Workshop on re-engineering

A newbie programmer has created a simple dungeon-crawler game called Dragon Treasure. The newbie is quite happy with the look and feel of the game, but wants it to be more flexible. There are a few features the newbie wishes to add, and a few bugs that newbie wants to have fixed. Please help the newbie by re-engineering his code and help him create a better design and code, which is easier to understand. The documentation consists of a drawing of the dungeon. The source code is provided in the `Application.java` file (compile with: javac Application.java run with: java Application).

The newbie wants to fix the following bugs: Potions keep re-spawning after they have been used, and after the Dungeon hero returns back to the room in which the potion was originally located. This is not supposed to happen. Furthermore, the beast and the dragon keep re-spawning with 0 health after they have been defeated if the dungeon hero returns to that room. The dungeon hero seems to also be able to steal the treasure multiple times. Please help the newbie to fix these bugs.

The newbie wishes the software architecture design was better made so that it would be easy to add new rooms, new monsters, new treasure, improved keyboard input functionality, etc. in the future. Your task is to completely re-engineer the code and pay close attention to good software architecture design and coding best practices (keeping in mind the quality attributes: coupling, cohesion, sufficiency, completeness, primitiveness - http://atomicobject.com/pages/OO+Quality). You should design your architecture for *Modifiability* and *Comprehensibility* (make the code easy to read and understand), and utilize code reuse and design-patterns where appropriate. You are of course allowed to add additional classes and change as much of the code as you wish/need, as long as the game plays and looks roughly the same.

Step 1. 30 minutes. Look through the code and the diagram. Discuss within your group about what the different parts do, and create additional diagrams if you think it is necessary in order to follow how the code works and how parts fit together (example: sequence diagrams of parts, state-charts, etc.). Discuss readability and add comments where you think they would be ample. Discuss flaws, features, bugs and bottlenecks. Also discuss coupling, cohesion, sufficiency, completeness, and primitiveness.

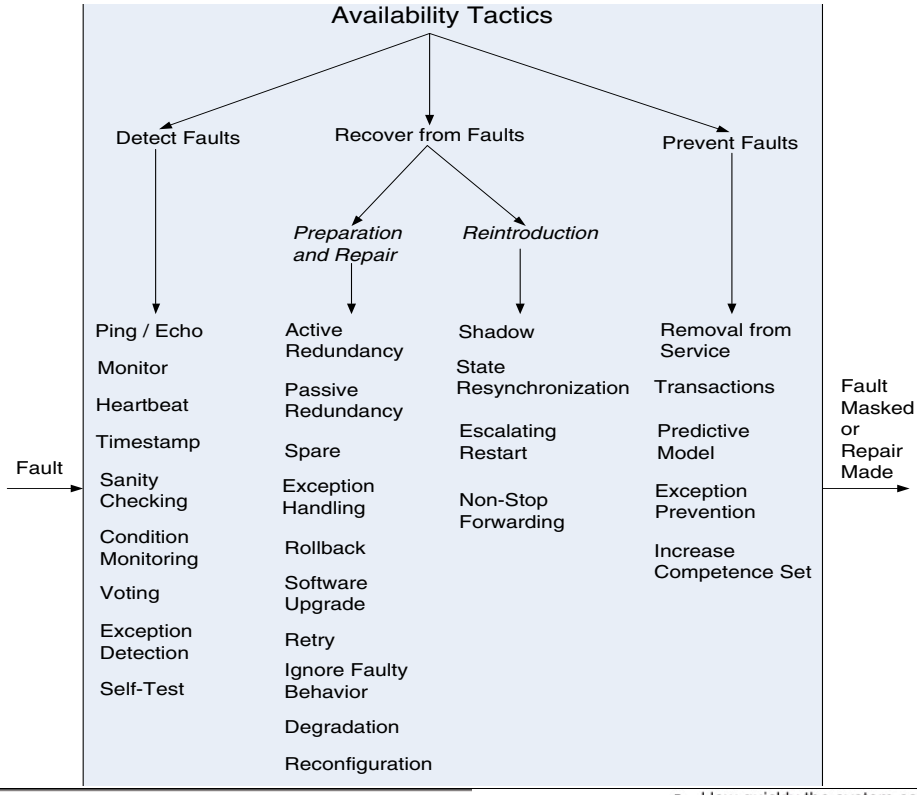
Step 2. 15 minutes. Meet the other group (Group 1+2, Group 3+4, and Group 5+6) and compare notes. Have you come to the same conclusions? Do your diagrams match? If not, why are they different? Try to understand how you may have understood things differently, and why you maybe reasoned about the design differently.

