۲-۱- انگیزه و اهداف پروژه

در سند پیشنهادی که برای پروژه ارائه دادیم، قرار بود آن را پروژه‌ایی در مورد ریزهسته‌ها معرفی کنیم. تاکید کردیم که مـــحـصـول کار این پروژه در نهایت، اگر کاملا خوب پیش برود، یک ریزهسته نسل سوم خواهد بود که بصورت صوری صحت‌سنجی شده است. در اینجا اهداف پروژه را در حالت ایده‌آل، و آن‌طور که می‌خواستیم به ترتیب اهمیت ذکر می‌کنیم:

۱- طراحی شئ‌گرا یک ریزهسته نسل سوم،
۲- پیاده سازی ریزهسته طراحی شده،
۳- صحت‌سنجی صوری ریزهسته (این که ریزهسته پیاده‌سازی شده دقیقا همان چیزی را پیاده سازی می‌کند که طراحی کرده‌ایم).

البته تاکید کرده بودیم که برای اینکه به هدف سوم نائل شویم، طراحی در مرحله اول باید فاقد ابهام باشد، فلذا باید تا حد امکان از استفاده از زبان طبیعی پرهیز کنیم، و از زبان‌های صوری همچون OCL به عنوان مکمل طراحی خود در دیاگرام‌های UML استفاده کنیم.

در سند پیشنهاد ذکر کرده بودیم مطالعات چندین ماهه ما نشان می‌دهد که تا بحال هیچ ریزهسته شئ‌گرایی طراحی نشده است که *بصورت صوری صحت‌سنجی* شود، بنابراین برایمان جالب بود که این کار جدید را انجام دهم. لازم به ذکر است که امروزه توجه اصلی در عرصه ریزهسته‌ها، تولید ریزهسته‌هایی با درجه قابلیت اطمینان بسیار بالا می‌باشد (پروژه‌هایی نظیر EROS شاهد این مدعا هستند) [WIKB11] و [DRDB11]. این موضوعی کاملاً روشن می‌باشد که طراحی درست (طراحی که ما را از پیچیدگی دور کند) و بدون ابهام (عدم استفاده از زبان طبیعی تا حد امکان)، راه حل رسیدن به این هدف است؛ بنابراین طـراحی به روش شئ‌گرا را در کنار روش‌های صوری انتخاب کرده‌ایم.

پروفسور Heiser و بقیه همکارانش در مقاله‌ایی که چند ماه قبل از ارائه سند پیشنهاد پروژه منتشر کرده بودند، آخرین دستاوردهای خود در زمینه ریزهسته‌ها را معرفی کردند؛ آن‌ها نوشتند که: «طراحان سیستم عامل ترجیح می‌دهند از روش‌های سنتی از قبیل طراحی پایین به بالا استفاده کنند،

این در حالی است که مهندسین نرم‌افزار ترجیح می‌دهند که با یک رویکرد مدل‌گرا مسائل را حل کنند؛ به همین دلیل ما از یک روش بین این دو دیدگاه استفاده خواهیم کرد و آن هم این است که از یک زبان Pure Functional که به مدل نزدیک است (و راحت‌تر می‌توان در مورد برنامه‌های نوشته شده در آن بحث کرد) برای تولید نمونه‌های (Prototype) پـروژه خود استفاده می‌کنیم». آن‌ها از یک متد جدید Prototyping استفاده کردند، که ابتدا یک نمونه در زبان Haskell ساخته، سپس آن را پالایش و نهایتاً به کد زبان C تبدیل می‌کردند؛ البته در هر مرحله از پالایش، بصورت صوری اثبات می‌کردند که نمونه پالایش شده در اصل همان نمونه اصلی می‌باشد و همان ویژگی‌ها و رفتارها را دارد.

