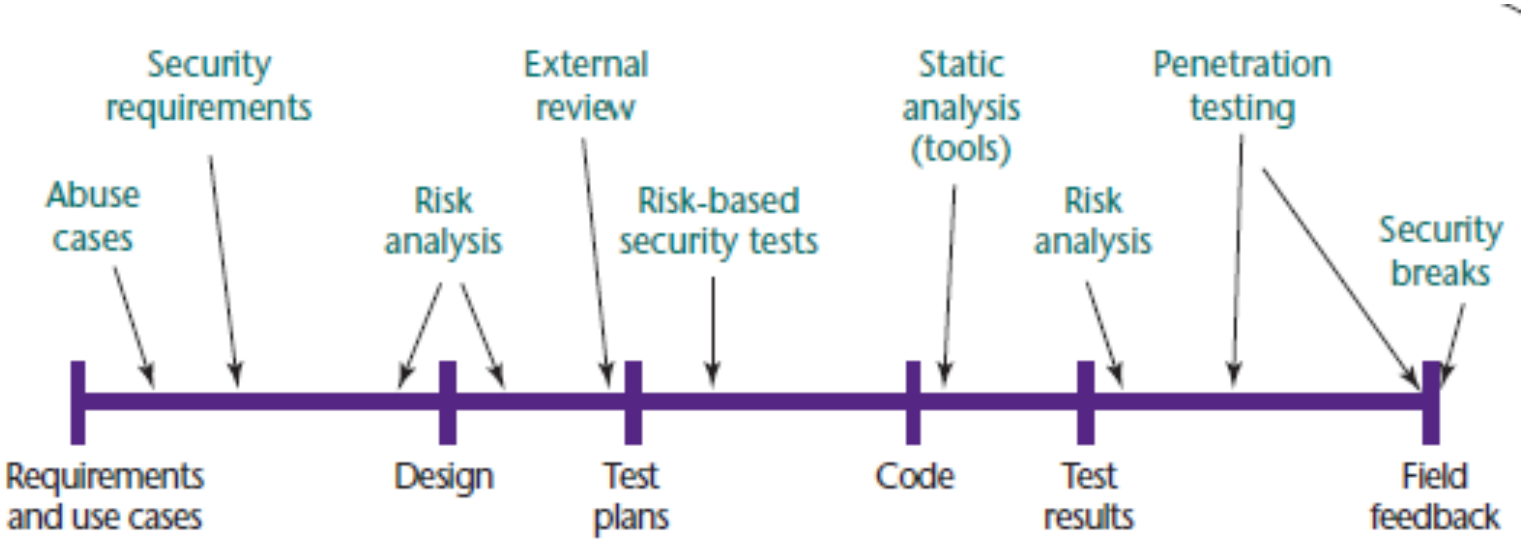


Figure 1. Software security best practices applied to various software artifacts. Although the artifacts are laid out according to a traditional waterfall model in this illustration, most organizations follow an iterative approach today, which means that best practices will be cycled through more than once as the software evolves.

BUILDING SECURITY IN(2)

- Security requirements ; deal with “abuse cases”- how the system will react under attack and inject requirements which will address that
- Take into account the various security principles while charting out the basic design or architecture
- Risk analysis to identify the risks. Will include external review

BUILDING SECURITY IN(3)

- At the code level , make use of static analysis tools – tools that can search code for common vulnerabilities (like buffer overflow)

- Security testing must involve two strategies :
  - *Testing security functionality with standard test techniques*
  - *Risk-based security testing based on attack patterns and models*

BUILDING SECURITY IN(4)

- Penetration testing

  Good to understand the behavior of your software in a real-time environment

- Security breaks

  Observe and study them . Recycling knowledge from attacks and exploits back into the organization.

Security Design Principles -yes !! We have principles to go by ! =)

- 10 principles which go by the “90/10” rule
- Avoid 90% of the potential problems by following 10 simple rules !

PRINCIPLE#1:SECURE THE WEAKEST LINK

- Your system is as secure as its weakest link
- Secure all parts of your system from bottom-up
- All the cryptography in the world will not help if there is an exploitable buffer overflow in the software

PRINCIPLE#1(CONT.)