Step 3. 10 minute break, then turn the page

Step 4. 50 minutes. The company wants to ensure that the server remains stable and that it is always available. You have therefore been asked to re-design the software with availability in mind. Decide whether it is best to do a re-engineering at source-code level, function level, architectural level, or at intentions level (motivate your choice). Also decide whether it is necessary to do a big bang re-engineering, or if parts of the old system can be reused. Provide pseudo code and a redesigned architecture, with the necessary documentation, for your new implementation. Remember to use design-patterns and best practices (coupling, cohesion, sufficiency, completeness, primitiveness) where appropriate.

Step 5. 10 minutes. Meet the other group (Group 1+2, Group 3+4, and Group 5+6) and compare notes. How much do your re-engineered designs differ and why? How much did the different scenarios and quality attributes influence design choices you made for your software? Have you utilised the appropriate design-patterns where necessary? Does the re-engineered design follow best practices?

Step 6. Short wrap-up and moving back the furniture.



Category	Checklist
Allocation of Responsibilities	Determine the system responsibilities that need to be high available. Within those responsibilities, ensure that additio responsibilities have been allocated to detect an omission crash, incorrect timing, or incorrect response. Additionally ensure that there are responsibilities to do the following: ▪ Log the fault ▪ Notify appropriate entities (people or systems) ▪ Disable the source of events causing the fault ▪ Be temporarily unavailable ▪ Fix or mask the fault/failure ▪ Operate in a degraded mode
Coordination Model	Determine the system responsibilities that need to be high available. With respect to those responsibilities, do the following: ▪ Ensure that coordination mechanisms can detect an omission, crash, incorrect timing, or incorrect response Consider, for example, whether guaranteed delivery is necessary. Will the coordination work under conditions degraded communication? ▪ Ensure that coordination mechanisms enable the loggin of the fault, notification of appropriate entities, disabling the source of the events causing the fault, fixing or mask the fault, or operating in a degraded mode. ▪ Ensure that the coordination model supports the repla ment of the artifacts used (processors, communication channels, persistent storage, and processes). For exar ple, does replacement of a server allow the system to continue to operate?
Resource Management	▪ How quickly the system can be reinstalled based on t units of delivery provided ▪ How to (re)assign runtime elements to processors, co munication channels, and data stores ▪ When employing tactics that depend on redundancy functionality, the mapping from modules to redundant components is important. For example, it is possible t write one module that contains code appropriate for b the active component and backup components in a pr tection group. Determine what critical resources are necessary to continue operating in the presence of a fault: omission, crash, incorrect timing, or incorrect response. Ensure there are sufficient remaining resources in the event of a fault to log the fault; notify appropriate entities (people o systems); disable the source of events causing the fault be temporarily unavailable; fix or mask the fault/failure; operate normally, in startup, shutdown, repair mode, degraded operation, and overloaded operation. Determine the availability time for critical resources, what critical resources must be available during specified time intervals, time intervals during which the critical resource may be in a degraded mode, and repair time for critical resources. Ensure that the critical resources are available during these time intervals. For example, ensure that input queues are large enough to buffer anticipated messages if a server fails so that the messages are not permanently lost.

	▪ Determine if the coordination will work under conditior of degraded communication, at startup/shutdown, in r pair mode, or under overloaded operation. For exampl how much lost information can the coordination mode withstand and with what consequences?	Binding Time	Determine how and when architectural elements are boun If late binding is used to alternate between components that can themselves be sources of faults (e.g., processes, processors, communication channels), ensure the chosen availability strategy is sufficient to cover faults introduced t all sources. For example: ▪ If late binding is used to switch between artifacts such as processors that will receive or be the subject of fault will the chosen fault detection and recovery mechanism work for all possible bindings? ▪ If late binding is used to change the definition or toler- ance of what constitutes a fault (e.g., how long a proce can go without responding before a fault is assumed), is the recovery strategy chosen sufficient to handle all cases? For example, if a fault is flagged after 0.1 millise onds, but the recovery mechanism takes 1.5 seconds t work, that might be an unacceptable mismatch. ▪ What are the availability characteristics of the late bind ing mechanism itself? Can it fail?
Data Model	Determine which portions of the system need to be highly available. Within those portions, determine which data abstractions, along with their operations or their properties could cause a fault of omission, a crash, incorrect timing behavior, or an incorrect response. For those data abstractions, operations, and properties, ensure that they can be disabled, be temporarily unavaila or be fixed or masked in the event of a fault. For example, ensure that write requests are cached if a server is temporarily unavailable and performed when the server is returned to service.		
Mapping among Architectural Elements	Determine which artifacts (processors, communication channels, persistent storage, or processes) may produce a fault: omission, crash, incorrect timing, or incorrect response. Ensure that the mapping (or remapping) of architectural elements is flexible enough to permit the recovery from t fault. This may involve a consideration of the following: ▪ Which processes on failed processors need to be rea signed at runtime ▪ Which processors, data stores, or communication cha nels can be activated or reassigned at runtime ▪ How data on failed processors or storage can be serv by replacement units	Choice of Technology	Determine the available technologies that can (help) detec faults, recover from faults, or reintroduce failed componen Determine what technologies are available that help the response to a fault (e.g., event loggers). Determine the availability characteristics of chosen technologies themselves: What faults can they recover from? What faults might they introduce into the system?