با استناد به سند پیشنهاد قرار شد که ما در این طراحی با رویکردی دیگر به مسئله نگاه کنیم، و نشان دهیم که *می‌توان ریزهسته‌ایی را با تکیه صرف بر مدل طراحی کرد*؛ لذا تصمیم گرفتیم ابتدا مدلی از سیستم عامل مورد نظر خود بسازیم، سپس با استفاده از پالایش صوری آن را به وسیله کد زبان C++ پیاده‌سازی کنیم. بدین ترتیب سیستم عاملی خواهیم داشت که تماماً از روی مدل طراحی شده است، و می‌توان ادعا کرد که این طراحی بسیار قابل حمل‌تر، قابل نگهداری‌تر، و ساده‌تر از طراحی‌های موجود است. هر تغییر در مدل‌های شئ‌گرا، که با استفاده از ابزار UML و OCL بیان خواهند شد، می‌تواند به سادگی و با کمترین عوارض جانبی انجام شود؛ چرا که این نحوه مدل‌سازی بسیار ساده‌تر و خواناتر از کد زبان Haskell خواهد بود، و همچنین تغییر در مدل‌های شئ‌گرایی که خوب مهندسی شده‌اند کم هزینه است (یکی از مزیت‌های اصلی طراحی شئ‌گرا). بررسی این مدل‌ها به دلیل خوانایی، سادگی، و خودتوصیفی بـــودن دیـاگـرام‌های UML، بسیار ساده‌تر می‌باشد. درضمن، همانطور که می‌دانیم دیاگرام‌های UML بخوبی می‌توانند سیستم مورد مطالعه را از دیدگاه‌های مختلف مدل کنند. نهایتاً می‌توان گفت که با در دست داشتن یک مـــدل که تا یک سطح معینی پالایش شده است، می‌توان برای هر سخت افزار در زمان دلخواه پیاده‌سازی مخصوص آن‌را داشته باشیم.

بر خلاف ریزهسته‌های نسل دوم که برای کامپیوترهای شخصی حرفه‌ایی (با میزان حافظه بسیار زیاد) طراحی شده بودند، ریزهسته‌های نسل سوم بیشتر برای کاربردهای نهفته (Embedded) در نـظر گرفته شده‌اند. این امر باعث شده است که بیشتر از اینکه در طراحی به Footprint حافظه نهان توجه

کنند، به Footprint حافظه اصلی توجه نمایند؛ البته توجه به Footprint حافظه نهان در صورتی که از یک سطح بالاتر (مثلا کد Haskell) شروع به ساخت سیستم نماییم، تقریباً غیرممکن است. بنابر اعتراف پروفسور Heiser این یاری شانس بوده که ریزهسته seL4 دچار همان مشکلی نشد که ریزهسته‌های نسل اول به آن دچار شدند. به نظر من این تجربه بالای برنامه نویسی تیم NICTA، و نیز آگاهی آن‌ها از مشکلات ریزهسته‌های نسل اول را نشان می‌داده که دچار مشکل استفاده نادرست از حافظه نهان نشدند. من معتقدم که در زمان پیاده‌سازی در سطوح پایین‌تر می‌توان تا حد قابل قبولی ملاحظات حافظه نهان را داشت، چرا که تجربه برنامه نویسی ما نشان می‌دهد که پیاده سازی درست حلقه‌های تکرار و ساختارهای شرطی می‌تواند در این کار مؤثر باشد.

در پایان این بخش لازم می‌بینیم که سایر دستاوردهای این پروژه را که انتظار می‌رفت که با نائل شدن به اهدافی که پیش‌تر ذکر شد، برسیم را بیاوریم. گفته بودیم که اگر بتوانیم به تمامی اهداف اصلی این پروژه دست پیدا کنیم دستاوردهای زیر را خواهیم داشت:

- خواهیم دید که می‌توان RUP را چنان تغییر داد که مناسب برای طراحی یک ریزهسته نسل سوم شود،
- خواهیم دید که می‌توان یک ریزهسته نسل سوم را فقط مبتنی بر مدل طراحی کرد، و مدل را می‌توان مرحله به مرحله پالایش و نهایتاً پیاده‌سازی کرد،
- خواهیم دید که طراحی و مدل‌سازی شئ‌گرا چگونه از پیچیدگی مسئله می‌کاهد.

روشن است که این نتایج برای مهندسی نرم‌افزار، و بخصوص برای توسعه‌دهندگان سیستم‌عامل‌ها بسیار مهم و ارزشمند می‌باشد. در ادامه و در فصول بعدی خواهیم دید که تقریباً به تمامی اهداف مورد انتظار پروژه دست یافته‌ایم.

۲-۲- مدل فرآیند پروژه

همانطور که از بخش‌های قبل می‌توان دریافت، مدل فرآیند مورد استفاده در این پروژه، RUP می‌باشد. همانطور که می‌دانیم RUP فراتر از یک مدل فرآیند ساده می‌باشد، این در اصل یک چهارچوب فرآیند می‌باشد؛ بدین معنی که تمامی ویژگی‌های فرآیندهای کلاسیک در آن قرار دارد، و همچنین می‌توانیم آن را متناسب با نیاز خود تغییر دهیم.