- Perform Risk analysis iteratively and keep identifying potential risk areas and fixing them

- Stop iterating at some point when it feels that all components are within an acceptable risk threshold.

PRINCIPLE#2:DEFENSE IN DEPTH

- Manage risk with multiple layers of defensive strategies.
- If the error is not caught my one layer of defense, it will be caught by the next layer.
- A bank is typically more secure than a convenience store . (security cameras, guards, a vault – that's a 3-tier defense already !)

PRINCIPLE#3:SECURE FAILURE

- Any system having complex functionality will have failure modes
- Problem – failures can cause insecure behavior in the system
- All someone needs to do is force failure to take advantage of the system.
- Thus make sure that if and when your system fails, it fails in a secure manner.

PRINCIPLE#4:LEAST PRIVILEGE

- Accesses to perform operations within the system should be given out very specifically and for specific amount of time

- Coding with the attitude “ someday I may want to perform these operations so lets set up these privileges for convenience” is a very bad idea

PRINCIPLE#5:COMPARTMENTALIZE

- The idea behind this to essentially minimize the damage that can be done if a part of your system is compromised
- This will make implementation of the previous principle easier!
- However, use compartmentalization in moderation or you will end up with lots of little fragments of software that you cannot manage!

PRINCIPLE#6:SIMPLICITY

- The KISS principle (Keep it simple stupid!)
- But don't oversimplify things!
- Designing a website online involving data transactions without using encryption – not a wise idea
- As simple as it can get while still meeting your security requirements
- Reuse of components

A LITTLE DIVERSION..

- **Choke Points**

  Design all the security-critical operations to be funneled through specific number of choke points. Monitoring anomalies becomes easier

- **Usability**

  The hardest and most important task – finding a balance between security and usability

PRINCIPLE#7:PROMOTE PRIVACY

- Privacy is of primary importance – especially for online software applications

- Trade-off :Privacy against usability

  Eg: storing credit card numbers in the website database and auto-prompting a returning user (high usability , low privacy/low security)

PRINCIPLE#8:IT'S HARD TO HIDE SECRETS

- No matter how hard you try to hide your data, there is always going to be someone who will find a way to break through to it
- Reverse engineering is not that difficult!
- So keep that in mind while chalking out your design – will my sensitive data still be safe if people know how my system works?

PRINCIPLE#9:DO NOT EXTEND TRUST EASILY

- Don't blindly assume that all end-users of your software are going to be trustworthy

  Social engineering attacks (customer support)

- Any entity that you are trusting with sensitive data , you are implicitly trusting anyone that the entity will in turn trust.

PRINCIPLE#10:TRUST THE COMMUNITY

- It looks contradictory to the previous principle (sometimes you need to use your own discretion)

- This principle applies as long as you are sure that the community is working on the security aspects of the components that you want to use and is dedicated to it.

  Eg: in cryptography it is considered bad to use an algorithm that is not publicly known and has been scrutinized.

SOME OO PRINCIPLES YOU CAN USE TO IMPLEMENT THESE PRINCIPLES !

- Encapsulation
- Abstraction
- Single Responsibility Principle

GOOD PRACTICES AND APPROACHES(1)

- Training and education

  -The most important people involved are the design architects , developers and testers

  -Essential to train them to think not just "functionality" but also "security"

  -Make security an integral part of learning how to program

GOOD PRACTICES AND APPROACHES(2)

- Microsoft's Trustworthy Computing Initiative

-The process encompasses the addition of a series of security-focused activities and deliverables to each of the phases of Microsoft's software development process

-Microsoft is focusing on people, process and technology to deal with the software security problem.