Workshop on re-engineering

Your company has a product that aggregates a number of different positioning techniques into a single simple-to-use framework. One of the services provided by the framework is a server that accepts incoming position updates from users. The users of this system value their privacy and it is of outmost importance that only the authorized people gain access to these position updates.

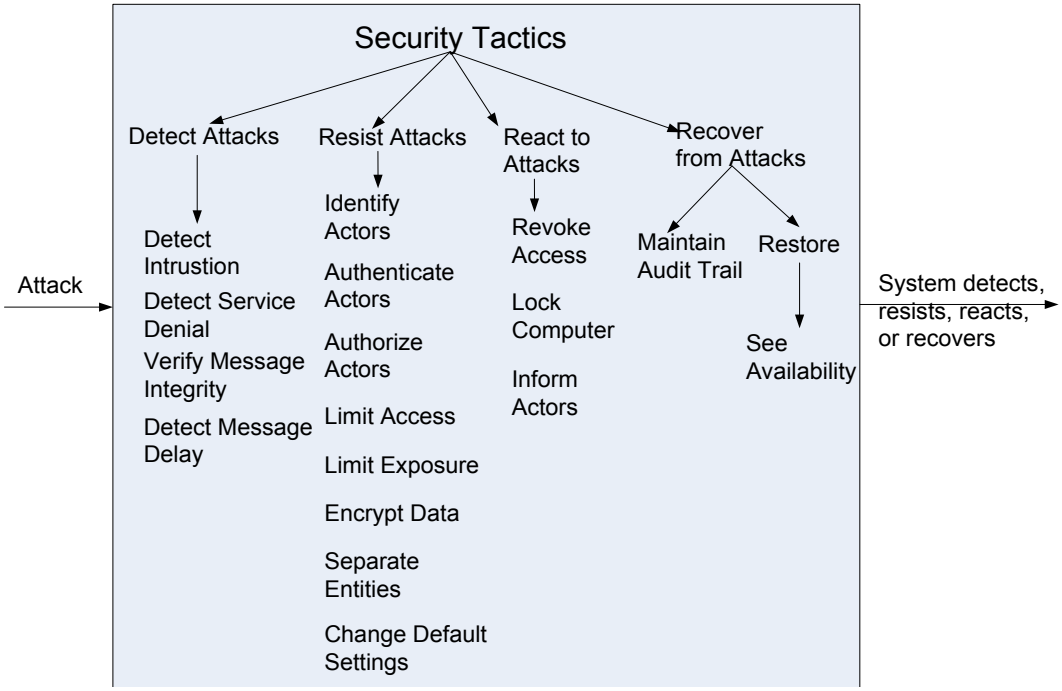
The position updates then be consumed by a number of different services, such as friend-finder applications. The framework, and the server in particular, is a little out-dated and the original creator is no longer around. So, you have been asked to review the code, find the features, bottlenecks, and the flaws of the code, and describe the different parts of the code.

Step 1. 40 minutes. Look through the code and the diagrams. Discuss within your group about what the different parts do, and create necessary additional diagrams to follow how the code works and how parts fit together (example: sequence diagrams of parts, state-charts, etc.). Discuss readability and add comments where you think they would be ample. Discuss flaws, features, bugs and bottlenecks. Also discuss coupling, cohesion, sufficiency, completeness, and primitiveness.