ممکن است سوالی پیش آید که آیا اصلاً RUP برای استفاده در پروژه‌های دانشگاهی و یا پروژه‌های تک نفری مناسب است؟ پاسخ این سوال را می‌توان در [RUPE06] یافت که در آن نویسنده مثالی از یک پـروژه (با نـــام Deimos) آورده است کـه توسط یک نفر انجام می‌شود، در این مثال مهندس نرم‌افزار با گفت‌گوی مفصل با مشتری هر یک از فازهای RUP را آنگونه که توسط این چهارچوب توصیه شده طی می‌کند؛ وی در هر زمان توجه به ریسک‌هایی که پروژه را تهدید می‌کند، دارد، و غیره. با وجود اینکه این پروژه، حتی در این مثال، برای طراحی یک سیستم نرم‌افزاری برای یک مشتری بوده، ما معتقد هستیم از RUP می‌توان در پروژه‌های آکامیک هم استفاده کنیم. طبق [RUPE06] چهارچوب RUP یک سفره پُر از غذاهای متنوع (در اصل توصیه‌ها و ابزارهای متعدد برای مشکلات متعدد) برای ما فراهم می‌آورد که استفاده از تمامی غذاهای موجود در این سفره کاری عملی و یا حداقل عاقلانه‌ایی نمی‌باشد، پس بنابراین ما نیز باید تنها از آن ابزارهایی از RUP استفاده کنیم که آن‌ها را لازم داریم. ما در این پروژه نیازمند در نظر داشتن [بطور دائمی] ریسکهای پروژه هستیم، و همچنین مدل فرآیند این پروژه باید بگونه‌ایی باشد که پروژه در برابر تغییرات انعطاف پذیر و با هزینه زمانی کمی قابلیت تکامل را داشته باشد.

در این بخش برآنیم که نشان دهیم چگونه RUP را متناسب با نیاز خود تنظیم کرده‌ایم. دو تغییر در ساختار دوبعدی RUP صورت گرفته است:

۱- مراحل Inception و Transition کمرنگ‌تر، و مراحل Elaboration و Construction بصورت *تکراری* و خیلی طولانی‌تر[۱]

۲- تعریف Discipline های *اصلی* و *پشتیبان* متناسب با نیازهای خود.

مرحله Inception، مرحله اثبات درک صورت مسئله، و توجیه ادامه پروژه است؛ مرحله Transition نیز مرحله انتقال محصول به محیط عملیاتی، تست بتا، و آموزش است. با توجه به این که این پروژه یک پروژه تحقیقاتی است، صورت مسئله خوب تعریف شده و کاملاً مشخص است، و همچنین ادامه پروژه هم اینگونه توجیه می شود که نتایج ارزشمند علمی این پروژه باعث می‌شود که ما این هزینه را متحمل شویم. در مرحله Inception مربوط به این پروژه، لازم نیست که با هیچ مشتری گفت‌گو

[۱] لازم به ذکر است که با اینکه ما در این پروژه مراحل Elaboration و Construction را با تکرارهای زیادی اجرا نکردیم، ولی RUP این امکان را به ما می‌دهد که با طراحی صحیح قابلیت تکرار به منظور متکامل‌تر کردن پروژه داشته باشیم.

شود، چرا که نیازهایی را که سیستم باید با توسعه خود برآورده کند، کاملاً مشخص است: *می‌دانیم که سیستم باید یک ریزهسته نسل سوم باشد*. البته این نیازمندی خیلی غیردقیق (زبانی) و کلی بیان شده است، اما می‌دانیم که نیازی به تعریف دقیق‌تر نیست، زیرا نیازمندی‌هایی که یـــک سیستم‌عامل و علی‌الخصوص یک ریزهسته باید برآورده کند را کاملاً می‌دانیم، بنابراین این کار را با آوردن یک توصیف جامع Use-Case در سند معماری انجام خواهیم داد.