Microsoft's trustworthy computing initiative

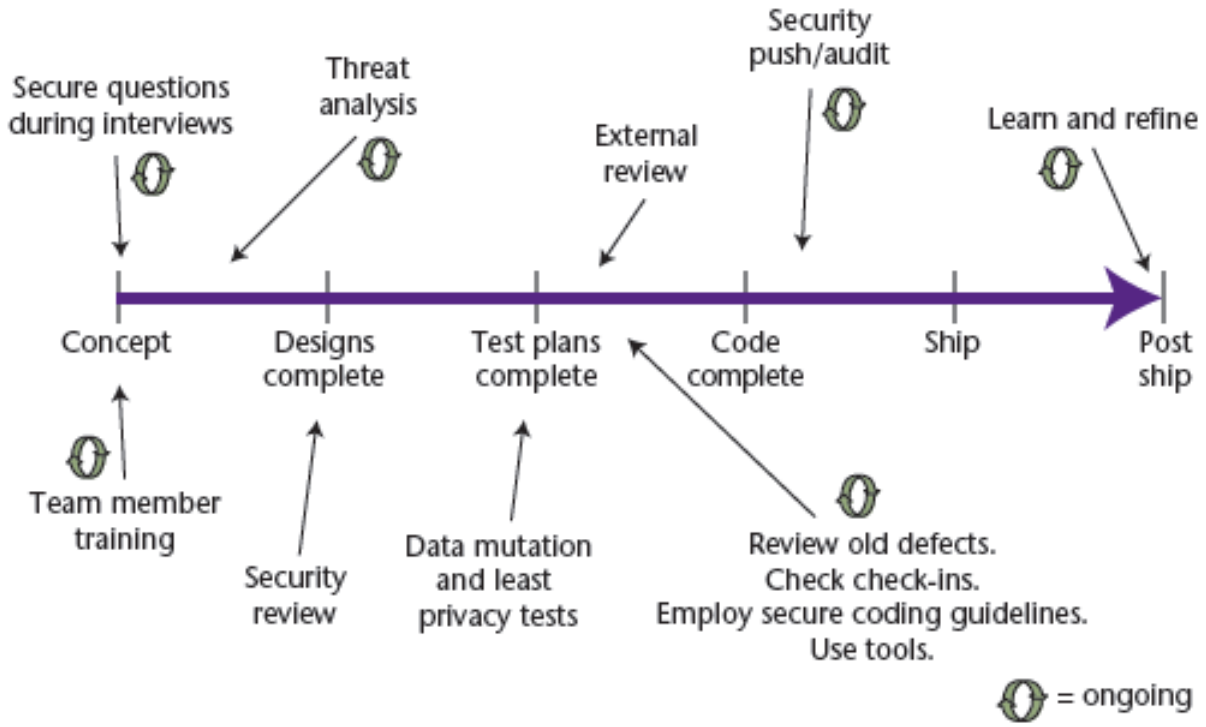


Figure 3. Microsoft has put the following software security process into place. Notice that security does not happen at one lifecycle stage, nor are constituent activities "fire and forget." The idea of constantly revisiting analyses at all levels is critical to software security.

GOOD PRACTICES AND APPROACHES(3)

- Measurement is an important part of effective software security implementation.

-At every step quantify the impact that your design decision is having upon the security of the software you are trying to build

IF WE HAVE NOT CONVEYED THE POINT ALREADY ..