Step 2. 15 minutes. Meet the other group (Group 1+2, Group 3+4, and Group 5+6) and compare notes. Have you come to the same conclusions? Do your diagrams match? If not, why are they different? Try to understand how you may have understood things differently, and why you maybe reasoned about the design differently.

Step 3. 40 minutes. The company wants to improve upon the security of the system, which up to this point has had no security at all. You have therefore been asked to re-design the software with security in mind, including to prevent intrusions etc.. Decide whether it is best to do a re-engineering at source-code level, function level, architectural level, or at intentions level (motivate your choice). Also decide whether it is necessary to do a big bang re-engineering, or if parts of the old system can be reused. Provide pseudo code and a redesigned architecture, with the necessary documentation, for your new implementation. Remember to use design-patterns and best practices (coupling, cohesion, sufficiency, completeness, primitiveness) where appropriate.

Step 4. 15 minutes. Meet the other group (Group 1+2, Group 3+4, and Group 5+6) and compare notes. How much do your re-engineered designs differ and why? How much did the different scenarios and quality attributes influence design choices you made for your software? Have you utilised the appropriate design-patterns where necessary? Does the re-engineered design follow best practices?



Category	Checklist
Allocation of Responsibilities	Determine which system responsibilities need to be secure. For each of these responsibilities, ensure that additional responsibilities have been allocated to do the following: ▪ Identify the actor ▪ Authenticate the actor ▪ Authorize actors ▪ Grant or deny access to data or services ▪ Record attempts to access or modify data or services ▪ Encrypt data ▪ Recognize reduced availability for resources or services and inform appropriate personnel and restrict access ▪ Recover from an attack ▪ Verify checksums and hash values
Coordination Model	Determine mechanisms required to communicate and coordinat with other systems or individuals. For these communications, ensure that mechanisms for authenticating and authorizing the actor or system, and encrypting data for transmission across the connection, are in place. Ensure also that mechanisms exist for monitoring and recognizing unexpectedly high demands for resources or services as well as mechanisms for restricting or terminating the connection.
Data Model	Determine the sensitivity of different data fields. For each data abstraction: ▪ Ensure that data of different sensitivity is separated. ▪ Ensure that data of different sensitivity has different access rights and that access rights are checked prior to access. ▪ Ensure that access to sensitive data is logged and that the l file is suitably protected. ▪ Ensure that data is suitably encrypted and that keys are separated from the encrypted data. ▪ Ensure that data can be restored if it is inappropriately modified.
	For these resources consider whether an external entity can access a critical resource or exhaust a critical resource; how to monitor the resource; how to manage resource utilization; how to log resource utilization; and ensure that there are sufficient resources to perform the necessary security operations. Ensure that a contaminated element can be prevented from contaminating other elements. Ensure that shared resources are not used for passing sensitive data from an actor with access rights to that data to an actor without access rights to that data.
Binding Time	Determine cases where an instance of a late-bound component may be untrusted. For such cases ensure that late-bound components can be qualified; that is, if ownership certificates for late-bound components are required, there are appropriate mechanisms to manage and validate them; that access to late-bound data and services can be managed; that access by late-bound components to data and services can be blocked; th mechanisms to record the access, modification, and attempts to access data or services by late-bound components are in place; and that system data is encrypted where the keys are intentiona withheld for late-bound components
Choice of Technology	Determine what technologies are available to help user authentication, data access rights, resource protection, and data encryption. Ensure that your chosen technologies support the tactics releva for your security needs.

Mapping among Architectural Elements	Determine how alternative mappings of architectural elements that are under consideration may change how an individual or system may read, write, or modify data; access system services or resources; or reduce availability to system services or resources. Determine how alternative mappings may affect the recording of access to data, services or resources and the recognition of unexpectedly high demands for resources. For each such mapping, ensure that there are responsibilities to do the following: ▪ Identify an actor ▪ Authenticate an actor ▪ Authorize actors ▪ Grant or deny access to data or services ▪ Record attempts to access or modify data or services ▪ Encrypt data ▪ Recognize reduced availability for resources or services, inform appropriate personnel, and restrict access ▪ Recover from an attack	
Resource Management	Determine the system resources required to identify and monitor a system or an individual who is internal or external, authorized or not authorized, with access to specific resources or all resources. Determine the resources required to authenticate the actor, grant or deny access to data or resources, notify appropriate entities (people or systems), record attempts to access data or resources, encrypt data, recognize inexplicably high demand for resources, inform users or systems, and restrict access.	