از آنجایی که این سیستم به روش فرمال طراحی، مدل‌سازی، و طی مراحل پالایش صوری پیاده‌سازی شده است، مطمئن هستیم کـه در حـین طراحی و ساخت، تمام رفتارهای ممکنـــی را که سیستم می‌تواند از خود بروز دهد را در نظر گرفته‌ایم، و محصول نهایی دقیقا همـان چیزی را پیاده سازی می‌کند که در مراحل Inception و Elaboration توصیف آن را آورده بودیم. در ضمن در این پروژه هیچ محیط عملیاتی موجود نیست که ما محصول خود را به آن منتقل کنیم، و نیز هیچ مشتری وجود ندارد که ما محصول خود را به او آموزش دهیم. البته در اینجا بحثی مطرح می‌شود که علاوه بر گزارش نهایی پروژه می‌بایست یک مرجع برای توابع API سیستم و نحوه برنامه‌نویسی تحت این سیستم عامل تهیه شود؛ در جواب به این خواسته می‌توان گفت که گزارش نهایی ما، نحوه پیاده سازی سیستم را نشان می‌دهد، و تهیه مرجع برای برنامه‌نویسی تحت این سیستم عامل موضوع دیگری است که نباید در اینجا به آن پرداخت.

مرحله Elaboration یکی از مهم‌ترین و طولانی‌ترین مراحل در فرآیند پروژه‌ی مورد نظر ما است. در این مرحله، تاکید اصلی بر روی تحلیل و علی‌الخصوص طراحی می‌باشد، و همانطور که قبلاً ذکر کردیم در این پروژه تاکید اصلی بر روی مدل است و پیاده‌سازی نهایی را با پالایش صوری مدل باید بدست آوریم؛ فلذا عمل مدل‌سازی باید خیلی با دقت و درست انجام شود؛ از آنجایی که RUP یک فرآیند حول معماری (Architecture Centric) می‌باشد، می‌توان این را به عنوان یکی از دلایلی که RUP را به عنوان مدل فرآیند انتخاب کردیم ذکر کنیم. این مرحله بصورت *تکراری* باید باشد؛ در این پروژه در یک تکرار، مدل‌سازی با استفاده از UML و در تکرار دیگر، تکمیل و دقیق‌تر کردن آن با استفاده از OCL انجام گرفت. در یک تکرار دیگر از مرحله Elaboration با توجه به چهارچوبی که در این پروژه آن را معرفی کرده‌ایم، مدل UML خود را که در تکرارهای قبل تکمیل‌تر و دقیق‌تر شده

بود را به مدل B ترجمه کردیم. در پایان این مرحله سند معماری را تهیه کردیم که در مراحل بعد نقشه راهمان بود.

در اینجا لازم می‌بینم که دلیل استفاده از OCL و مشکلاتی که می‌تواند بوجود آورد را ذکر کنم. در اصل OCL یک زبان برای توصیف دقیق و صوری می‌باشد (لازم است یادآوری کنم که این زبان خیلی شبیه به زبان AMN می‌باشد، و علاوه بر آن هر دو دارای معناشناخت دلالتی صوری و کامل می‌باشند–این بدین معنی است که این دو زبان می‌توانند بدون بوجود آمدن هیچ مشکلی به هم ترجمه شوند[۲]) که فقط برای استفاده در مدل‌سازی به وسیله UML در نظر گرفته شده است. این زبان در حقیقت مدل‌های UML را دقیق‌تر می‌کند؛ از طرفی هیچ مکانیزمی برای پالایش این زبان به زبان‌های سطح پایین‌تر مثل C++ و یا OCaml، و حتی پالایش آن به مدل‌هایی با جزئیات بیشتر وجود ندارد، این در حالی است که B-Method روش‌هایی را برای پالایش توصیف تا هر سطح دلخواهی از تجرید را فراهم آورده است. در حقیقت اگر ما پیاده‌سازی را مستقیماً از روی UML+OCL انجام دهیم، اثبات این موضوع که این پیاده‌سازی دقیقاً همان رفتاری را دارد که مدل توصیف آن را کرده است مشکل و حتی شاید غیر ممکن خواهد شد.