- The importance of looking into security right at the beginning of development

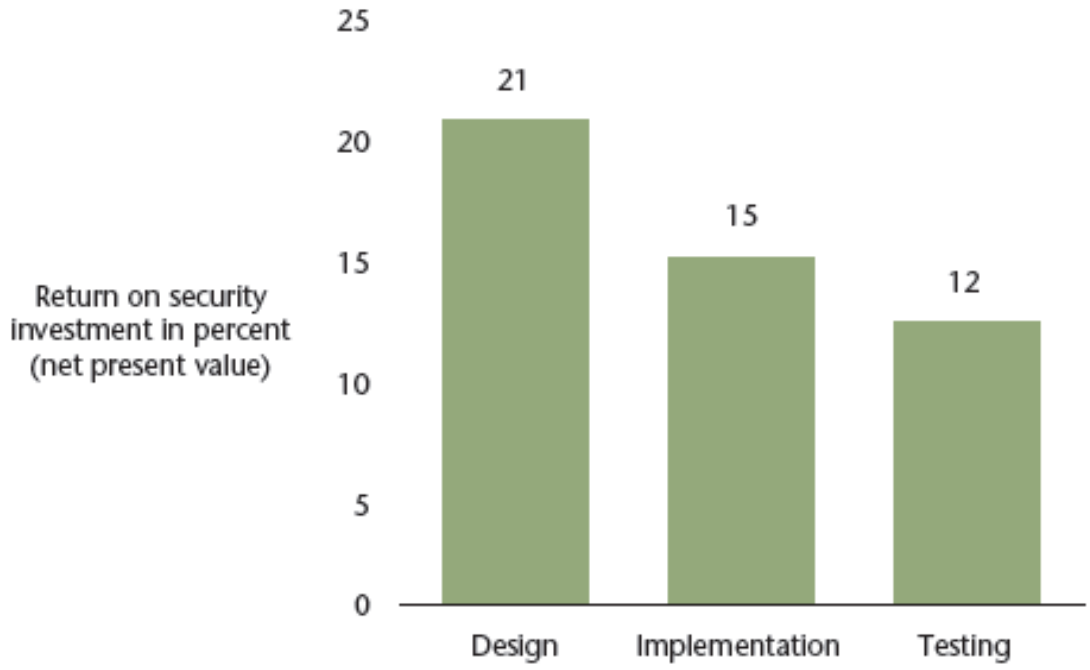


Figure 4. Phase return on investment (ROI) in percent as measured by @stake over 23 security engagements. This study demonstrates the importance of design-level risk analysis as software security best practice.

NOW MOVING ONTO A DIFFERENT ASPECT..

Integrating security with agile methodologies

Agile **– high development speed with more customer satisfaction.**

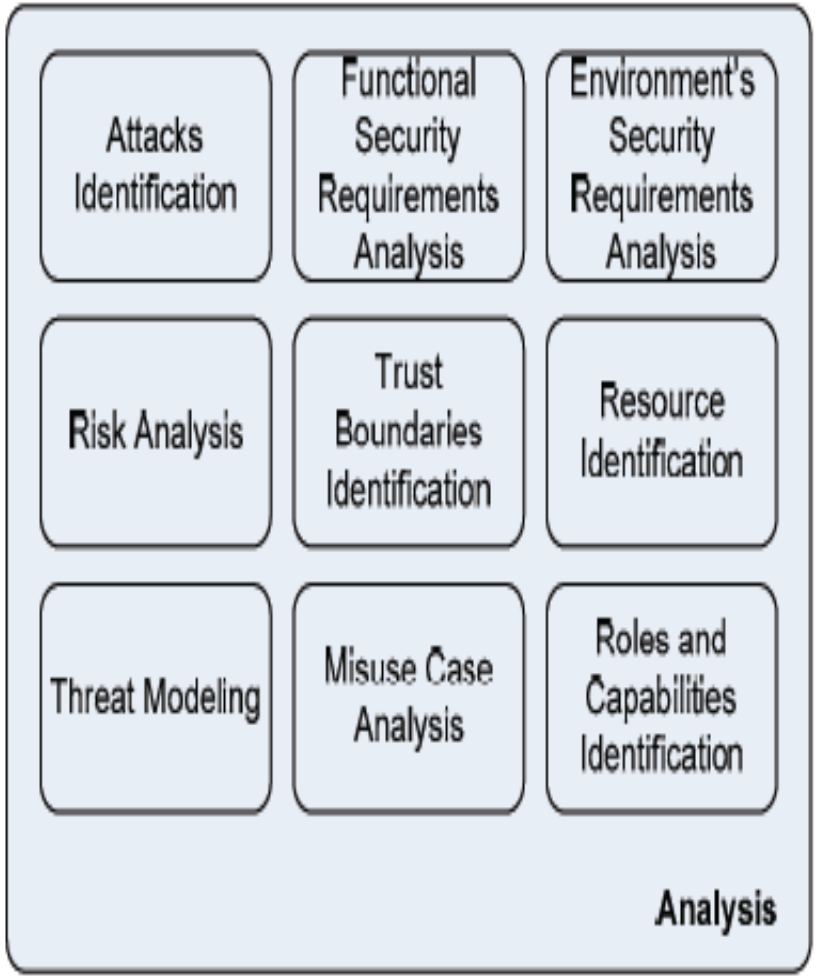
Agile methods are characterized by *small releases, simple design, tests, pair programming, continuous integration, collective ownership, on-site customer, open workspace and 40-hour weeks.*