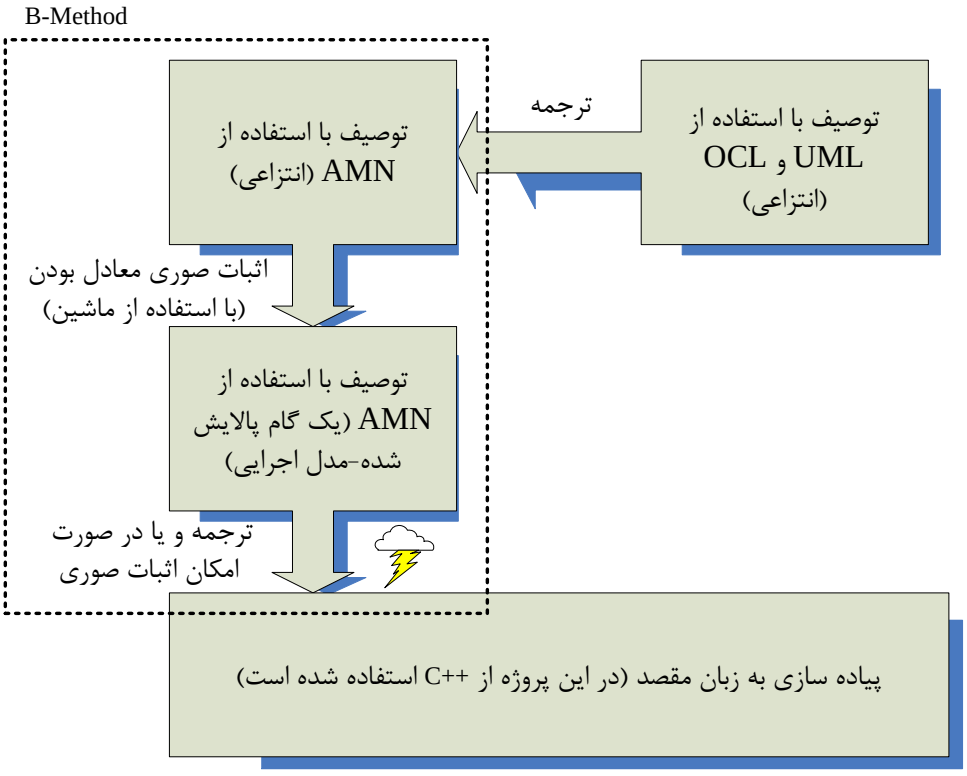
در سند پیشنهاد پروژه، پیشنهاد دادیم که ابتدا مدل‌سازی را با استفاده از UML و OCL انجام دهیم و سپس آن را به AMN تبدیل کنیم، بعد از آن می‌توانیم آن را یک مرحله بیشتر پالایش کنیم (البته این پالایش باید در مرحله Construction صورت پذیرد). بدین ترتیب می‌توانیم مدلی از برنامه اجرایی هم داشته باشیم، که با در دست داشتن این مدل بسیار راحت‌تر می‌توان پیاده سازی را انجام داد، و حتی اگر معناشناخت دلالتی زبان مقصد را داشته باشیم، می‌توانیم بطور صوری اثبات کنیم که پیاده‌سازی در حقیقت همان رفتاری را خواهد داشت که قبلاً در طی فرآیند مدل‌سازی همه آن‌ها را پیش بینی کرده‌ایم. در شکل صفحه بعد نموداری را آورده‌ایم که فرآیند مدل‌سازی و پالایش را که در دو مرحله از RUP صورت می‌گیرد را نشان می‌دهد. دقت کنید که علامت صاعقه در نمودار شکل ۱ بدین معنا است که احتمال به وجود آمدن خطا در این مرحله وجود دارد، چرا که زبان C++ فاقد معناشناخت دلالتی است و معنای آن بصورت زبانی توسط طراح آن ذکر شده است، ولی از آنجایی

[۲] البته جالب است بدانید که آزمایشگاه LORIA نیز اخیراً مقالاتی را در زمینه یکپارچه‌سازی OCL و AMN منتشر کرده است؛ در این مقالات رویکرد ترکیب دو روش مدل‌سازی UML و B بسیار توصیه شده است [LEDA01]، [LEDB01]، و [LEDC01].

که یک گـام بیشتر عمـل پالایش را انجام داده‌ایم احتمال بروز این خطـا بسیـار ناچیز می‌باشد، و می‌توانیم از آن صرف‌نظر کنیم.

مرحله بعد، اولین مرحله فرآوری می‌باشد و Construction نام دارد؛ در این مرحله عمل پالایش صوری و پیاده‌سازی در C++ انجام خواهد شد. همانطور که می‌دانیم، عمل پالایش مرحله‌به‌مرحله، و نیازمند تکرار می‌باشد. همانطور که در ادامه گزارش پروژه خواهیم دید به دلیل کمبود وقت موفق به پیاده‌سازی کامل پروژه نشدیم و در مرحله Construction تا حدی عمل پالایش و فرآوری را انجام دادیم و برای پیاده‌سازی صرفاً راهکاری را نشان دادیم.

طی تغییری که در مدل فرآیند RUP صورت پذیرفت، برخی از Discipline های اصلی و پشتیبان از آنچه که در کتابها برای سیستم‌های همه‌منظوره پیشنهاد شده است، متفاوت شد؛ برخی از آنها حذف، و تعدادی از بیرون به آنها اضافه شده است.



شکل ۱ – فرآیند اصلی مدل‌سازی، و پالایش صوری، و نهایتاً پیاده سازی پروژه.

بطور کلی، می‌توان گفت که فرایند RUP مورد استفاده ما دارای این Discipline ها می‌باشد:

- اصلی:

- نیازمندی‌ها،
- تحلیل و طراحی،
- پالایش،
- پیاده‌سازی،
- ارائه؛

- پشتیبان:
  - مدیریت پروژه،
  - مدیریت تغییرات و پیکربندی،
  - محیط.

حال نوبت آن رسیده که توضیح مختصری در مورد اصطلاحات استفاده شده در لیست قبل ارائه دهیم. به این Discipline ها، *ساختارهای ایستا* هم می‌گویند، و در چند پاراگراف آینده به منظور سادگی، از واژه ساختار استفاده خواهیم کرد. ساختار نیازمندی‌ها در حقیقت با مطالعه کارهای گذشته، و مرتبط، و نیز با رجوع به دانش خود از سیستم عامل‌ها می‌تواند انجام شود. در این پروژه برای جمع آوری نیازمندی‌ها به مستندات سیستم‌عامل‌های L4، seL4، CHORUS، و به مقدار کمی به کتاب [TANE97] که در مورد پیاده‌سازی سیستم‌عامل MINIX است رجوع کردیم.

ساختـــار تحلیل و طراحی اصلی‌ترین ساختار در فرآیند RUP ما می‌باشد؛ در اینجا به مدلسازی بخش‌های مختلف سیستم در UML و ترجمه آن به مدل B پرداختیم. محصول این مرحله سند معماری بود که با تهیه آن به میزان قابل توجهی از ریسک‌های پروژه کاستیم.

ساختار پالایش، ساختاری می‌باشد که در طی تغییر فرآیند RUP، به آم اضافه کردیم. این ساختار دارای بیشترین تاکید، در گذر از مرحله *مهندسی* (فازهای Inception و Elaboration) به مرحله *فرآوری* (فازهای Construction و Transition) است. این ساختار با به کار گیری متد B اقدام به پالایش مدل، و توصیف عملیات کلاس‌ها تا یک حد معین از تجرید، می‌نماید. در این پروژه به دلیل کمبود وقت نتوانسیم عمل پالایش را بطور کامل انجام دهیم و تنها به پالایش بخش‌هایی از سیستم اکتفا کردیم.

ساختار بعدی ساختار پیاده‌سازی می‌باشد، این ساختار تماماً در فاز Construction قرار دارد و تقریباً بطور موازی با ساختار پالایش انجام می‌شود. در این مرحله می‌بایست مدل و طراحی پالایش شده، به زبان C++ پالایش شود. یک اثبات از اینکه کد زبان C++ دقیقاً همان چیزی را پیاده سازی می‌کند که مدل و طراحی پالایش شده آن را توصیف می‌کند نیز بسیار مفید خواهد بود، البته این کار نیازمند دانش کافی از معناشناخت دلالتی یا همان Denotational Semantics زبان C++ دارد که در دسترس نیست. می‌توان این مرحله را بصورت غیرصوری ولی با دقت فراوان پالایش کرد، یا حتی می‌توان از سایر زبان‌ها مثل Java یا OCaml استفاده کرد، ولی از آنجایی که این زبان‌ها بر مبنای Garbage Collection هستند (که این برای کاربردهای Real-time مناسب نیست [KLEI09])، و نیز سیستم Run-time برای این زبان‌ها ممکن است دارای ایراد[۳] باشد، خوب نیست از این زبان‌ها استفاده شود. بدین ترتیب تنها نقطه‌ایی که ممکن است موجب ورود خطا و نقص به سیستم شود، مرحله پالایش به زبان C++ است. البته گذشته از همه این‌ها در این پروژه ما موفق به پیاده‌سازی و حتی پالایش کامل سیستم نشدیم که این هم به علت کمبود وقت بوده است؛ در این پروژه بخش‌هایی از سیستم را پالایش و راهکارهایی را هم برای پیاده‌سازی ارائه دادیم.