PROBLEMS & MEASURES

- Some of the agile methods are in direct conflict with secure SDLC processes.
- Eg: Thread modelling – secure SDLC process, conflicts with design principles of agile methods.
- Care should be taken while integrating an agile methodology with a security measure activity.
- It could decrease the agility.
- We present a five-step method to introduce security measures in the software development cycle, published by *Hossein Keramati, Seyed-Hassan Mirian-Hosseinabadi*.

Step 1: Extracting Security Activities

- List activities and sub activities to be performed to reach secure systems.
- Name the list as *security activities*.
- Used as a basis for next steps.
- Classify them.
- Figure represents the security activites in the analysis phase of SDL



Step 2: Calculating Agility degree of Security Activities

- Agility degree of an activity- level of compatibility with agile methodology.
- Grade between 0 and 5. Greater number – higher compatibility, lower grade – conflict.
- 1- column matrix – *Agility degree vector (ADVect)* as shown.
- Agility degree – sum of values in ADVect of an activity.

$$\begin{bmatrix} \text{Simplicity} \\ \text{Customer Interaction} \\ \text{Free of Modeling and Documentation} \\ \text{Change Tolerate} \\ \text{Speed of Execution} \\ \text{Informality} \\ \text{People Orientation} \\ \text{Iterative} \\ \text{Flexibility} \end{bmatrix}$$

Step 3: Integration of Agile and Security Activities

- Analyze agile methodologies and identify their core engines.
- Calculate agility degree for extracted agile activities.
- Compare agility features of extracted activities and security activities.
- Estimate agility degree of a combined activity.
- *Activity Integration Compatibility Matrix (AICM).*

STEP 3: CONTD….

- AICM – table , row – agile activities, column- security activities.
- Compatible activities – 1 inserted, 0- non-integrable security and agile activities.
- Introducing a new parameter: ART - *Agility Reduction Tolerance* , decimal number > 0 and <5.
- *ART* controls insertion of low agile security activity to existing software development process.

STEP 4:
ACTIVITY PROCESS INTEGRATION ALGORITHM

1) Highest agility degree security activity selected.

2) Using AICM, list agile activities integrable with the security activity selected.

   Select heaviest or the lowest agility degree activity.

3) Integrate both the activities to form *newAct* with new agility degree.

4) Replace *newAct* with original activity, recalculate agility degree.

STEP 4: CONTD…

4) *Process agility degree* + *ART* >= *newAct*, integration acceptable. If not, security activity should not be inserted.

5) Remove the selected security activity from the list.

6) Redo if there are security activities in the list.

- ART parameter controls agile nature of the method.

Step 5: Agility Reduction Tolerance

- ART can be tuned for *balancing* between *decreased level of agility* due to security activity and *benefits from developing secure software*.

- With a cost/benefit analysis and cost of agility reduction in development process , ART parameter's value can be calculated to inject security activities in the process.

Conclusion

- Security is a very important aspect of software development.
- Measures can be taken to integrate it in the software development life cycle.
- It is possible to effectively integrate security into agile development as well

DISCLAIMER

- We have covered only a few approaches on how to build secure software
- There's a whole LOT of information available if you Google Software Security !

THE SOURCE OF OUR INFORMATION

- http://www.cigital.com/papers/download/bsi1-swsec.pdf
- http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1193213&userType=inst&tag=1
- http://www.ibm.com/developerworks/linux/library/s-link.html
- http://www.ibm.com/developerworks/library/s-fail.html
- http://www.ibm.com/developerworks/library/s-priv.html
- http://www.ibm.com/developerworks/library/s-simp.html
- http://www.ibm.com/developerworks/library/s-princ5.html
- http://ce.sharif.edu/~mirian/Accepted%20Paper/AICCSA08-keramati.pdf

THANK YOU

QUESTIONS ?