در ساختار ارائه، همانطوری که از نامش برمی‌آید، ارائه کتبی سیستم طراحی شده، آماده شد.

ساختارهای پشتیبان ذکر شده در فهرست چند صفحه قبل، همان ساختارهای پیشنهادی RUP می‌باشد، بنابراین به توضیحی مختصر بسنده می‌کنیم. لازم است بدانیم بنا به طبیعت تحقیقاتی بودن این پروژه و نیز فردی بودن آن، این ساختارها قدری کم اهمیت هستند. این ساختارها همگی از نوع Umbrella Task بوده و در تمام فازهای RUP، و به موازات بقیه فعالیت‌ها انجام می‌شود. مهمترین آنها ساختار مدیریت پروژه می‌باشد؛ این ساختار وظیفه دارد که ریسک‌های پروژه را شناسایی و آنها را مدیریت کند. این موضوع برای پروژه‌ی پر از ریسک ما بسیار با اهمیت بود، زیرا هر لحظه ممکن بود که در وسط کار متوجه شویم که از عهده کار بر نمی‌آییم[!]، و اینجاست که مدیریت ریسک به داد ما خواهد رسید. به همین خاطر هم قدری محافظه کارانه عمل کردیم و در ابتدا و قبل از شروع پروژه قدری کاهش حوزه (با توجه به اینکه به هدف پروژه هم بطور کامل برسیم) انجام دادیم.

[۳] البته این حرف قدری افراطی می‌باشد، و منظور ما این است که چون سیستم Run-time این زبان‌ها بطور صوری صحت‌سنجی نشده‌اند، قابل اطمینان نخواهند بود [KELI09].

مدیریت تغییرات و پیکربندی هم در این پروژه مهم است، چرا که تغییر برای جلوگیری از ریسک‌ها باید مدیریت شود، و نیز باید تمامیت دستاوردهای در حال تکامل حفظ شود. ساختار بعدی Environment یا محیط می‌باشد که طی این فعالیت ابزارها و منابع لازم برای توسعه پروژه تهیه شدند.

۲-۳- مدیریت ریسک

در این بخش قصد داریم در مورد موضوع بسیار مهم مدیریت ریسک بحث کنیم. در اینجا قصد نداریم یک سند Risk Management Plan ارائه دهیم، و صرفاً چند مورد از ریسک‌هایی را که پروژه را تهدید می‌کنند را ذکر خواهیم کرد؛ برای هر مورد، راه حل عاقلانه لازم را نیز خواهیم آورد. لازم به یادآوری است که این موارد را عیناً در سند پیشنهاد پروژه نیز ارائه داده بودیم، در اینجا هر یک از این موارد را دوباره ذکر کرده و می‌گوئیم که چنین پیشبینی‌هایی که قبل از شروع پروژه صورت گرفته بودند تا چه حد صحت داشته‌اند. در صفحه بعد جدول ریسک را آورده‌ایم.

همانطور که مشاهده می‌کنیم بیشتر ریسک‌ها مربوط به نبود دانش کافی از یک موضوع خاص بوده است. برای همین تصمیم گرفتیم در دوره سه ماهه مطالعه و تحقیق بطور بسیار فشرده و جدی به مطالعه و کسب مهارت بپردازیم. اولین مورد از ریسک‌های پروژه دقیقاً همان چیزی بود که اتفاق افتاد؛ در این پروژه ما دانش کافی از پالایش و توصیف صوری را (در حد نیاز) کسب کردیم، و حتی روشی را هم برای تبدیل مدل‌های UML ارائه دادیم، اما به دلیل نبود وقت کافی نتوانستیم آن را بطور کامل پیاده‌سازی کنیم، بنابراین با قدری کاهش حوزه پروژه را به طراحی و صحت سنجی صوری یک ریزهسته نسل سوم تقلیل دادیم و برای پیاده‌سازی و پالایش به ارائه راهکاری اکتفا کردیم. خوشبختانه در طول سه ماه مطالعه دانش کافی از متد B و نیز از مدلسازی به زبان UML را توانستیم کسب کنیم، و نیز در طول مدت انجام پروژه نیز زبان OCL و نیز نحوه استفاده از نرم‌افزار AtelierB را فراگرفتیم، بدین ترتیب توانستیم بخش‌های اصلی پروژه را در موعد مقرر به اتمام برسانیم. مورد سوم هم به علت کاهش حوزه در ابتدای انجام کار هیچ وقت به وقوع نپویست، و نهایتاً مورد چهارم هم با مطالعه چندین کتاب در مورد مدلسازی شئ‌گرا و UML برطرف شد.

جدول ۱- ریسک‌های پیش‌بینی شده قبل از شروع پروژه

ردیف	عنوان	احتمال وقوع	تاثیر	اقدامات لازم
۱	دانش کافی از پالایش صوری، و تحلیل و طراحی وجود دارد، ولی فرصت برای پیاده‌سازی موجود نیست	متوسط	بحرانی	در این صورت می‌توان پروژه را به طراحی و تحلیل شئ‌گرا، و صحت‌سنجی صوری یک ریزهسته تقلیل دهیم؛ بدین ترتیب پروژه ما فاقد پیاده‌سازی و محصول عملیاتی خواهد بود
۲	نبود دانش و یا مهارت کافی از نحوه پالایش و توصیف صوری	متوسط	فاجعه بار	در این صورت بهترین اقدام این است که یک ریزهسته شئ‌گرا را بدون پالایش و توصیف صوری پیاده‌سازی کنیم؛ البته صوری‌سازی این پالایش هم درجاتی دارد، که در این حالت می‌توانیم خیلی غیرفرمال‌تر عمل کنیم
۳	نبود دانش کافی از امنیت اطلاعات	بالا	بحرانی	در این صورت باید حوزه مسئله را کاهش دهیم؛ به عبارتی دیگر به جای طراحی یک ریزهسته نسل سوم، یک ریزهسته نسل دوم طراحی کنیم.
٤	نبود دانش کافی از طراحی و تحلیل شئ گرا	پایین	فاجعه بار	در این صورت باید سریعا موضوع پروژه تغییر داده شود و یک ریزهسته ساده با استفاده از دانش فعلی خود طراحی کنیم

۲-۴- نتیجه‌گیری

اصلی‌ترین موضوع این فصل بررسی انگیزه‌ها و اهداف پروژه بود. گفتیم که ارائه روشی جدید برای استفاده از RUP، ترجمه مدل‌های شئ‌گرا به مدل‌های B به منظور تسریع توسعه ریزهسته‌های نسل سوم، و نهایتاً بدیع بودن ایده پروژه اصلی‌ترین انگیزه‌های انجام این پروژه بودند. اهداف این پروژه هم ارائه یک مدل فرآیند مبتنی بر RUP، ارائه چهارچوبی برای ترجمه مدل‌های UML به B به منظور تسریع توسعه ریزهسته‌های نسل سوم، و نهایتاً و اصلی‌ترین هدف این بود که یک ریزهسته نسل سوم را به این شیوه طراحی، صحت سنجی صوری، و نهایتاً پیاده‌سازی کنیم. همانطور که ذکر شد در این پروژه ما موفق به پالایش کامل و پیاده‌سازی محصول خود نشدیم، ولی از آنجایی که در سند پیشنهاد چنین ریسکی را به عنوان یک تهدید برای انجام کامل پروژه ذکر کرده بودیم، خللی در صحت انجام این پروژه وارد نمی‌کند؛ البته ما روش و راهکاری را نیز برای پالایش و پیاده‌سازی پروژه ارائه دادیم، و حتی بخش‌هایی از آن را نیز پیاده سازی کردیم.

در این فصل همچنین مشاهده شد که چگونه RUP را متناسب با نیاز خود تغییر دادیم. در انتهای فصل نیز مروری بر ریسک‌هایی داشتیم که در سند پیشنهاد پروژه آن‌ها را ذکر کرده بودیم و در اینجا با ذکر دوباره آن‌ها میزان دقت هر یک را بیان کردیم، و دیدیم که برخی از آن‌ها اتفاق افتاد و برخی نیز اصلاً به وجود نیامد. البته ما در مواجهه با هر یک از این مشکلات آن‌ها را با کمترین هرینه مدیریت کردیم، و مانع از آن شدیم که ما را از انجام صحیح و کامل پروژه باز بدارند.